Proyecto de Grado

Investigación y análisis de plataformas CMS/WCMS y su impacto en las etapas de desarrollo

Montevideo, Uruguay
Año: 2017

Tutor: Jorge Corral
Federico Mujica
Martin Santagata

UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

Resumen

Las computadoras fueron construidas para procesar datos. Los datos consisten en pequeños fragmentos de información con parte de su significado humano extraído y muchas veces carecen del mismo. Como los datos, el contenido también es información, pero es muy importante su significado humano y contexto.
En términos resumidos el contenido es información producida por un proceso editorial y tiene como único cometido ser consumido por personas vía publicación.

Hoy en día las organizaciones compiten para brindar un servicio cada vez más personalizado a sus clientes, adaptando y creando el contenido ofrecido a sus clientes de forma casi instantánea. Las exigencias de los consumidores han ido aumentando a lo largo del tiempo, generando así la proliferación de las aplicaciones que brindan una experiencia única para los usuarios. Entre los desafíos que hoy están presentes en las organizaciones surgen tener un conocimiento claro de quienes son sus clientes, análisis constante del mercado, flexibilidad al cambio y constante actualización digital y de contenido.

Esta problemática generó la creación de nuevas soluciones tecnológicas y arquitectónicas llamadas WCMS (Web Content Management System). Existen diferentes definiciones para estos sistemas, pero básicamente consisten en procesos de control y creación de contenido para ser consumido a partir de diferentes canales (típicamente web, móvil y social), desde páginas web hasta aplicaciones móviles. Estos productos han evolucionado en los últimos años y se han transformado en algo más que un administrador de contenido; se han convertido en la columna vertebral para brindar una experiencia digital a todos los clientes. Estas plataformas manejan enormes volúmenes de contenido para miles de páginas web y aplicaciones móviles que utilizan las organizaciones hoy en día, siendo una valiosa fuente para el análisis de información.

A partir del conocimiento de esta problemática nace Conexio –nuestro cliente-, una empresa dedicada a brindar servicios de *software* utilizando la plataforma WCMS llamada Adobe Experience Manager. Además Conexio presenta un gran interés en ampliar la oferta de servicios a otra plataforma WCMS llamada Sitecore. Para lograr tener la certeza de que dichas plataformas son líderes en el mercado, vamos a recurrir a consultoras internacionales –Gartner y Forrester- que evalúan el mercado de los WCMS.

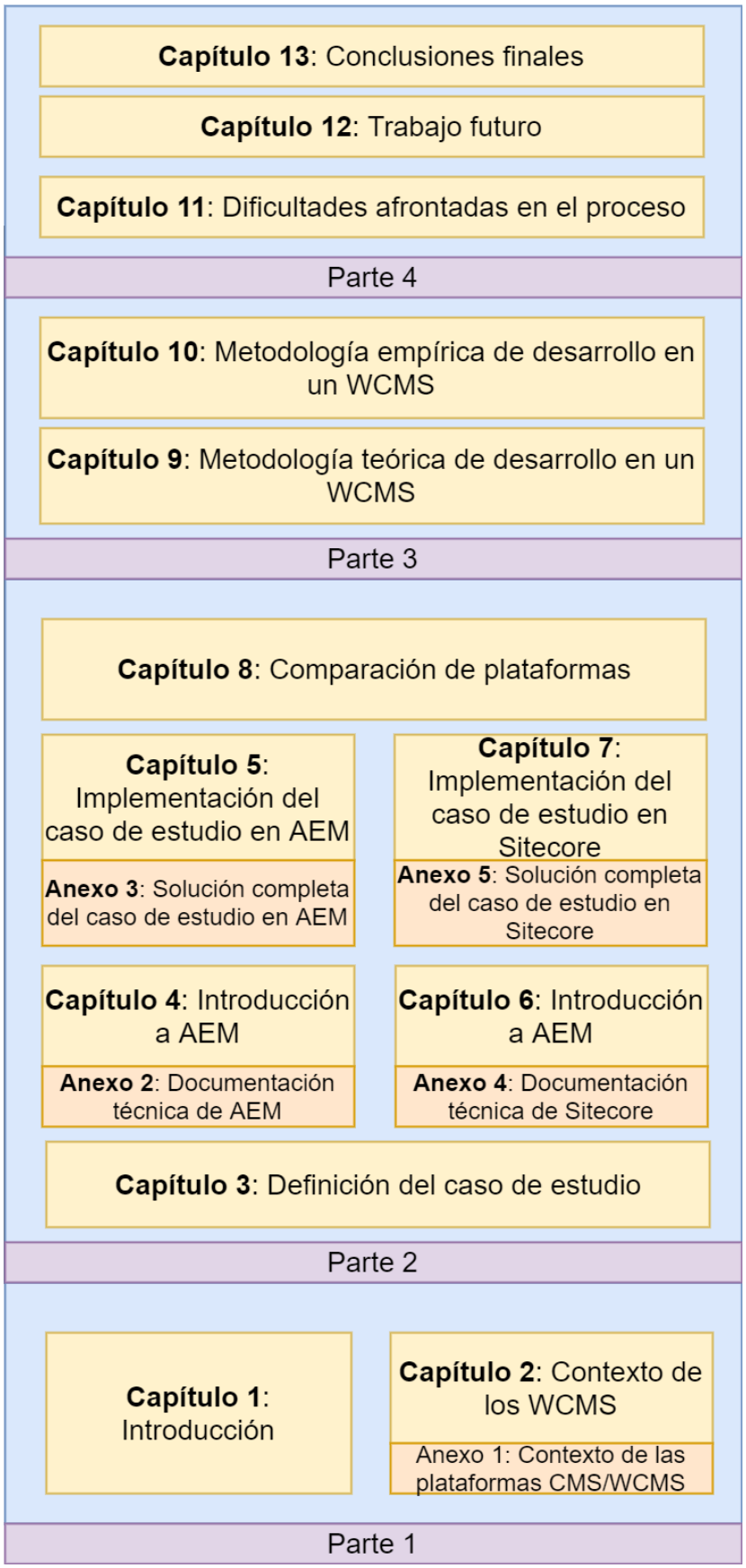
Junto con Conexio, definiremos un caso de estudio el cual va a ser implementado en ambas plataformas. El mismo incluye tanto un desarrollo de un sitio web como el de una aplicación móvil. Con la experiencia adquirida en la implementación del caso de estudio en ambas plataformas –AEM y Sitecore- realizaremos una comparación, resaltando las fortalezas y debilidades de cada una. Desde el punto de vista tecnológico, el equipo de trabajo deberá afrontar desafíos no solo en tecnologías de *front end* sino también en *back end*, pasando también por tecnologías móviles.

Por último investigaremos las metodologías recomendadas para los proyectos WCMS. Adicionalmente especificaremos una metodología que puede ser llevada a la práctica, definida en conjunto con Conexio, la interacción de Conexio y sus clientes y la experiencia adquirida en el transcurso del presente proyecto.

Palabras clave: contenido, componente, autor, instancia, renderizado.

Organización general del documento

El presente documento está organizado en cuatro partes las cuales se encuentran divididas en capítulos.
La primera consta básicamente de una introducción, definiendo el problema a resolver, planteando la motivación del trabajo, los objetivos propuestos, junto a los resultados esperados y un marco conceptual de las plataformas CMS/WCMS.
La segunda parte presenta el caso de estudio y la solución del mismo en las plataformas líderes –Sitecore y AEM-. Además se realiza una introducción técnica previa a cada solución de las plataformas utilizadas, con el objetivo de que el lector comprenda las soluciones propuestas.
La tercera parte abarca la metodología de desarrollo utilizada por los proyectos basados en herramientas WCMS. Comenzaremos con un marco teórico para luego especificar una metodología aplicada en la práctica, la cual surge a partir de la combinación de entrevistas con nuestro cliente.
Finalmente, la cuarta parte contiene las conclusiones, lecciones aprendidas, y trabajo futuro.
A continuación se puede observar una ilustración de la organización del informe. Es importante aclarar que hay dos anexos que no se corresponden con ninguna de las cuatro partes de este informe, los mismos son el *Anexo 6 Actas de reuniones* y el *Anexo 7 Planificación y estimación del proyecto*.


Capítulo 13: Conclusiones finales
Capítulo 12: Trabajo futuro
Capítulo 11: Dificultades afrontadas en el proceso
Parte 4
Capítulo 10: Metodología empírica de desarrollo en un WCMS
Capítulo 9: Metodología teórica de desarrollo en un WCMS
Parte 3
Capítulo 8: Comparación de plataformas
Capítulo 5: Implementación del caso de estudio en AEM
Anexo 3: Solución completa del caso de estudio en AEM
Capítulo 7: Implementación del caso de estudio en Sitecore
Anexo 5: Solución completa del caso de estudio en Sitecore
Capítulo 4: Introducción a AEM
Anexo 2: Documentación técnica de AEM
Capítulo 6: Introducción a AEM
Anexo 4: Documentación técnica de Sitecore
Capítulo 3: Definición del caso de estudio
Parte 2
Capítulo 1: Introducción
Capítulo 2: Contexto de los WCMS
Anexo 1: Contexto de las plataformas CMS/WCMS
Parte 1

Índice

Parte 1

En esta parte se expondrán los puntos generales que motivan y han dado forma al proyecto documentado en este informe. También se brindará un marco contextual que ubicará al lector en la temática involucrada en el proyecto –CMS/WCMS-, para introducirlo en la misma y facilitar la comprensión del resto del informe.

Capítulo 1: Introducción

A continuación se plantearán los puntos más importantes para comprender a grandes rasgos en que consiste el proyecto realizado y documentado en este informe. Para dicho fin, se expondrán la definición, motivación, objetivos y resultados esperados.

1.1 Definición

El presente proyecto está dedicado a la investigación y el estudio de plataformas CMS –Sistemas de administración de contenido- con especial énfasis en WCMS –Sistemas de Administración de Contenido Web-. A su vez se definirá un caso de estudio el cual se implementará por dos plataformas líderes en el mercado, con el fin de adquirir conocimiento en lo que atañe al desarrollo y generar documentación técnica al respecto. Por último se planteará una posible metodología a ser utilizada para este tipo de proyectos.

1.2 Motivación

En primer lugar, nuestro cliente, Conexio (www.conexiogroup.com) [1], una empresa que brinda servicios de desarrollo -a clientes de Estados Unidos- con metodologías ágiles utilizando plataformas WCMS, en particular Adobe Experience Manager –AEM- [2], presenta un marcado interés en el desarrollo del proyecto. Esto se debe a que trabaja con herramientas WCMS y desea profundizar su conocimiento en el rubro –especialmente Sitecore-, y así disponer de nuevas plataformas para brindar sus servicios.

En segundo lugar, nos dedicamos al estudio de estas herramientas debido a su crecimiento en el mercado. En la actualidad, la mayoría de los sitios web cuentan con un sistema de administración de contenido, cuyos volúmenes crecientes de información exigen mayor almacenamiento y capacidad de administración. En particular, los sitios web de grandes empresas a nivel mundial, y en respuesta a la rapidez con la que cambia el mercado y la sociedad de hoy en día, deben dar respuestas rápidas y eficientes a dichas problemáticas. Una plataforma WCMS resulta necesaria para poder afrontar los desafíos mencionados.

1.3 Objetivos del proyecto

- Comprender qué ofrecen las plataformas WCMS. Además, se realizará una comparación de las diferentes herramientas que existen en la actualidad.
- Comprobar la hipótesis sobre que AEM y Sitecore son las plataformas líderes del mercado.
- Definir e implementar un caso de estudio que será utilizado para comparar las plataformas líderes del mercado. El caso de estudio debe abarcar la mayor cantidad de módulos que estas plataformas ofrecen a la hora de administrar contenido.
- Realizar un análisis comparando las plataformas utilizadas en el caso de estudio.
- Definir y documentar una metodología de desarrollo para plataformas WCMS, fruto de la experiencia de Conexio con sus clientes y de la información teórica obtenida en el marco de este proyecto, esto se debe al interés de Conexio de sistematizar su proceso de desarrollo.
- Generar documentación suficientemente profunda para brindarle a Conexio la posibilidad de expandir su oferta de desarrollo utilizando este tipo de herramientas.
- Generar documentación la cual pueda ser utilizada en el proceso de inducción de nuevos empleados.

1.4 Resultados esperados

- Adquirir conocimiento sobre las plataformas WCMS, así como comprender las partes que las componen y su funcionamiento, a través del desarrollo de un caso de estudio.
- Generar documentación técnica sobre las plataformas elegidas para la implementación del caso de estudio.
- Llevar a cabo una implementación de un sitio web y una aplicación móvil utilizando dos plataformas WCMS –AEM y Sitecore [3]-, además de generar la documentación que justifique y explique el desarrollo de las partes involucradas en el proceso.
- Generar documentación sobre una posible metodología aplicable a este tipo de proyectos.
- Generar oportunidades comerciales a Conexio con los conocimientos adquiridos junto con la documentación técnica de la implementación del caso de estudio.

1.5 Resumen de capítulo

Se aborda este proyecto motivado por el interés de Conexio en la investigación de plataformas WCMS –especialmente Sitecore- y por el crecimiento en el mercado que ha presentado el uso de estas plataformas.

Como objetivos del proyecto, entre otros, se presenta el estudio y profundización de plataformas WCMS –en particular AEM y Sitecore- mediante la implementación de un caso de estudio en ambas plataformas. También se espera generar documentación sobre las mismas, así como documentación teórica sobre una posible metodología de desarrollo en plataformas WCMS –que pueda ser un producto de valor para Conexio de cara a la implementación de la misma en nuevos proyectos comerciales-.

Capítulo 2: Contexto de los WCMS

Se presentarán los conceptos básicos concernientes a los WCMS y al contenido, que ayudarán a tener una idea general sobre la temática involucrada en este proyecto, y que servirá para la comprensión del resto del informe.
El presente capítulo es un resumen del *Anexo 1 Contexto de las plataformas CMS/WCMS.*

2.1 Datos y contenido

Las computadoras fueron construidas para procesar datos. Los datos consisten en pequeños fragmentos de información con parte de su significado humano extraído y muchas veces carecen del mismo. Como los datos, el contenido también es información, pero es muy importante su significado humano y contexto.
En términos resumidos el contenido es información producida por un proceso editorial y tiene como único cometido ser consumido por personas vía publicación.
Los WCMS tienen el gran desafío de manejar ese contenido.
En la actualidad el principal uso de las computadoras es como un procesador de datos. En contraparte, la mayoría de los usuarios buscan que las computadoras analicen y manejen enormes cantidades de información -no fracciones- y sean capaces de enviar la información que ellos seleccionen en el menor tiempo posible [4].

2.2 Marco histórico de la evolución del contenido

En los últimos 30 años se ha dado una revolución en materia de información. Lo que impulsó esta revolución fue la posibilidad de crear medios digitales como imágenes, sonidos y videos en las computadoras, así como también la posibilidad de acceder a ellos desde diferentes dispositivos. A lo que se adicionó la disposición para el consumo masivo de distintos artefactos de almacenamiento de alta capacidad y baratos [4].
A partir de estos avances se potenció el rápido desarrollo de la industria de la multimedia y más aún el uso de la multimedia en diferentes industrias tradicionales. Por primera vez se podía generar contenido y distribuirlo de forma masiva y barata con los CD-ROMs. Pronto la industria se despegó y los CD-ROMs proliferaron, repartiendo contenido que variaba desde grandes enciclopedias, catálogo o juegos. Las computadoras comenzaron a emerger como un reemplazo a los canales tradicionales para hacer llegar información a los usuarios como libros, televisión y radio. Estos canales transmitían contenido y no datos. La revolución había comenzado [4].
La web terminó de consolidar lo que las industrias de contenido multimedia habían comenzado. En la actualidad, buscar el contenido de forma *online* no solo es posible, sino que es lo que la mayoría de los usuarios eligen por encima de la utilización de otros canales. Aunque las necesidades y expectativas de los usuarios cambiaron, la esencia de las computadoras no. Hace 30 años las personas llegaban a las computadoras con datos de entrada, procesos y esperaban datos de salida. Hoy los usuarios utilizan las computadoras para **consumir contenido**. A pesar de esto la base tecnológica de la computación sigue siendo la misma. Se basa en la idea de que se puede reducir cualquier problema a un conjunto de simples instrucciones que trabajen con fragmentos de datos estructurados y discretos [4].
El contenido y los datos son diferentes, pero eso no significa que no exista interacción entre ellos. Innumerables transiciones de uno a otro ocurren todos los días. Sin embargo desde el punto de vista de los sistemas computacionales el contenido no existe, solo existen los datos [4].

2.3 Clasificación de tipos de gestión de contenido

Los cuatro grandes tipos de gestores de contenido pueden ser separados de la siguiente forma [5]:

2.3.1 WCM (WEB Content Management)

Brinda la gestión de contenido, para en primera instancia ser distribuido masivamente a través de un sitio web. Se destaca en separar el contenido entre la presentación y la publicación en múltiples canales. Es el tipo de gestor de contenido más utilizado hoy en el mercado. Por citar algunos ejemplos: WordPress, Drupal [6], AEM, Sitecore.

2.3.2 ECM (Enterprise Content Management)

El término contenido empresarial es a menudo usado para hacer referencia a contenido interno que usualmente no es publicado fuera de la organización.
Consiste en la gestión de contenido de negocio general, no necesariamente dirigido a un público masivo o para consumo interno de una organización o empresa, como por ejemplo *currículums vitae* de empleados, reportes de incidentes, *memorándums*, etc. Este tipo de CMS ha sido tradicionalmente mejor conocido como un administrador de documentos. Algunos ejemplos de ECMs son: IBM [7], OpenText [8], EMC [9].

2.3.3 DAM (Digital Asset Management)

Realiza la administración y manipulación de archivos digitales como imágenes, audio, y video para que estos sean utilizados en otros medios.
Mientras casi cualquier sistema gestor de contenido puede almacenar archivos de imagen y video, los sistemas DAM va un paso más allá proveyendo herramientas para crear y transformar archivos digitales. Las imágenes pueden ser redimensionadas y los videos pueden ser ajustados y editados directamente en el sistema. Por lo tanto, las características principales de un sistema DAM son estrechamente similares a las de un sistema ECM. De hecho, muchos sistemas DAM son vendidos simplemente como complementos a sistemas ECM. Asimismo muchas plataformas WCMS tienen incorporado un sistema DAM.

2.3.4 Gestión de registros (Records Management)

Gestiona la información transaccional y otros registros que se crean como subproducto de operaciones de negocio, por ejemplo, registros de ventas, registros de acceso, contratos, etc.). Sobresale en retención y control de acceso.

En conclusión, la línea entre los tipos de gestores de contenido citados es difusa. Un sistema DAM es a menudo usado para proveer contenido a un sitio web a través de una integración con un sistema WCM. Además, algunos sistemas ECM tienen subsistemas por los cuales pueden publicar algo de su información en la web.
Por lo tanto, las definiciones brindadas no son estrictas y muchos sistemas son reconocidos sólo a través de su intención de uso y de su percepción en la industria.

2.4 Definición de WCMS

Un WCMS es una aplicación que permite a los usuarios obtener control sobre la creación y distribución de contenido y funcionalidades [4]. Adicionalmente provee cierto nivel de automatización para las tareas requeridas para la administración de contenido [5].
Un WCMS es usualmente un *software* basado en el servidor web y multiusuario, que interactúa con el contenido almacenándolo en un repositorio. El repositorio puede estar localizado en el mismo servidor –como parte del mismo *software*-, o estar ubicado en forma completa en una instalación de almacenamiento externa.

Los WCMS son plataformas extensas y complejas con múltiples actores involucrados. Están compuestos por varias partes, como los son la interfaz de edición, el repositorio, los mecanismos de publicación, entre otros. Para quien use este tipo de sistemas y no posea conocimientos técnicos, las partes mencionadas previamente son vistas como un todo, de forma monolítica: "el WCMS". Sin embargo, podrían estar separadas e inclusive son partes autónomas de por sí en el sistema.

A pesar que la utilización primaria de los WCMS es la creación de sitios web de gran porte, el potencial de los WCMS para ayudar a mejorar a las organizaciones va más allá de la web. En el repositorio se persisten y organizan enormes volúmenes de información. Partiendo de ese repositorio, las organizaciones serían capaces de producir publicaciones de dicho contenido a un sitio web o a cualquier otro canal posible [4].

En conclusión, un WCMS permite la creación, edición, ejecución de procesos editoriales de contenido, y en última instancia, hacer que el mismo esté disponible para el consumo de los usuarios.

A continuación se puede visualizar -Figura 1- un entorno web para administrar contenido que presenta un WCMS o ambiente de autor –se verá más adelante-, en este caso corresponde a WordPress [10] –uno de los WCMS más utilizados en la actualidad-.

La gran mayoría de los WCMS presentan un formato similar de cómo está presentada la aplicación. En la parte central de la pantalla se provee de un panel administrador, el cual puede contener acceso a distintas funcionalidades y/o mostrar distintas estadísticas o métricas -dependiendo del WCMS-. Sobre la izquierda y barra superior se brindan menús con todas las herramientas que provee el WCMS para la administración del contenido -en todas sus etapas-.

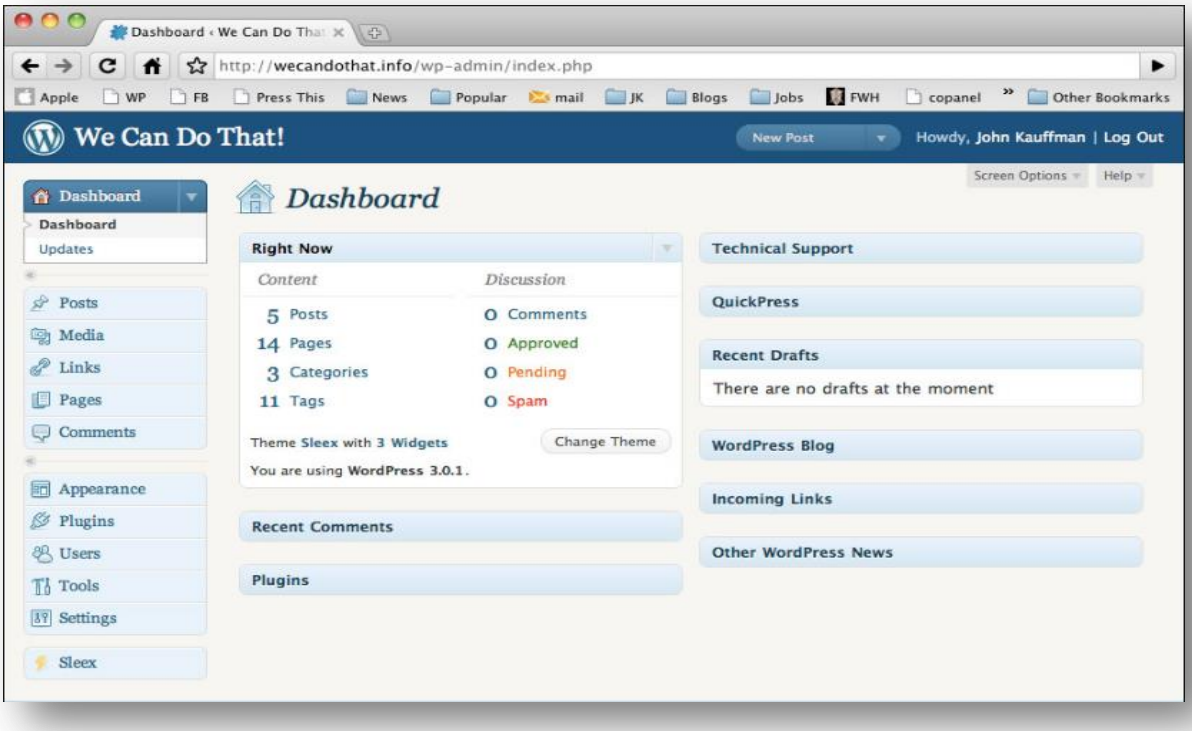


Figura 1 - Toma de pantalla de WCMS WordPress. Fuente: [11].

2.5 Stack tecnológico de un WCMS

Como toda aplicación, un WCMS es ejecutado sobre un conjunto de *software* de soporte, bases de datos y lenguajes de programación. Los mismos están implementados en un lenguaje de programación y utilizan un *framework* de almacenamiento específico [5].
El lenguaje y *framework* utilizados influencian sensiblemente el ambiente de alojamiento necesario para que el WCMS pueda ejecutar.
El stack tecnológico está compuesto por:

- El WCMS propiamente dicho.
- Un *framework* de programación.
- Un lenguaje de programación.
- Un servidor de base de datos.
- Un servidor web.
- Un sistema operativo.

2.6 Ciclo de vida del contenido

Uno de los términos más importantes –sino el más importante- de este informe es el del contenido. El mismo tiene un ciclo de vida -en un WCMS- compuesto por cinco etapas -Figura 2-, que van desde su creación a su eliminación. Se muestra a continuación una figura representando el citado ciclo de vida y la descripción de las etapas que componen el mismo [5].

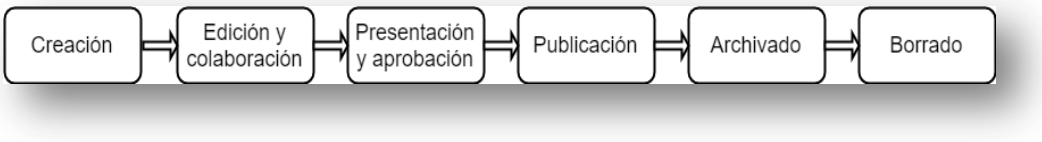


Figura 2 – Ciclo de vida del contenido en un WCMS. Fuente: Autoría Propia, en base a [5].

- **Creación**
  El contenido es iniciado en el WCMS. No está completo, pero existe como un objeto de contenido gestionable.

- **Edición y colaboración**
  El contenido es activamente editado, tanto por un solo editor –autor- como por parte de un equipo. El contenido todavía no es visible en esta etapa.

- **Presentación y aprobación**
  El contenido creado y editado es presentado a una o más aprobaciones. Continúa sin ser visible por los usuarios.

- **Publicación**
  Se publica en el sitio web. En este estado el contenido es visible para el consumo de los usuarios.

- **Archivado**
  El contenido es removido del acceso al público pero no es borrado. Usualmente no va a ser visible para los usuarios de aquí en adelante.

- **Borrado**
  El contenido es borrado de forma definitiva del WCMS

Es importante mencionar que las etapas descritas son iterativas y pueden ser aplicadas en simultáneo a distintas versiones del mismo contenido.

2.7 Tareas realizadas por un WCMS

Un WCMS permite controlar y registrar el historial del contenido.
Específicamente, un WCMS provee funciones de control básicas, que ayudan a mejoran el nivel de control sobre el contenido. Algunas de las funciones provistas son [5]:

2.7.1 Control de contenido

2.7.1.1 Permisos

Un WCMS debe administrar los permisos de los diferentes usuarios que acceden al contenido, teniendo en cuenta quién puede ver, cambiar, o eliminar el mismo.

2.7.1.2 Manejo de estado y workflow

Permite tener control sobre el estado del contenido y sus posibles estados, como lo es saber si el contenido ha sido publicado, si está como borrador o si ha sido archivado o removido.

2.7.1.3 Versionado

Se mantiene un control sobre las versiones que se van generando del contenido, con lo cual se puede rápidamente verificar si el contenido ha sido cambiado, cómo era el mismo tiempo atrás, qué diferencia presenta la versión actual con una anterior o si es posible restaurar o republicar una versión anterior.

2.7.1.4 Manejo de dependencias

Se puede verificar, por ejemplo, qué contenido está siendo usado por otro contenido, como afecta al resto del contenido si se borra cierto contenido, o saber qué contenido está “huérfano” o en desuso.

2.7.1.5 Búsqueda y organización

Se puede realizar una búsqueda de una parte específica de contenido, encontrar todo el contenido que refiere a un término o agrupar y relacionar contenido para que sea más sencillo de manejar.

2.7.2 Permitir el reuso del contenido

Usar contenido en más de un lugar y en más de una forma aumenta su valor.
La habilidad de reusar contenido es altamente dependiente de la estructura del mismo. La habilidad para estructurar el contenido adecuadamente para un reuso óptimo depende en gran medida en las prestaciones del WCMS usado.

2.7.3 Permitir automatización del contenido y agregación

Teniendo todo el contenido en una única ubicación hace más fácil su consulta y manipulación.
Si el contenido es estructurado correctamente, se puede manipular para ser mostrado en diferentes formatos, publicarlo en diferentes lugares, y reorganizarlo sobre la marcha para satisfacer las necesidades de los visitantes más eficazmente:

- Se puede permitir a usuarios que consuman el contenido en otros formatos, como PDF o formatos de libro electrónico.
- Se pueden crear listas automáticamente y navegación para el sitio web.
- Se pueden crear múltiples traducciones de contenido para asegurar que se entregue en el lenguaje más apropiado al usuario actual.
- Se puede alterar el contenido que se publica en tiempo real basado en los comportamientos específicos exhibidos por los usuarios.

Un WCMS permite estas cosas estructurando, almacenando, examinando, y proveyendo facilidades de consulta sobre el contenido.

2.7.4 Mejorar la eficiencia editorial

La capacidad de los editores –mejor conocidos como autores en el ambiente WCMS- para crear y editar el contenido de forma rápida y precisa está enormemente afectada por la plataforma utilizada.
La eficiencia del autor se incrementa en un sistema que controla qué tipo de contenidos pueden y no pueden añadir, que herramientas de formato están disponibles para ellos, cómo su contenido es estructurado en el ambiente de autor, cómo es el *workflow* editorial y cómo se gestiona la colaboración. Adicionalmente a lo que le sucede al contenido después de que se publique.
Un buen WCMS permite a los editores publicar más contenido en un marco de tiempo más corto -que aumenta el "rendimiento editorial"-, y para controlar y gestionar el contenido publicado con una menor cantidad de fricción o arrastre sobre su proceso.

2.8 Tareas no realizadas por un WCMS

A continuación se describen actividades que un WCMS no realiza y que en caso de ser realizadas podría generar problemas o expectativas no satisfechas [5]:

2.8.1 Crear contenido

Un WCMS simplemente gestiona contenido, no lo crea. Todavía un ser humano debe proveer el poder editorial para generar el contenido que se supone va a ser gestionado.
Un WCMS no asegura la calidad del contenido. Aunque puede ofrecer muchas herramientas para minimizar la pobre calidad del mismo desde un punto de vista técnico. Un WCMS por sí solo no puede editar el contenido para asegurarse que tenga sentido y satisfaga las necesidades del público.

2.8.2 Crear planes de marketing

Aun asumiendo que el contenido está creado de forma consistente y bien administrado, eso no significa que provea a la organización algún valor - que es contrario al objetivo de un WCMS-.
Un WCMS desconoce sobre marketing. Mientras algunas plataformas tienen herramientas de marketing incorporadas, las mismas aun dependen de los humanos para su dirección. El marketing efectivo es una práctica humana que involucra una combinación de estética, sociología, psicología, experiencia e intuición. Un WCMS puede hacer que se ejecuten los planes de marketing de manera más sencilla y más eficiente, pero esos planes todavía necesitan ser concebidos, creados, y analizados por un humano competente.
Un WCMS no toma el lugar de un equipo creativo que entiende el mercado, los clientes, los competidores, y lo que es necesario para diferenciarse. No hay *software* que pueda tomar el lugar de una buena estrategia o buen equipo de marketing digital.

2.8.3 *Efectivo formato de contenido*

Mientras un WCMS puede estructurar contenido y darle formato automáticamente durante la publicación, todavía hay un amplio margen para que el autor cometa errores. La mayoría de los WCMS tienen un editor de texto o alguna otra interfaz que permite a los autores dar formato a texto e imágenes. Esto puede llevar a errores, como por ejemplo:

- Uso excesivo de letra tipo negrita e itálica.
- Alineamiento inconsistente del contenido.
- Relaciones inconsistentes.
- Pobre emplazamiento de imágenes.

2.8.4 *Proveer definiciones de gobernanza*

La gobernanza describe el acceso a procesos alrededor del contenido:

- Determinar quién tiene acceso a qué parte.
- Se encarga de la definición de procesos o pasos a seguir ante un cierto evento.

Cada WCMS tiene un conjunto de métodos para limitar las acciones que un usuario puede tomar, pero estos límites tienen que ser definidos por adelantado. El WCMS simplemente llevará a cabo lo que la organización dicte que se haga. Estos planes tienen que ser creados a través de la interacción y juicio humano, entonces convertido en permisos y límites de acceso que el WCMS puede hacer cumplir.
La gobernanza es ante todo una disciplina humana. Se determinan los procesos y políticas que los seres humanos acatarán cuando se trabaja con el contenido. El WCMS es sólo un marco de aplicación.

2.9 Descomposición de los principales sistemas de un WCMS

Las plataformas WCMS se pueden descomponer en tres sistemas principales que nuclean su funcionamiento, los cuales son [4]:

- El sistema recolector.
- El sistema de gestión.
- El sistema de publicación.

2.9.1 *Sistema Recolector*

El sistema recolector es responsable por todos los procesos que suceden antes de que el contenido esté listo para su publicación. Es el encargado de la transformación de la información cruda en un conjunto organizado y estructurado de componentes de contenido. El proceso incluye:

2.9.1.1 Autoría

La autoría se refiere al proceso de creación de contenido desde cero. Dentro de este proceso se define al autor como alguien que específicamente está encargado de crear contenido para el WCMS.
El WCMS debe ayudar al autor a trabajar de forma eficiente y efectiva brindándole las siguientes prestaciones:

- **Ambiente de autor**, el cual brinda un entorno para la creación y administración de contenido (ya sea una aplicación completa, un ambiente web o una extensión al ambiente nativo del autor).
- Una audiencia y un propósito claro para los esfuerzos del autor.

- Funcionalidades de asistencia para incluir información estándar. Por ejemplo, los WCMS fácilmente pueden incluir en el contenido la fecha de la creación y el nombre del autor para registrar los esfuerzos hechos en el mismo.
- *Workflows*, *status* y control de versión para el contenido que se encuentra en proceso.

Sin importar cuantas herramientas y procesos pueda brindar el WCMS, la autoría es esencialmente una tarea manual. Ningún WCMS le puede decir a un autor que escribir o que resaltar como importante o cuál es la mejor forma de comunicar una idea o un objetivo. Esto lleva a que la autoría sea lenta y costosa.

2.9.1.2 Adquisición

La adquisición es el proceso de recolectar información que originalmente no fue creada para el WCMS. El proceso puede ser parcialmente manual o completamente automático. Generalmente el mayor esfuerzo en este proceso se encuentra en transformar o adaptar dicha información a los estándares del WCMS.
Mientras la información generada a partir de los autores tiene poco volumen pero una gran calidad, la información adquirida se encuentra en gran volumen y con una calidad baja.

2.9.1.3 Conversión

La información que es creada o adquirida generalmente no se encuentra en el formato o estructura que el sistema de contenido requiere, esta información debe ser adaptada para que cumpla con los estándares definidos por el sistema.

2.9.2 Sistema de Gestión

El sistema de Gestión del WCMS es responsable de la persistencia de los componentes y otro tipo de recursos. Debe permitir mantener la información de todos los recursos que se han recolectado y cuál es su ubicación. Entre otras cosas el sistema administrador debería ser capaz de indicar:

- El contenido persistido en detalle, incluyendo qué tipo de componentes contiene y en qué etapa del ciclo de vida se encuentra cada uno.
- Cómo se están usando los componentes de contenido en las publicaciones y cuales no están siendo utilizados.
- Quien ha tenido acceso al contenido y lo ha modificado.

Para poder proveer esta funcionalidad, el sistema de gestión incluye:

- **Repositorio**
  Un lugar donde guardar el contenido.

- **Administración**
  Un sistema administrador para configurar el WCMS.

- **Workflow**
  Define entre otros procesos, los pasos necesarios que deben realizarse en el contenido para que sea publicado.

- **Conexiones**

  Un conjunto de conexiones (hardware y software) a otros sistemas.

2.9.2.1 Repositorio

El repositorio es la pieza más importante del sistema de gestión. Básicamente es un conjunto de bases de datos, directorios u otro tipo de estructuras que guardan el contenido y cualquier otro dato asociado con el WCMS. Los componentes de contenido y otros recursos del WCMS son persistidos en el repositorio mediante los servicios de recolección –como ilustra la Figura 3-, y los servicios de publicación los extraen. El repositorio puede contener los siguientes tipos de archivos:

- Bases de datos y archivos de contenido.
- Archivos de configuración y control.

2.9.2.1.1 Bases de datos y archivos de contenido

Los archivos y bases de datos de contenido almacenan el contenido de los componentes del sistema. Las bases de datos de contenido pueden consistir en una base estándar relacional, una base de datos de objetos XML o un híbrido de ambas.

Las bases de datos relacionales utilizan tablas, columnas y filas para representar los componentes, en cambio en las bases de datos orientadas a objetos XML los componentes quedan completamente representados como un XML y son persistidos en una enorme jerarquía. En este caso cada tipo de contenido, componentes y elementos que contienen son representados mediante una etiqueta.

2.9.2.1.2 Archivos de configuración y control

Los archivos de configuración y control no son archivos de contenido, pero de todas formas serán manejados por el sistema administrador y persistidos en el repositorio.

Los archivos de control y configuración incluyen entre otros:

- Archivos de configuración de acceso al personal de la organización y archivos y bases de datos del usuario final.
- Archivos de reglas y bases de datos.
- *Logs* y archivos de control y estructura.
- *Scripts* y rutinas de mantenimiento automático.

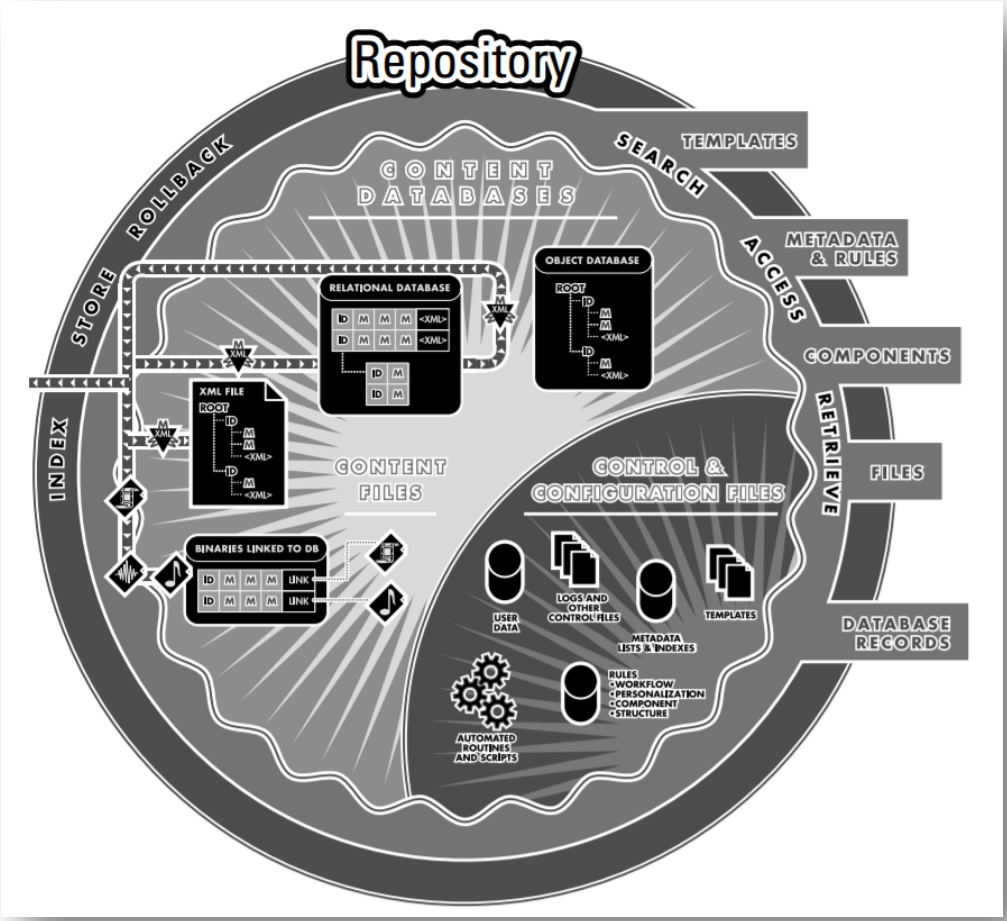


Figura 3 – Esquema del repositorio para un WCMS. Fuente: [4].

2.9.3 Sistema de publicación

El sistema de publicación es responsable por extraer los componentes de contenido y otros recursos del repositorio creando publicaciones automáticas con dichos elementos.

2.9.3.1 Publicaciones web

En la actualidad el uso más frecuente de los WCMS es el de crear y administrar sitios web de gran porte.

Si las publicaciones web son dinámicas, básicamente cada vez que una página web del sitio es cargada en el navegador, los servicios de publicación web realizan las siguientes tareas:

1. Cargan las plantillas asociadas.
2. Transfieren los parámetros que hayan llegado con el pedido que creó la interacción del usuario con la web.
3. Ejecutan el código que contiene las plantillas para producir la página web que va a ver el usuario.
4. Envían la página web creada a través del servidor web para que sea mostrado en el navegador del usuario.

En la actualidad la mayoría de los WCMS son dinámicos. Las páginas web son creadas en el momento de ejecución a partir de las plantillas disponibles. Generalmente los WCMS crean distintas estrategias de caché para evitar generar nuevamente una página si la plantilla no sufrió ningún cambio.

2.10 Prestaciones de un WCMS

Las plataformas WCMS son productos potentes -que van más allá de la simple publicación de contenido para un sitio web- que cuentan con varias prestaciones que fundamentan dicha afirmación y que les brindan un valor agregado. Algunas de las prestaciones que brindan son las siguientes:

- **Escalabilidad**
  En particular al contar con un WCMS desacoplado –gestión y consumo de contenido en distintos servidores-, la publicación de contenido se vuelve escalable, estable y segura [5].
- **Integración de módulos**
  Permiten la integración con módulos externos, por ejemplo, con los relacionados a la faceta comercial y de marketing –en particular *Analytics*-.
- **Experiencia de usuario**
  Cada combinación de un visitante más contenido es una nueva experiencia de usuario. Un WCMS es capaz de optimizar la interacción con el visitante a través de una variedad de herramientas, como por ejemplo, personalización, *testing*, *analytics* [5]. El sitio web podría cambiar en función del perfil del usuario, proveyendo contenido y funcionalidades que ese visitante puede requerir en ese momento [5].
- **Independencia de personal de IT**
  Es posible la utilización de un WCMS sin conocimiento técnico en el área de tecnologías de la información, por ejemplo para los autores, solo bastaría con la práctica suficiente para la utilización del WCMS.
- **Multicanal**
  Se pueden brindar muy buenas experiencias de usuario para una gran variedad de dispositivos, en particular para los dispositivos móviles. Puede ser logrado a través de diseño adaptativo, responsivo (web) o a través de diseño a través de APIs que permitan el desarrollo de aplicaciones móviles. Adicionalmente, atendiendo al avance y auge de los últimos años de las redes sociales, éstas se han convertido en un canal más que importante.
- **Soporte de varios idiomas**
  Permite la creación de traducciones para asegurar la distribución del mismo en el idioma adecuando según el usuario del sitio web.
- **Administración multisitio**
  Es posible administrar una cantidad arbitraria de sitios web a través de una única instancia.
- **Procesamiento de imágenes**
  Se brinda una biblioteca de imágenes, permitiendo subir las que se deseen para su posterior uso, además de poseer imágenes precargadas.

2.11 Presente del WCMS

Actualmente, la utilización de WCMS ha ido en expansión, principalmente por los grandes volúmenes de información manejados por los sitios web y por las ventajas que presenta la utilización de una plataforma de este tipo. Los usuarios de estas plataformas van desde usuarios particulares, hasta grandes empresas a nivel mundial que necesiten de una gestión eficiente y eficaz de su sitio web.

En la actualidad, de acuerdo al sitio https://builtwith.com [12], el cual realiza un análisis de todas las plataformas WCMS utilizadas para construir sitios web que componen Internet, se llega a la conclusión que los WCMS de código abierto lideran ampliamente en el mercado, en particular WordPress que es utilizado por el 51% de los sitios web que utilizan alguna plataforma WCMS. Los WCMS que secundan a WordPress también son de "código abierto", entre los cuales se encuentran Joomla [13] (7%), Blogger [14] (2%), y Drupal (2%). Se puede apreciar lo mencionado en la Figura 4 .

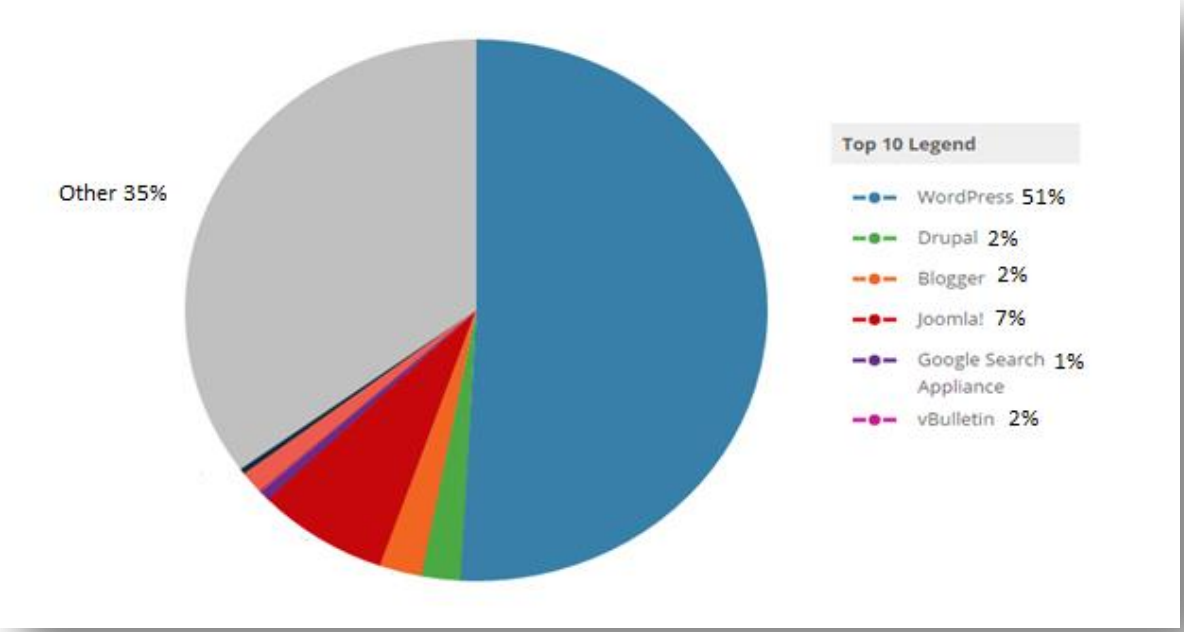


Figura 4 - Gráfico de uso de plataformas WCMS en sitios web. Fuente: www.builtwith.com [12].

En cuanto a los productos propietarios, las consultoras Gartner [15] y Forrester [16] analizan los principales productos WCMS del mercado de acuerdo a determinados criterios tomados por cada una de ellas. Es importante mencionar que no se puede brindar una comparativa de las plataformas de código abierto –al menos las más importantes dentro de ese segmento-, ya que los informes evalúan prácticamente en su totalidad WCMS comerciales. Se puede afirmar que no estén considerados por no cumplir los criterios de aceptación que tienen las consultoras. Tanto para Forrester como para Gartner, se van a tener en cuenta los dos últimos informes, que abarcan desde el año 2015 hasta el 2017 – como se verá a continuación-.

Para el caso de Forrester, presenta los resultados en su llamada Ola de Forrester. Esta grafica está compuesta por cuatro categorías, y cada producto WCMS evaluado es ubicado de acuerdo a la oferta actual que ofrecen y la fortaleza de su estrategia. También se visualiza en una escala de cinco posibles valores, la presencia en el mercado de cada plataforma. En primer lugar, en la Figura 5, se aprecia la ola de Forrester correspondiente a su informe del año 2015 [17]:

Figura 5 - Ola de Forrester. Fuente: Informe Forrester febrero 2015 [16].

Se desprende de la figura anterior que para Forrester, Adobe AEM y Sitecore son líderes del mercado en cuanto al producto WCMS ofrecido, además de poseer una buena presencia en el mercado. Forrester saca la siguiente conclusión en su informe del año 2015:
"*Adobe y Sitecore son líderes, con Adobe dominando el campo. Adobe sobresale por su portafolio integrado. Sitecore tiene un producto fuerte y con gran crecimiento*".
En su informe más reciente sobre WCMS que data de enero de 2017 [18], la ola de Forrester resultó ser la que se muestra en la Figura 6.
Se desprende que AEM sigue como líder de mercado, además de contar con la mayor presencia de mercado, según la escala proporcionada.
Sin embargo para el caso de Sitecore, la situación ha tenido un pequeño retroceso en comparación con el informe anterior. Sigue manteniendo una muy buena presencia en el mercado, pero ha pasado a perder la categoría de líder que ostentaba previamente, además de ser superado por algunos competidores -que pasaron a ser líderes-. Sin embargo, la posición que ostenta es muy buena de todas formas, siendo todavía de las mejores opciones en el mercado.

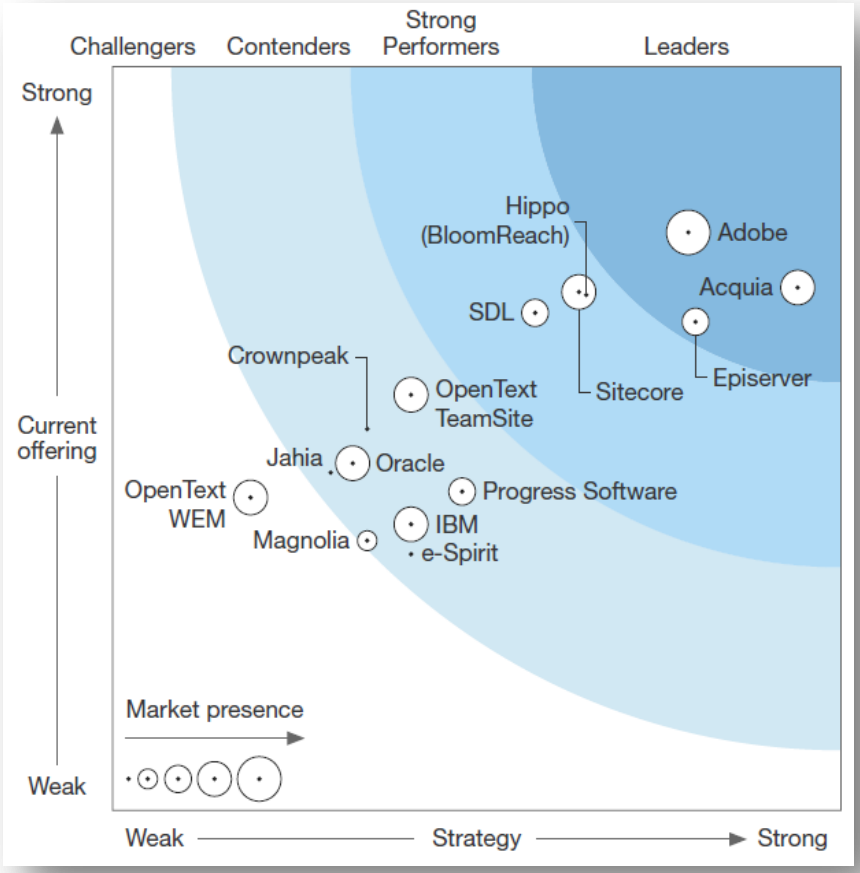


Figura 6 - Ola de Forrester. Fuente: Informe Forrester enero 2017 [18].

Por otra parte Gartner también ha publicado informes sobre productos WCMS, presentando su cuadrante mágico, el cual posee cuatro categorías en donde los productos evaluados son ubicados según la habilidad de ejecución y completitud. El último informe publicado fue en setiembre de 2016 [20] -Figura 8-, adicionalmente se cuenta con el informe correspondiente a julio del 2015 [19] -Figura 7-.

En ambas figuras -Figura 7 y Figura 8 - se observa que, pese a pequeñas variaciones, Sitecore y Adobe AEM se mantienen como líderes según Gartner.

Las cuatro evaluaciones muestran con bastante certeza que en la actualidad Adobe AEM y Sitecore son los productos WCMS líderes en el mercado –en particular AEM está avalado en los cuatro informes estudiados-, validando las hipótesis planteadas sobre las plataformas a usar en este proyecto –como fue mencionado al comienzo de este informe.

Figura 7 - Cuadrante mágico de Gartner. Fuente: Informe Gartner, julio 2015 [19].

Figura 8 - Cuadrante mágico de Gartner. Fuente: Informe Gartner, setiembre 2016 [20].

2.12 Resumen de capítulo

El contenido –a diferencia del término dato-, es información producida bajo un proceso editorial para el consumo de usuarios. Es muy importante su significado humano y contexto.
El contenido posee un ciclo de vida en el marco de los WCMS, y que la gestión del contenido está enmarcada en las etapas que lo componen.
La gestión de contenido se divide en cuatro categorías, pero como ya es mencionado previamente, este informe se enfoca en los WCMSs.
Un WCMS es una aplicación que permite a los usuarios obtener control sobre la creación y distribución de contenido y funcionalidades. Para su uso es necesario poseer un determinado *stack* de tecnologías. A su vez los WCMS cumplen ciertas tareas y otras no –como erróneamente se puede pensar en un primer momento-, que ayudan a proveer cierto nivel de automatización para las tareas requeridas para la administración de contenido.
Los WCMS son plataformas extensas y complejas con múltiples actores involucrados. Están compuestos por varias partes, que forman sistemas –recolector, de gestión y de publicación- que son encargados de cumplir funciones específicas en un WCMS.
A nivel de mercado del WCMS, predomina el uso de plataformas *open source*, liderando ampliamente Wordpress. Entre las plataformas WCMSs comerciales, Adobe AEM lidera de forma sostenida en los últimos años. Sitecore sigue de cerca a AEM, siendo un producto muy competitivo y completo.

Parte 2

Con el principal objetivo de enfrentar los desafíos técnicos que genera el desarrollo con plataformas WCMS, creamos un caso de estudio, junto con Conexio, que implementaremos utilizando dos diferentes productos –AEM y Sitecore-. A partir del mismo buscamos obtener los conocimientos necesarios para desarrollar componentes de contenido –en ambas plataformas-, siendo Sitecore la plataforma que Conexio tiene principal interés, ya que no cuenta con ningún conocimiento sobre la misma.

Definimos el caso de estudio tratando de abarcar la mayor cantidad de temáticas posibles en lo que ataña al desarrollo de un típico proyecto de WCMS (dentro del alcance del proyecto de grado).

Entre las temáticas abordadas, definiremos e implementaremos los componentes de contenido a partir de la documentación de requerimientos -asumiendo que parte de ésta nos fue entregada-.

Los capítulos siguientes incluirán una introducción a cada una de las plataformas –AEM y Sitecore- para luego describir la solución implementada. En el último capítulo realizaremos una comparación de ambas plataformas.

Capítulo 3: Definición del caso de Estudio

En el presente capitulo será presentado el caso de estudio a ser implementado. El mismo será un sitio web basado en una web real de Adobe - http://summit.adobe.com/na/ (último acceso: 14 Agosto, 2016) [21]- que tiene como objetivo presentar un conjunto de eventos de negocio. Implementaremos parte de este sitio utilizando AEM y Sitecore.

3.1 Propósito

Summit es un nuevo formato de evento de negocios, innovador en su categoría a nivel internacional. A través del mismo se busca ampliar la plataforma de negocios entre los proveedores de servicios, los usuarios y sus clientes. Generalmente este tipo de mega-eventos ofrecen charlas y centros de entrenamiento donde la atención es personalizada o en grupos reducidos. Los *Summits* abarcan varios días donde los participantes pueden asistir a las charlas o presentaciones que más les interesen.

El propósito del sitio web es presentar un Summit que ofrece varios eventos, los cuales son dirigidos por un orador y abarcan una temática específica. Adobe creo este sitio web - http://summit.adobe.com/na/ (último acceso: 14 Agosto, 2016)- para que los usuarios puedan obtener información sobre los diferentes eventos y oradores que estarán presentes. Además los usuarios identificados son capaces de reservar lugares en los eventos que les resulten más interesantes, facilitando así la asistencia de los participantes del Summit.

3.2 Funcionalidad Global

El caso de estudio está compuesto por cuatro páginas distintas y el diseño *front end* está basado en el sitio http://summit.adobe.com/na/.

El sitio web permite a los autores administrar diferentes tipos de evento. Los usuarios web identificados dentro de la aplicación tendrán la posibilidad de inscribirse a dichos eventos, creando reservas, así como también dar de baja inscripciones, eliminando las mismas. A su vez, los usuarios web no identificados tendrán la posibilidad de crear una cuenta dentro del sitio.

Para cada página contaremos con una imagen en la cual nos basaremos para definir los componentes de contenido-son los encargados de desplegar y configurar pequeñas cantidades de contenido -. También utilizaremos los archivos de estilo y HTML obtenidos del sitio base. En principio definiremos la funcionalidad global que comparten todas las páginas, para luego especificar la funcionalidad específica de cada una de ellas.

Adicionalmente el caso de estudio incluye una aplicación móvil, desde la cual se les ofrece a los usuarios identificados –esto es mediante un log in-la posibilidad de consultar los distintos eventos del Summit y reservar o eliminar una reserva de un evento.

Las funcionalidades globales del sitio web son:

3.2.1 Log In

El usuario deberá ser capaz de iniciar sesión desde cualquier página del sitio web mediante un nombre de usuario y una contraseña.

3.2.2 Log Out

El usuario -una vez iniciado la sesión- debe poder acceder a la funcionalidad de finalizar la sesión.

3.2.3 Navegación

El usuario debe poder acceder mediante un navegador global a las páginas de *Agenda*, *Adobe Attend* y *Home Page*. En cambio, a la página de registro de usuario se accede a través de un *box* de *Log In*.

3.3 Páginas del Sitio web

Observamos en la Figura 9 las cuatro páginas del sitio web, donde disponemos del código parcialmente completo *front end* (HTML, CSS) de las mismas.

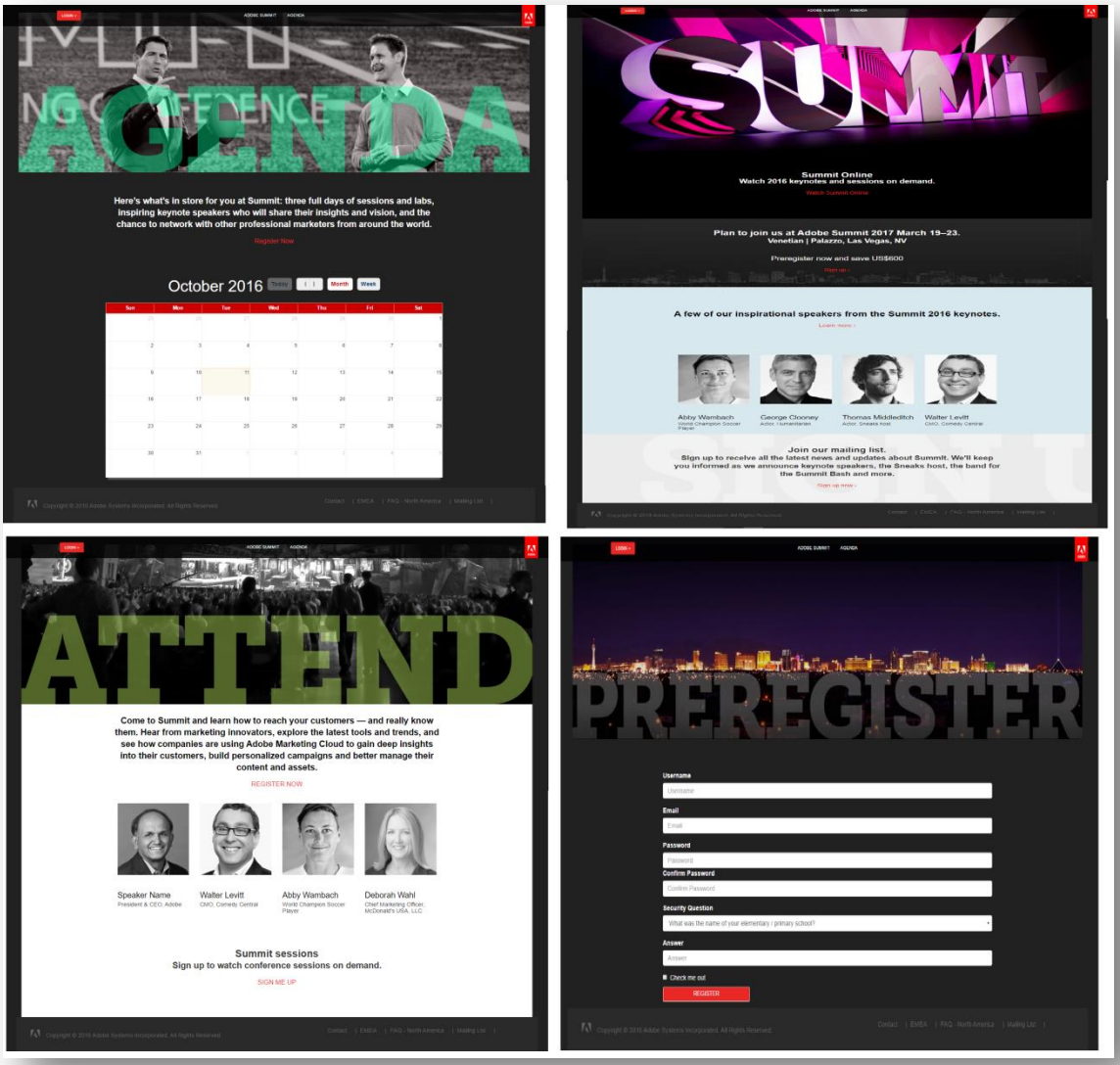


Figura 9 – Las cuatro páginas que forman el sitio web –excluyendo la aplicación móvil- del caso de estudio. Fuente: Autoría Propia.

3.3.1 Home Page

La página *Home Page* la podemos observar en la esquina superior derecha en la Figura 9. No ofrece ninguna funcionalidad específica, sin embargo despliega una gran cantidad de contenido, entre el cual se encuentran los oradores –a partir de ahora *speakers*- destacados del Summit.

3.3.2 Agenda

En la página *Agenda* todos los usuarios (ya sea que estén o no logueados al sistema) deben ser capaces de visualizar los diferentes eventos disponibles en el calendario -como se puede observar en la esquina superior izquierda de la figura Figura 9- .Los usuarios logueados pueden inscribirse al evento o eliminar su suscripción, creando o eliminando una reserva. Por tanto, los autores deben contar con la posibilidad de crear o eliminar eventos dentro del calendario.
Un **evento** está compuesto por:

- Título.
- Descripción.
- Fecha y hora de Inicio.
- Fecha y hora de fin.
- Capacidad.
- Tipo de evento.
- *Speaker* asignado.

El **tipo de evento** es utilizado para resaltar con diferentes colores en el calendario, básicamente estos podrán ser catalogados como:

- Normal.
- Important.
- Warning.
- Custom.

Un **speaker** está compuesto por:

- Nombre.
- Profesión.
- Fotografía.

Una **reserva** está compuesta por:

- Un evento.
- Nombre de un usuario registrado al sistema.

Como podemos ver en la Figura 10 la relación y restricciones entre las entidades descriptas anteriormente son:

- Un *speaker* puede tener cero o más eventos relacionados.
- Un evento debe tener un único *speaker* asociado.
- Un evento puede tener de cero a un máximo de n reservas, donde n es igual a la capacidad definida del evento.
- Una reserva debe tener un único evento relacionado.
- Una reserva puede tener un único usuario relacionado.

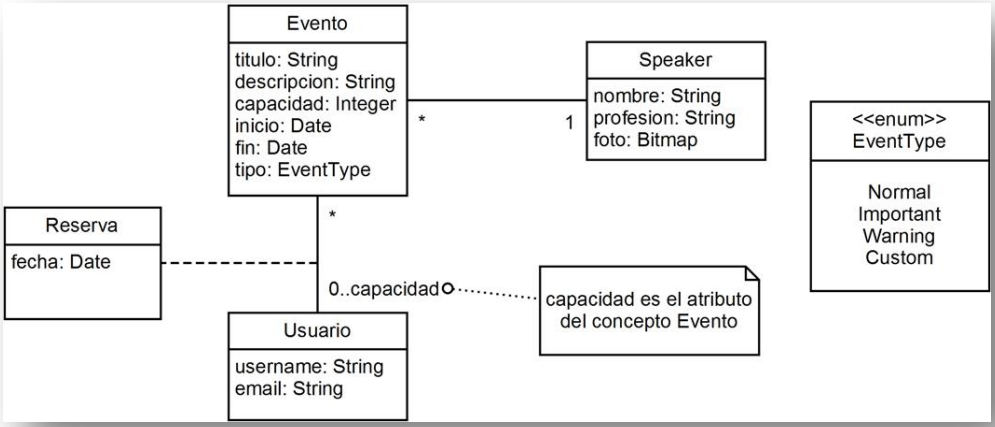


Figura 10 – UML de conceptos del calendario. Fuente: Autoría Propia.

La creación de reservas tiene las siguientes restricciones:

- Un usuario puede realizar una única reserva a un evento.
- Los usuarios sólo pueden crear o eliminar reservas si la fecha hora de comienzo del evento es posterior a la fecha hora actual.

3.3.3 Registration

La página de registro de usuario –tal como se muestra en la esquina inferior derecha de la figura Figura 9- permite a un usuario no identificado darse de alta al sistema mediante un formulario, y así poder acceder a la funcionalidad de agendarse a los diferentes eventos.

3.3.4 Adobe Attend

La página se puede observar en la esquina inferior izquierda de la figura Figura 9. No ofrece ninguna funcionalidad específica. Pero de forma similar a la Home Page, despliega contenido, entre el cual se presenta otro conjunto de speakers destacados.

3.4 Móvil

El caso de estudio consta de una aplicación móvil la cual debe permitir a los usuarios:

- Identificarse mediante un nombre de usuario y una contraseña.
- Los usuarios identificados deben poder registrarse o darse de baja a un evento específico, cumpliendo con las restricciones antes descritas.

Los eventos mostrados en la aplicación móvil deben estar sincronizados con el sitio web. La aplicación móvil permite todas las funcionalidades anteriores mientras se tenga acceso a internet.

3.5 Definición de componentes

A partir de los documentos anteriores y el análisis de los requerimientos, realizaremos la descomposición del mismo en componentes de contenido. La descomposición está basada en maximizar la reutilización de los componentes definidos, minimizando la cantidad de los mismos.
Definimos cinco componentes de contenido para poder cumplir con los requerimientos, ya que con los mismos es posible construir todo el sitio web del caso de estudio.

3.5.1 Topnav

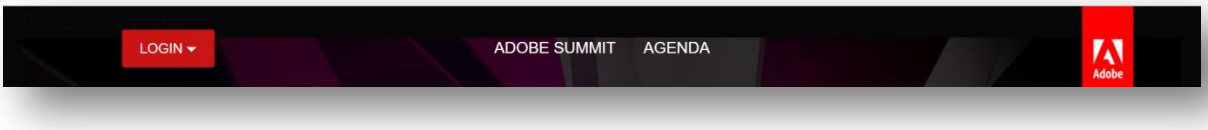


Figura 11 – Captura de pantalla del Topnav del sitio web con el *log in box* cerrado. Fuente: Autoría Propia.

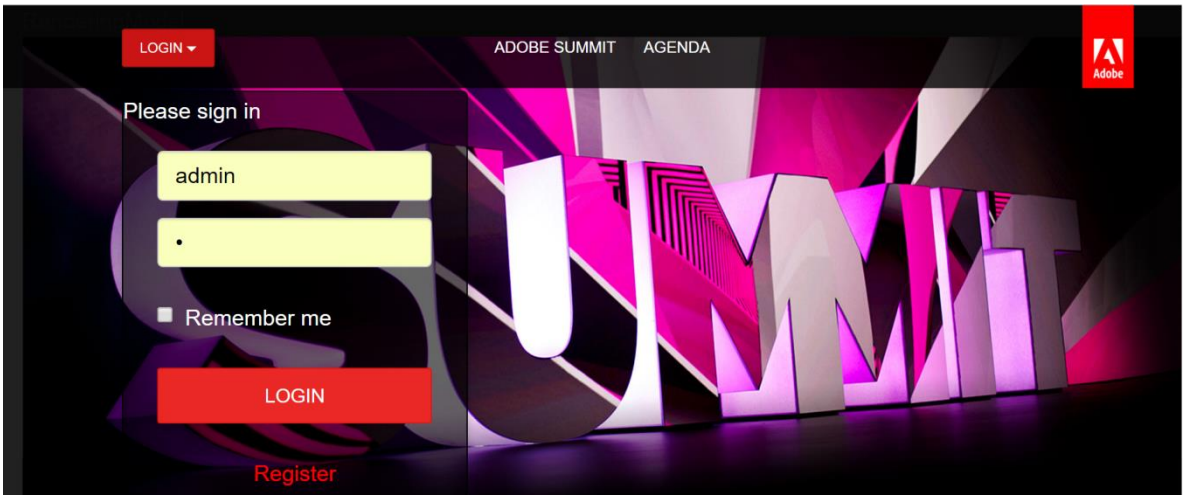


Figura 12– Captura de pantallas del Topnav del sitio web con el *log in box* abierto. Fuente: Autoría Propia.

Este componente es el encargado de manejar la navegación del sitio web y la identificación de los usuarios. Los links a las páginas deben ser cargados dinámicamente por el navegador y de forma automática (se encuentra de forma fija en la parte superior de todas las páginas del sitio web).
Se debe poder configurar:

- El link a donde apunta el logo que se encuentra a la derecha en la Figura 11.
- Los textos que podemos observar en la Figura 12 dentro del *log in box*.

A partir de este componente el usuario debe ser capaz de loguearse al sistema y desloguearse.

3.5.2 Footer

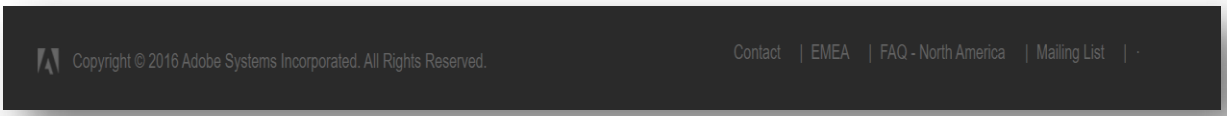


Figura 13 – Captura de pantalla del *footer* del sitio web. Fuente: Autoría Propia.

El mismo se encuentra fijo en la parte inferior de todas las páginas que componen el sitio web.
Los autores deben ser capaces de modificar:

- La imagen que se encuentra en la esquina inferior izquierda de la Figura 13.

- El texto que se encuentra contiguo al logo del *Footer*.
- Un conjunto de enlaces, que pueden ser agregados en el orden y cantidad que prefiera el autor. Cada enlace tiene definido:
    - Un texto.
    - Una URL que puede ser externa o interna a la página.

3.5.3 Banner

Figura 14 – Captura de pantalla de un banner del sitio web. Fuente: Autoría Propia.

Figura 15 – Captura de pantalla de un banner del sitio web. Fuente: Autoría Propia.

Debe contar con una imagen de fondo como se puede observar en las Figura 14 y Figura 15, la cual puede ser establecida por el autor.

3.5.4 Jumbo

Este componente se encuentra en todas las páginas del sitio web pero con distintas configuraciones como vemos en las Figura 16, Figura 17 y Figura 18.

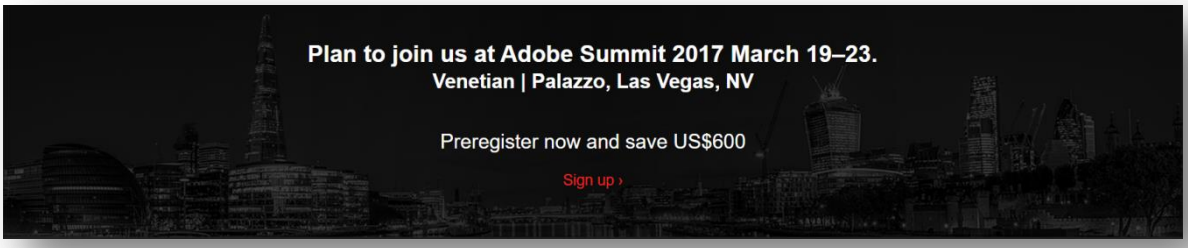


Figura 16 – Captura de pantalla de un componente jumbo en el sitio web. Fuente: Autoría Propia.

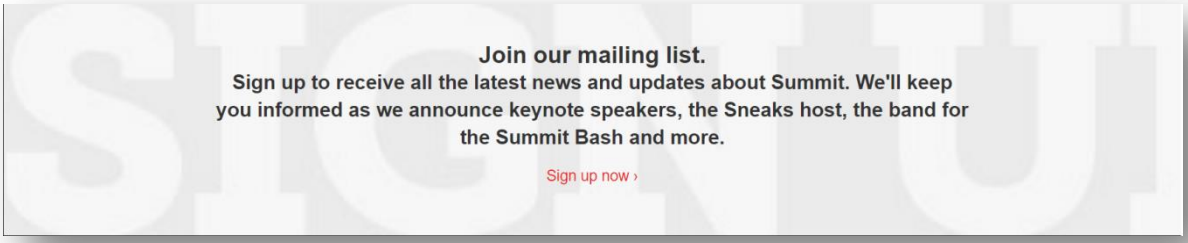


Figura 17 – Captura de pantalla de un componente jumbo en el sitio web. Fuente: Autoría Propia.

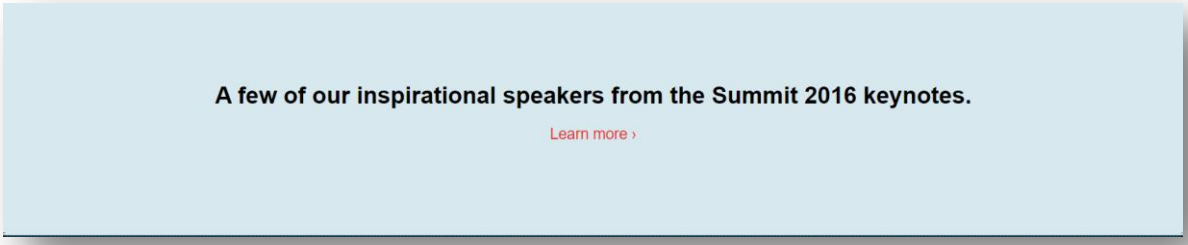


Figura 18 - Captura de pantalla de un componente jumbo en el sitio web. Fuente: Autoría Propia.

El componente es totalmente configurable por el autor, el cual cuenta con la siguiente información para la edición del mismo:

- Título.
- Subtítulo.
- Texto.
- Texto del link y URL del mismo.
- Imagen de fondo.
- Color de fuente.
- Color de fondo (en caso de no contar con imagen).

3.5.5 User registration

Este componente es el encargado de dar de alta a usuarios del sitio web en el sistema.

Figura 19 – Captura de pantalla del registro de usuario del sitio web. Fuente: Autoría Propia.

El autor debe ser capaz de modificar todos los textos que se pueden observar en la Figura 19.

3.5.6 Speaker Box

El componente *speaker box* se encuentra en las páginas *Home Page* y *Adobe Attend* con dos diferentes configuraciones, tal como muestran la Figura 20 y Figura 21.

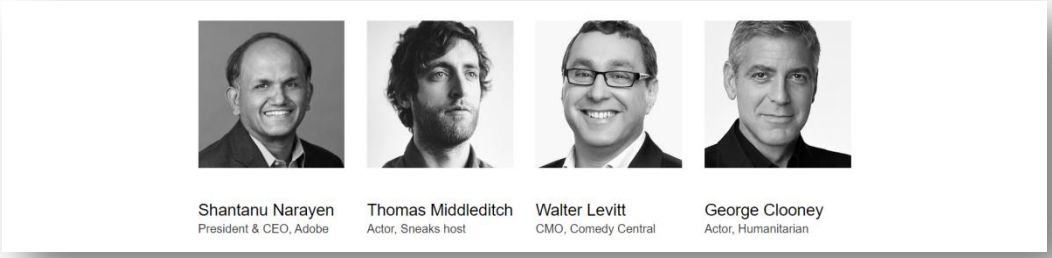


Figura 20 – Captura de pantalla de dos speaker box diferentes del sitio web. Fuente: Autoría Propia.

El autor debe ser capaz de configurar las siguientes propiedades del componente:

- Color de fondo
- Conjunto de cuatro *speakers*, cada uno cuenta con:
  - Texto para el nombre.
  - Texto para su profesión .
  - Imagen.
- El autor podrá cambiar los *speakers*, así como su orden de aparición.

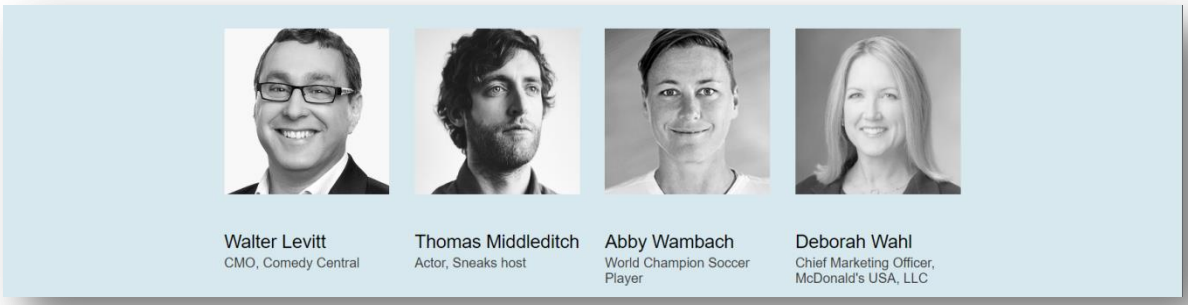


Figura 21 – Captura de pantalla un speaker box del sitio web. Fuente: Autoría Propia.

3.5.7 Calendar

El componente *Calendar* es el encargado de la administración de eventos y reservas del sitio web. El usuario del sitio web debe ser capaz de recorrer el calendario pudiendo acceder a los diferentes eventos, y dependiendo de las restricciones antes descritas puede realizar dos acciones: registrase a un evento generando una reserva o baja a un evento eliminando una reserva. Tal como vemos en la Figura 22, el calendario puede mostrar los eventos con diferentes colores que están asociados al tipo de evento.

El autor debe ser capaz de:

- Crear, eliminar o modificar eventos del calendario, definiendo para cada evento:
  - Titulo
  - Descripción
  - Fecha y hora de Inicio.
  - Fecha y hora de fin.
  - Capacidad.
  - Tipo de evento.
  - Tiene un *Speaker* asignado, este speaker tiene que tener los mismos atributos que los *speakers* definidos en el *Speaker Box*.

Para el usuario identificado en el sistema tiene que ser posible, además:

- Inscribirse o darse de baja a un evento seleccionado (siempre que se respeten las restricciones antes descriptas).

Tanto para el usuario (ya sea identificado o no en el sitio web) como para el autor deberá ser posible poder ver la información de un evento seleccionado.

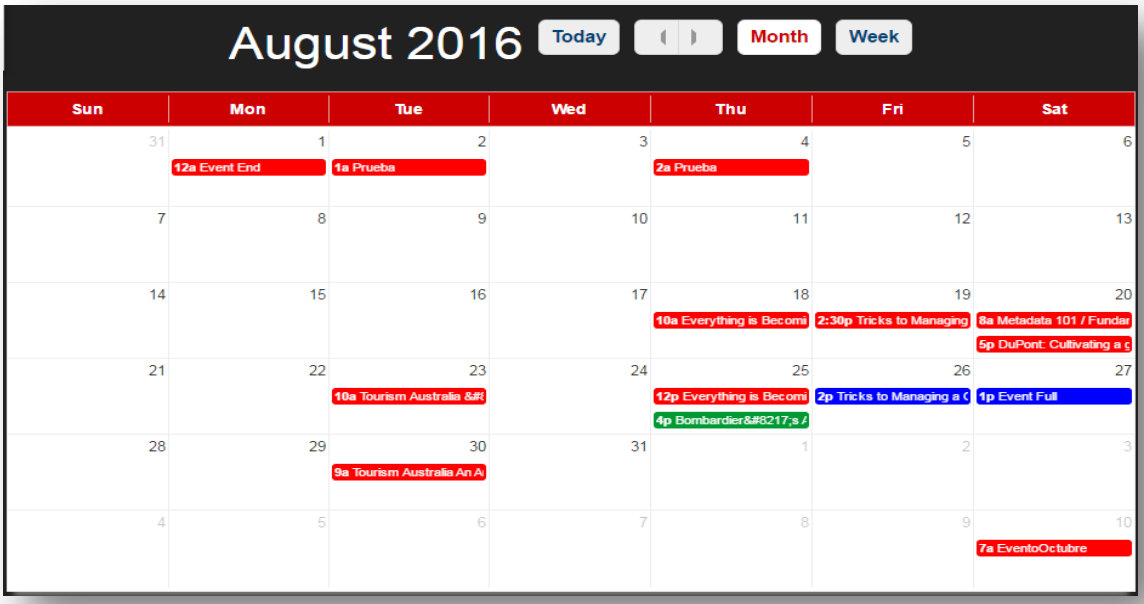


Figura 22 – Captura del calendario del sitio web. Fuente: Autoría Propia.

3.6 Resumen de capitulo

El caso de estudio definido anteriormente está basado en un sitio web de Adobe el cual es utilizado para presentar su evento Summit -http://summit.adobe.com/na/ (último acceso: 14 Agosto, 2016)-. El sitio web a implementar en el caso de estudio brindará la posibilidad a los autores de administrar un conjunto de eventos, los cuales tienen un *speaker* –orador- asociado. Además, los usuarios tendrán la posibilidad de registrarse en el sitio web y realizar reservas en los eventos publicados. La implementación del sitio web antes mencionado utilizará siete componentes, los cuales serán implementados en dos plataformas de WCMS –AEM y Sitecore- partiendo de los mismos requerimientos.

Capítulo 4: Introducción a AEM

El presente capítulo es un resumen del *Anexo 2 Documentación técnica de AEM.* El objetivo del mismo es presentar una introducción técnica, la cual debe ser suficiente para comprender la implementación del caso de estudio en AEM que se presenta en el próximo capítulo.

4.1 Arquitectura

AEM es un WCMS que se autodefine [22] como un sistema cliente-servidor ofreciendo una plataforma web cuya función oscila entre la construcción, gestión y el despliegue de sitios web comerciales, así como servicios relacionados. Combina una serie de funciones a nivel de infraestructura y de aplicación agrupadas en un único paquete integrado.
Es una plataforma que permite entregar contenido digital a través de distintos canales y proveer un entorno propicio para que los autores cuenten con soporte para realizar edición en línea de contenido.
El sistema está dividido en tres partes:

- **Web Application Server**
  AEM puede ser desplegado en modo *standalone* ya que cuenta con un servidor web integrado, el cual utilizaremos para implementar el caso de estudio o como una aplicación con cualquier servidor web especifico–utilizado para ambientes de producción-.

- **Web Application Framework**
  AEM incorporó el *framework* Sling que simplifica la escritura de servicios *RESTful* y las aplicaciones web orientadas al contenido –se definirá más adelante-.

- **Repositorio de Contenido**
  AEM incluye un *JAVA Content Repository* (a partir de ahora JCR), una base de datos especializada para persistir contenido.

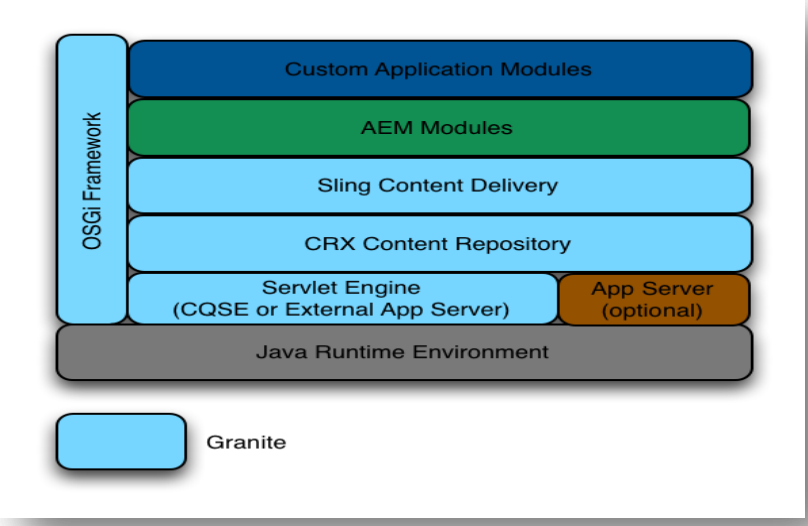


Figura 23 – Esquema de arquitectura de AEM. Fuente: [23].

Si detallamos el *stack* de tecnologías [24] que utiliza AEM, partiendo de la base de la Figura 23 encontramos la plataforma JAVA, cuya versión recomendada para AEM 6.2 es JAVA 1.8

4.1.1 Granite

Es el *framework* desarrollado por Adobe que es encargado de sincronizar e integrar las diferentes tecnologías que componen el *stack* del AEM. Granite entre otras prestaciones, ofrece:

- Exportación e importación de paquetes de contenido.
- Provee la interfaz gráfica web para poder realizar todas las operaciones posibles del repositorio de contenido JCR.
- Brinda un ambiente web –CRXLite- para facilitar la tarea del desarrollador, brindando la posibilidad de generar código directo en la plataforma.

4.1.2 OSGi framework

Define la arquitectura para desarrollar y desplegar aplicaciones modulares y bibliotecas. Los contenedores OSGi [25] permiten descomponer la aplicación en módulos individuales (son archivos jar que cuentan con meta información y son llamados paquetes en terminología de OSGi). También ofrece la capacidad de poder cargar o descargar dichos módulos (que llamaremos *bundles*) sin la necesidad de reiniciar el servidor.

4.1.3 Repositorio de contenido CRX

Absolutamente todos los datos dentro de AEM se almacenan en el repositorio de contenido CRX, el cual constituye una implementación creada por Adobe de la API JCR 2.0 (JCR) [26]. A su vez, parte del código base de CRX está basado en el proyecto *open source* de Apache Jackrabbit [27].
Un repositorio basado en la API JCR es un tipo específico de base de datos diseñado para el acceso de datos jerárquicos -siendo éstos últimos no estructurados o semi-estructurados-. De esta manera, combina las características de una base de datos relacional tradicional con los de un sistema de archivos convencional.

4.1.4 Sling Framework

Se trata de un framework web basada en los principios REST para almacenar y administrar contenido. Sling [28] utiliza un repositorio JCR, como Apache Jackrabbit; o bien, en el caso de AEM, el repositorio de contenido CRX como su base de datos para administrar, almacenar y enviar contenido.

El framework Sling tiene incorporado al framework OSGi y juntos proporcionan un entorno de ejecución dinámico, donde los paquetes de código y contenido pueden cargarse, descargarse y reconfigurarse en tiempo de ejecución.

4.2 Conceptos básicos

Debido al gran tamaño de la plataforma AEM -y considerando que algunas de las funcionalidades que ofrece quedaron por fuera del alcance del proyecto- nos centraremos en definir los conceptos y herramientas que atañen tanto a la construcción de componentes de contenido como a la creación de contenidos que ofrece la plataforma.
Definiremos en primer lugar algunos conceptos básicos para poder luego profundizar sobre la temática.

4.2.1 Ítem, Nodos y Propiedades

Comenzaremos definiendo los tres conceptos básicos del repositorio de AEM [29].

Un ítem [29] es la unidad básica del repositorio, es decir, una interfaz que tiene definida dos implementaciones: nodos y propiedades.
Los nodos [29] definen la estructura del contenido. Pueden tener cero o más ítems como hijos.
La propiedad más relevante de los nodos es *jcr: primary Type* la cual define el tipo de nodo (es especificado en el momento de creación del nodo y es inmutable). AEM trae un conjunto de tipos de nodos ya definidos. Mediante configuración avanzada es posible crear nuevos tipos. En los ambientes de Sling un nodo también es denominado como un recurso.

Las propiedades [29] en cambio, persisten los datos específicos y la metadata del contenido. Una propiedad no puede tener hijos pero puede tener cero o más valores definidos. Las propiedades tienen ciertos atributos que deben ser definidos:

- ***Name***
  El nombre de una propiedad la identifica de forma local al nodo padre.
- ***Type***
  Define el tipo de dato que va a ser persistido por la propiedad, por ejemplo *String*, *Long*, *Double* y *Date*, entre otros, dichos tipos son brindados por AEM.

En la Figura 24 vemos una sección de la interfaz gráfica que ofrece AEM del repositorio CRX. Notamos el nodo *text* contiene varios ítems hijos y a su vez podemos ver las distintas propiedades que tiene definidas.

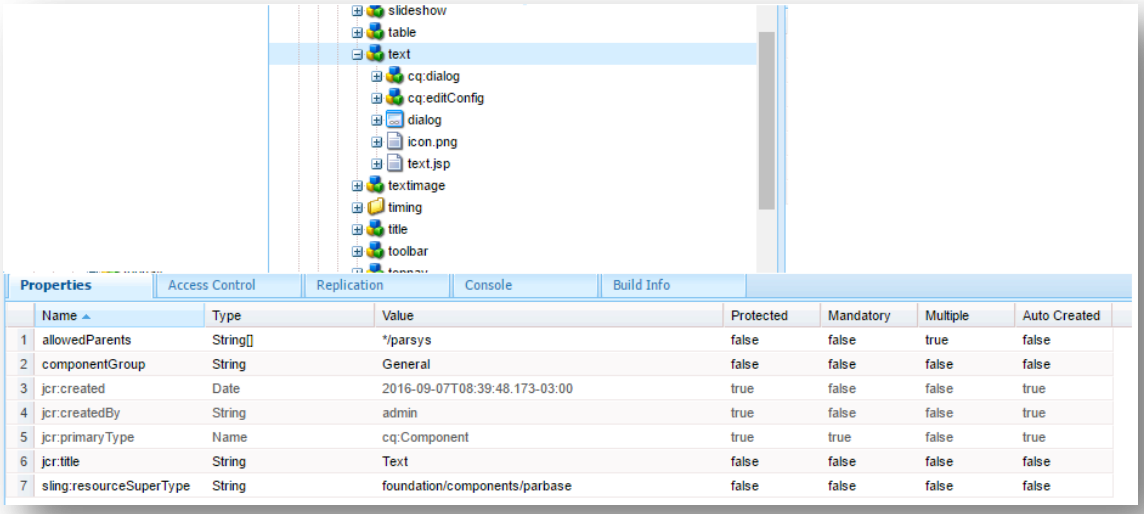


Figura 24 – Captura de pantalla recortada de la interfaz web del CRX. Fuente: Autoría propia.

4.2.2 Componente

Un componente de contenido desde el punto de vista de AEM es un nodo que agrupa *scripts*, archivos, datos y propiedades, que tienen como objetivo renderizar y administrar una porción de contenido. En el proyecto nos centraremos en los componentes desarrollados para que sean utilizados por los autores; estos forman parte fundamental de un proyecto WCMS y pueden variar desde simples títulos hasta un navegador complejo con varios niveles de paginación. Aunque AEM ofrece la ventaja de tener un conjunto pre definido de componentes definidos y listos para su uso,

en la mayoría de los casos no se ajustan a los requerimientos del sitio web. De todas formas, resultan útiles para ejemplificar el funcionamiento *per se* de la plataforma o extender su funcionalidad-como se verá más adelante-.

En la Figura 25 podemos observar como se ve un componente pre definido por AEM en el entorno de autores llamado *Text and Image,* sin ningún contenido definido. Este componente permite al autor, como sugiere su nombre, establecer un texto y una imagen para mostrar en una página. Los pequeños íconos con diferentes herramientas que se observan en la imagen se despliegan al hacer clic en el mismo, permitiendo copiarlo, cortarlo, eliminarlo o acceder a su configuración, entre otras funciones.

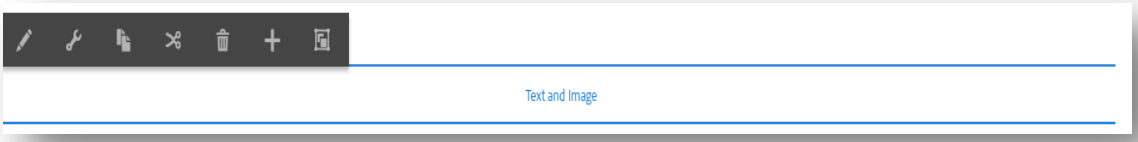


Figura 25 – Captura de pantalla del componente *Text and Image* en el ambiente de autor. Fuente: Autoría propia.

4.2.3 Widget

En AEM todas las entradas de usuarios son manejadas por un tipo de nodo denominado *widget*. Éstos usualmente son utilizados para editar una pieza de contenido, y los diálogos son construidos mediante una combinación de los mismos. La plataforma cuenta con varios *widgets* definidos, los cuales son utilizados por el desarrollador para incluirlos en los diálogos, o bien, crearlos de forma personalizada. Podemos observar en las figuras Figura 26 y Figura 27 dos ejemplos de *widgets* de tipo imagen y texto respectivamente.

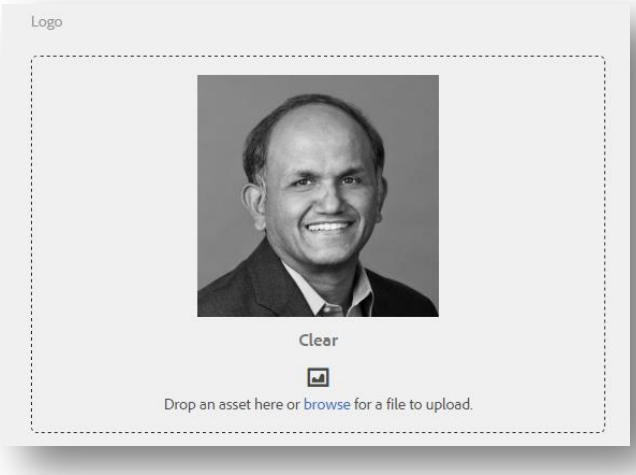


Figura 26 – *Widget* de tipo imagen. Fuente: Autoría propia.

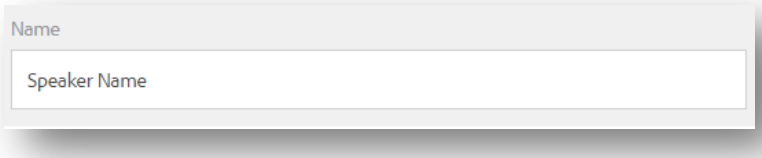


Figura 27 - *Widget* de tipo texto. Fuente: Autoría propia.

4.2.4 Diálogo

Un dialogo es un tipo especial de *widget*.

Para editar el contenido de un componente, AEM utiliza los diálogos definidos por los desarrolladores. Estos diálogos combinan una serie de *widgets* que les brinda a los usuarios un conjunto de campos relacionados al contenido, los cuales son editables. Los diálogos también pueden ser utilizados para editar metadata y configuración administrativa.

Continuando, en la Figura 28, podemos ver el diálogo asociado al componente *Text and Image* antes mencionado. Señalado mediante flechas rojas en la imagen podemos ver los distintos tipos de *widgets* que constituyen este diálogo en particular, permitiendo al usuario definir por ejemplo la imagen que desea mostrar en el componente, el título y la URL asociada a la imagen.

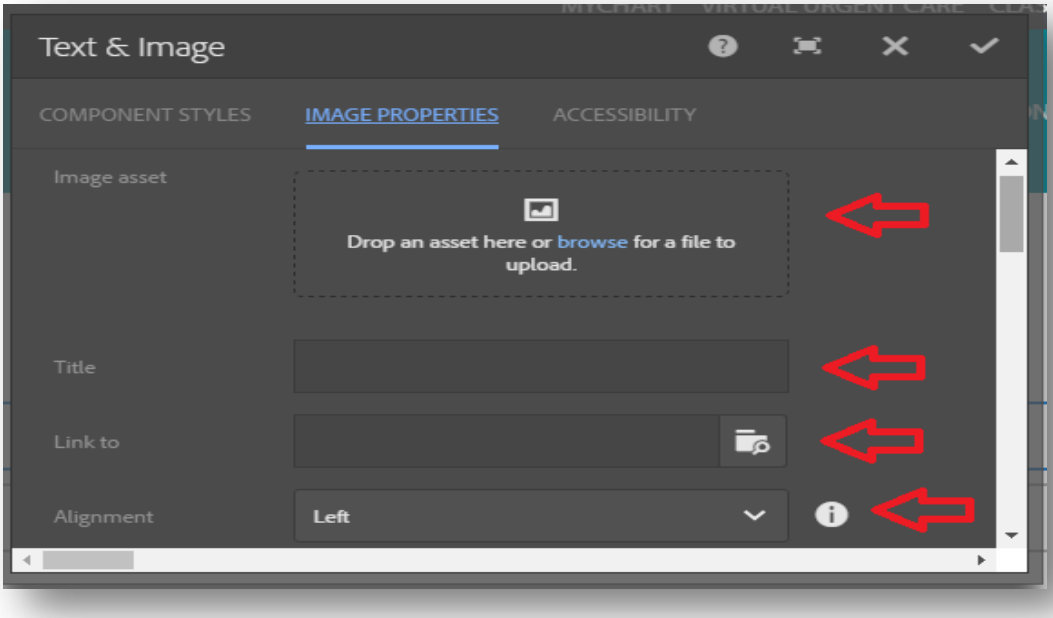


Figura 28 – Captura de pantalla del dialogo del componente *Text and Image*. Fuente: Autoría Propia.

4.2.5 Template

Un *template* define un tipo de página. Al crearse una página en AEM debe seleccionarse un *template* de pertenencia. De esta forma, un *template* conforma una jerarquía de nodos que contienen la misma estructura que debe tener la página que está definiendo, pero sin ningún contenido definido.

4.2.6 Página

Todas las páginas de un sitio web implementado en AEM tienen como base un *template*. De esta forma se pueden crear plantillas HTML para que sean reutilizadas por las páginas.

4.2.7 Paragraph System

El *Paragraph System* [30] (a partir de ahora *parsys*), es un componente pre definido por AEM que permite a los autores agregar otros tipos de componentes a una página. También permite mover, copiar, cortar y pegar los distintos componentes contenidos en el *parsys*. Es posible configurar los *parsys* para restringir qué componentes o qué grupo pueden ser incluidos.
En una página pueden existir diferentes *parsys*, cada uno de ellos es identificado con un nombre definido por el desarrollador. A su vez, cada uno de estos *parsys* puede ser configurado de forma individual. En la Figura 29 se puede observar una página de AEM la cual tiene definida dos parsys-recuadros de bordes azules con el texto *'Drag components here'*-.

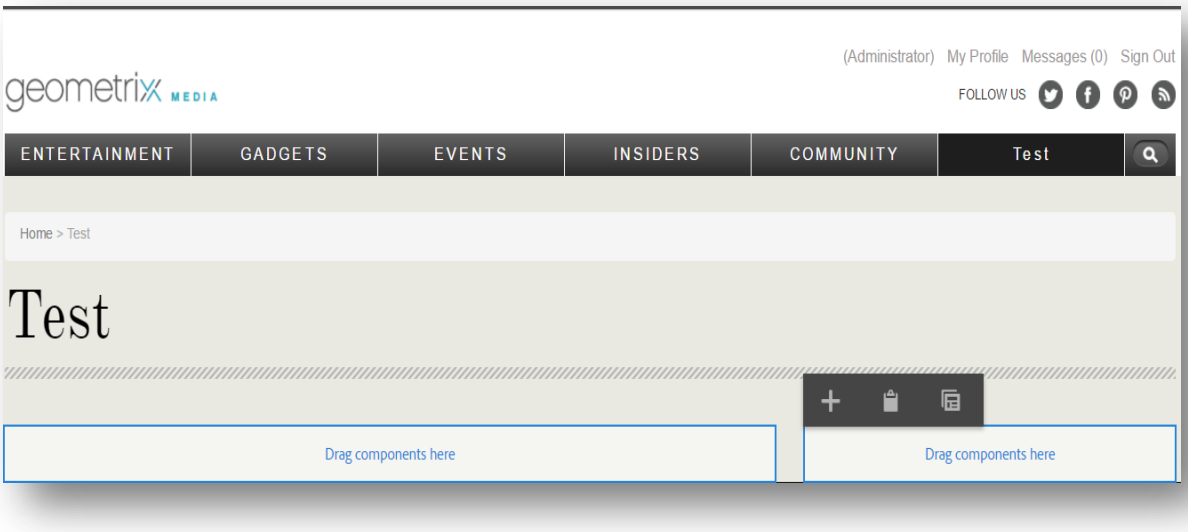


Figura 29 – Página de AEM en el ambiente de Autor con dos *parsys* definidos. Fuente: Proyecto Geometrixx de AEM [31].

4.2.8 Resumen de los conceptos básicos

Resumiendo los conceptos mencionados, podemos observar en la Figura 30 que los *templates* definen a las páginas, pudiendo existir varias de ellas. A su vez, las páginas pueden contener cero o más *parsys*, dentro de los cuales, los autores pueden insertar cero o más componentes. También podemos observar un diagrama que ejemplifica el modelado de conceptos -Figura 31-.

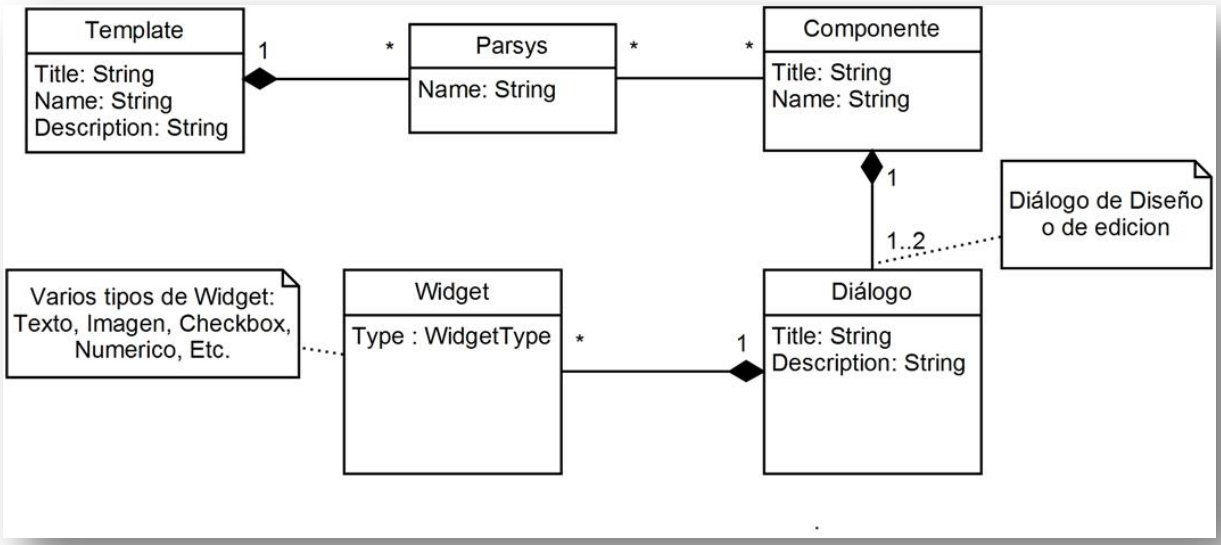


Figura 30 – Modelado de conceptos de AEM. Fuente: Autoría Propia.

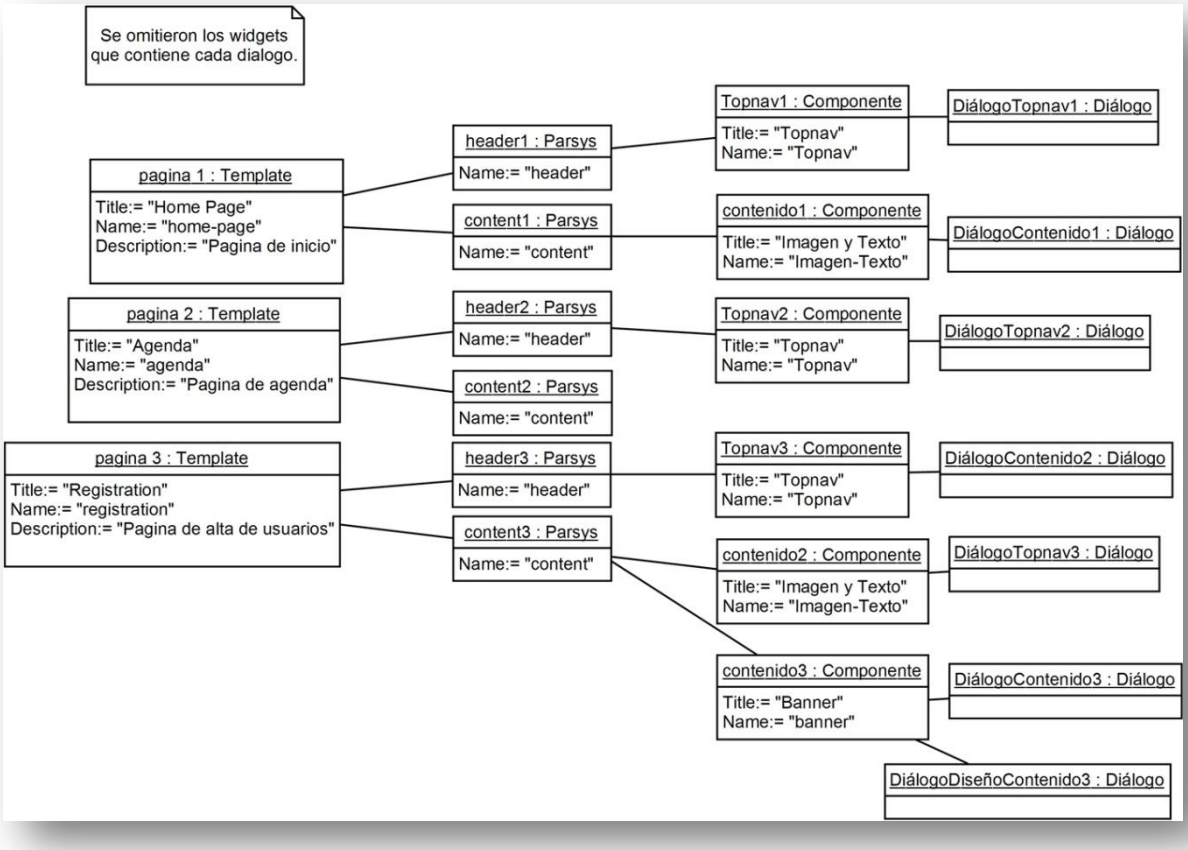


Figura 31 - Diagrama de conceptos de AEM. Fuente: Autoría Propia.

4.3 Ambiente de Desarrollo

Las herramientas que vamos a utilizar para desarrollar el caso de estudio son:

- IDE *Eclipse* versión 4.5.1.
- Apache Maven versión 3.3.9.
- JAVA versión 1.8.0_101.
- GIT versión 2.8.3.windows.1.
- *Brackets* versión 1.7.

4.3.1 Eclipse

La decisión de utilizar *Eclipse* como IDE surge a partir de la disponibilidad del plug-in *AEM Developer Tool* [32] que facilita el desarrollo de proyectos para la plataforma en AEM. A partir del *plug-in* antes mencionado podemos crear un arquetipo de proyecto [33] diseñado bajo las recomendaciones de Adobe, con una estructura modular dividida en carpetas. Cada carpeta es un proyecto que utiliza el *framework* Maven. Señalaremos los tres proyectos más significativos del arquetipo para nuestro caso de estudio:

- ***core***
  El proyecto *core* contiene todas las clases JAVA utilizadas por el sitio que vamos a desarrollar. Cuando nosotros desplegamos nuestro proyecto a un servidor AEM el JAR generado a partir de dicho proyecto es cargado en el servidor como un *bundle* de OSGi. De esta forma es posible acceder a las clases definidas dentro de dicho *bundle*.
- ***ui.apps***
  En ui.apps encontramos todos los componentes que vamos a definir en nuestro proyecto. Estos componentes son representados por carpetas y dentro de las mismas encontramos los *script*s, archivos HTML y otros diferentes archivos que pertenecen a los nodos correspondientes en AEM. Las propiedades de los nodos son representadas mediante archivos XML que son generados automáticamente por la plataforma. Nosotros vamos a utilizar el IDE *Brackets* creado por Adobe para modificar los archivos definidos en este proyecto en particular.
- ***ui.content***
  Similar a la estructura proyecto ui.apps, tenemos el proyecto ui.content el cual guarda el contenido del sitio web correspondiente a nuestro proyecto.

A su vez el arquetipo divide a los componentes en dos grandes grupos:

- **Componentes estructurales**
  Son parte de la estructura del sitio web y comparten la misma configuración para todas sus instancias incluidas en el mismo.
- **Componentes de contenido**
  A diferencia de los componentes estructurales, cada instancia puede contener una configuración y contenido único.

4.3.2 Brackets

Se trata de un IDE desarrollado por Adobe para la edición de archivos que atañan al desarrollo *front end*. Lo particular de este *IDE* es que cuenta con un *plug-in* para AEM que permite descargar o desplegar cambios al servidor a nivel de archivo –a diferencia de Maven que es a nivel de proyecto-. Nuevamente debemos ingresar la dirección del servidor de AEM, en la cual queremos descargar o desplegar los cambios realizados junto con un usuario administrador y su contraseña. Debido a que como ya mencionamos es posible la sincronización con el servidor a nivel de archivos, utilizaremos esta herramienta para el desarrollo de HTML, *scripts* o estilos, que se encuentran en el proyecto *ui.apps*.

4.3.3 Sightly

A partir de la versión 6.0, AEM incorpora un lenguaje de modelo HTML llamado Sightly para sustituir los archivos JSP de la solución. El mismo brinda funcionalidades que facilitan la integración de los archivos HTML con AEM. Esto permite, por ejemplo, acceder a propiedades de nodos o del componente al cual pertenece el archivo HTML, instanciar una clase JAVA de cualquier *bundle* cargado en el servidor a partir de su ruta completa de paquetes, recorrer listas o importar otros archivos.
Dicho lenguaje se utilizará para el desarrollo del caso de estudio. Esto es, básicamente un HTML con la extensión de las funcionalidades ya mencionadas.

4.3.4 CRX Lite

AEM ofrece una interfaz gráfica a través del servidor web para su repositorio CRX. Tal como observamos en la Figura 32, podemos acceder a todo el repositorio del servidor, recorriendo el contenido del mismo. Asimismo, permite crear o eliminar nodos, modificar sus propiedades y editar archivos, entre otras opciones.
Es precisamente desde esta interfaz que comenzamos el desarrollo de un componente, creando el nodo correspondiente para luego descargarlo en el IDE *Brackets* y continuar editando los archivos que definen el renderizado del mismo.

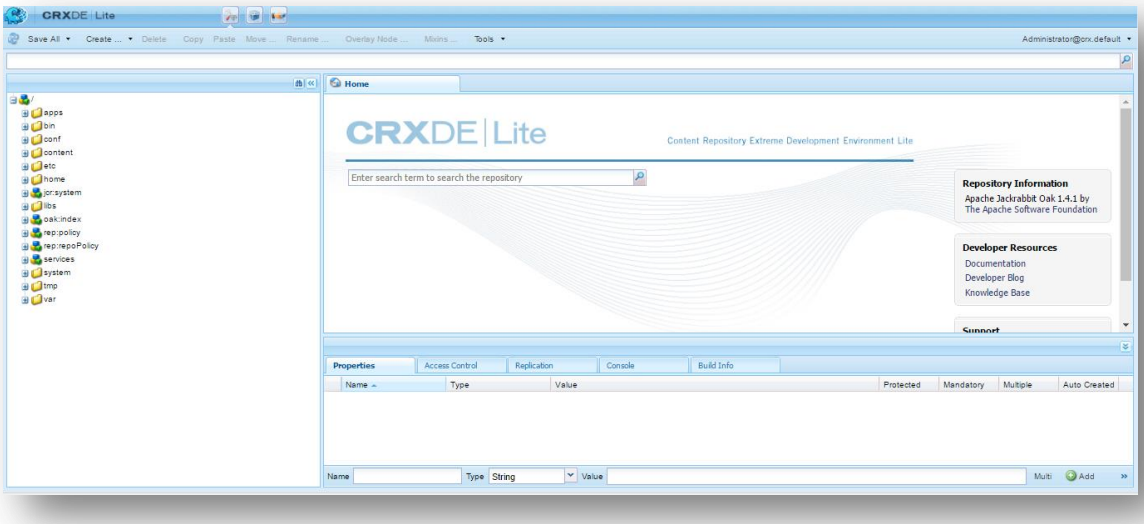


Figura 32 – Captura de pantalla de la interfaz gráfica que ofrece AEM del repositorio CRX. Fuente: Autoría Propia.

4.3.5 Git

Con el fin de generar un ambiente colaborativo, utilizaremos GIT y el repositorio gratuito *Bitbucket*. El proyecto anteriormente definido resulta completamente funcional, por lo que prescinde de herramientas extra para ser desplegado y compartido por otro desarrollador. Esto potencia la creación de un ambiente colaborativo.

4.3.6 Resumen del ambiente de desarrollo

Nuestro ambiente de desarrollo se podría dividir en tres partes. Tal como vemos en la Figura 33, nuestro proyecto contiene todo el código del sitio web que será editado con dos IDEs, *Brackets* y *Eclipse*. De este modo, utilizaremos Maven para desplegar los cambios que son realizados del código a nuestro servidor AEM (local). A su vez, utilizaremos la interfaz web *CRX Lite* que provee AEM para crear nodos y definir sus propiedades -estos cambios serán descargados al código del proyecto mediante el *plug-in* que ofrece Adobe en el IDE *Brackets*-. Por último, persistiremos el proyecto en el repositorio de *Bitbucket* utilizado GIT.

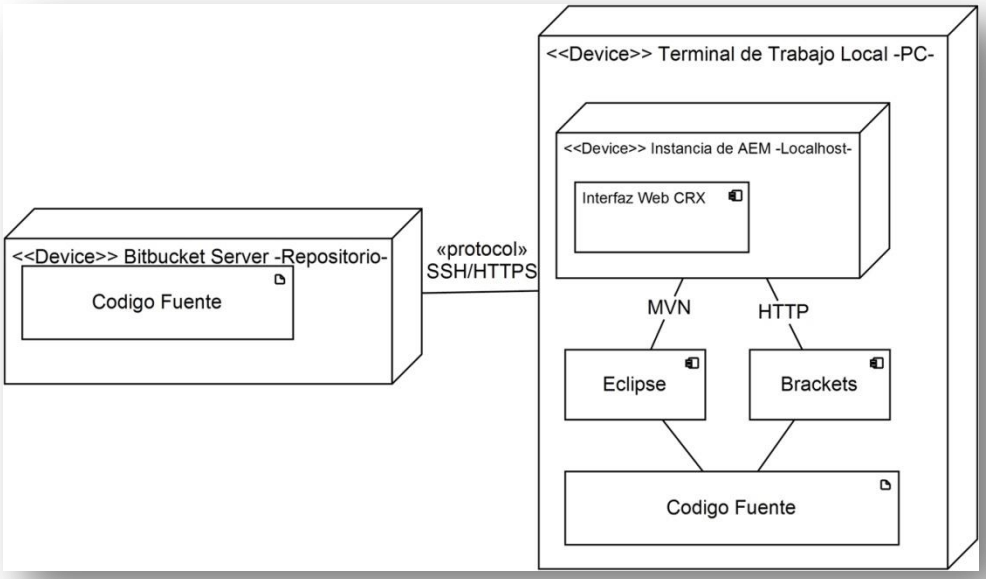


Figura 33 – Modelado de ambiente de desarrollo utilizado. Fuente: Autoría Propia.

4.4 Ambiente de Autor

El ambiente cuenta con diferentes modos, de los cuales vamos a nombrar tres que son los más utilizadas por los autores:

- **Modo Edición:**
  El modo de edición es donde el autor puede insertar, eliminar o mover los componentes de contenido en los *parsys* que contiene la página. También se le permite acceder a los diálogos de los componentes y editar su contenido.
- **Modo Diseño**
  El modo diseño le permite a los autores –con permisos suficientes- configurar cuales son los componentes habilitados para cada *parsys*. También permite acceder al diálogo de

diseño de los componentes ya agregados a la página. El contenido y los parámetros definidos en dichos diálogos tienen un comportamiento diferente a los definidos en el modo edición (más adelante detallaremos la diferencia).

- **Modo Pre visualización**
  El modo pre visualización muestra a la página renderizada tal como si estuviera publicada y siendo vista por el usuario final.

4.5 Conceptos para el desarrollo de componentes

4.5.1 Estructura

Existe una configuración básica para que un componente de AEM pueda funcionar de forma correcta. Las principales propiedades de un componente de AEM son:

- **jcr:primaryType**
  Esta propiedad define el tipo de nodo. En el caso de los componentes de AEM, su tipo es *cq:component*.

- **jcr:title**
  Establece el título con el cual el autor identificará al componente.

- **componentGroup**
  Define a qué grupo pertenece el componente. Básicamente su función es facilitar la tarea del autor, agrupando los componentes en diferentes conjuntos. También se puede utilizar para restringir el acceso a cierto grupo de componentes -si un componente no tiene esta propiedad asociada, se lo asociada al grupo llamado *undefined* – o para ocultar a los autores ciertos componentes que el desarrollador considere necesario, definiéndolos en la categoría *hidden*.

Adicionalmente, el componente debe tener por lo menos dos nodos hijos: uno que defina el diálogo, y un segundo archivo que represente el código que será utilizado para el renderizado (generalmente este archivo es HTML o JSP):

- **cq:dialog**
  Este nodo contiene una jerarquía de nodos hijos que definen el diálogo y los diferentes *widgets* que lo componen. Su *jcr:primaryType* es *nt:unstructure.* Es posible acceder al diálogo desde el modo de edición del ambiente de autor.

- **Archivo de renderizado**
  Este archivo debe tener el mismo nombre que el nodo del componente, puede ser un HTML o JSP. Cuando un componente es insertado en una página por el autor, por defecto AEM buscará este archivo para renderizarlo.

4.5.1.1 Diálogo de diseño

También es posible definirle un diálogo de diseño al componente. La jerarquía y estructura de especificación es idéntica a un *cq:dialog*, la diferencia en el repositorio es el nombre, el cual debe

ser *cq:design_dialog.* Dicho diálogo es accedido por el autor en el modo diseño -el contenido definido en este tipo de diálogo es común a todas las instancias del componente, a diferencia del dialogo en el modo edición, el cual es específico para la instancia del componente en el que se está editando-.

4.5.2 Administración de archivos de estilos y scripts

AEM ofrece una funcionalidad especial para administrar los archivos de estilos CSS, LESS y JavaScript individualmente en cada componente, para que luego puedan ser exportados de forma simple dentro del sitio web. Como vemos en la Figura 34, el componente *read-more*, contiene un nodo llamado *clientlibs* [34] (a partir de ahora librerías) cuyo tipo primario es *cq:ClientLibraryFolder*. Dentro del mismo se pueden diferenciar cuatro archivos de los cuales los referentes a su configuración son:

- **css.txt**
  En el archivo *css.txt* se especifica el nombre de los archivos de estilos que deben ser incluidos en el momento que se importe la librería. En la Figura 34, únicamente tiene escrito *read-more.less*.

- **js.txt**
  Análogo al archivo *css.txt*, en *js.txt* se definen los archivos JavaScript que deben ser incluidos cuando se importe la librería. En el ejemplo de la Figura 34, únicamente tiene escrito *read-more.js.*

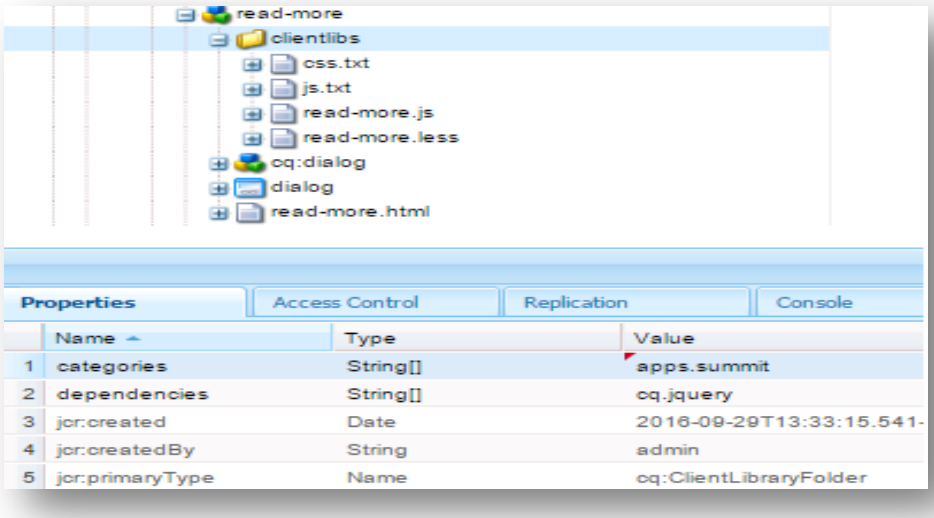


Figura 34 – Captura de pantalla de la interfaz gráfica web del CRX, mostrando un client library. Fuente: Autoría Propia.

Podemos observar dos propiedades con múltiples valores:

- ***categories***
  Dicha propiedad define una o más categorías a la cual pertenece particularmente la librería. Estas categorías pueden ser utilizadas luego de forma simple para importar todos los archivos que incluyen todas las librerías bajo dicha categoría.

- ***dependencies***
  En esta propiedad el desarrollador puede definir cero o más librerías como dependencias, que son cargadas antes de ejecutar cualquier estilo o *script* definido en la misma.

Mediante las librerías los desarrolladores pueden administrar, editar y mantener los archivos de estilos que son utilizados por los componentes ordenadamente dentro de los mismos.

4.5.3 Enfoque de creación de componentes de contenido

Existen tres diferentes enfoques a la hora de crear un componente en AEM: construir el componente de cero, extendiendo la funcionalidad de un componente ya creado o sobrescribir completamente un componente a través de nuestra implementación.

4.5.3.1 Creación desde cero

Este enfoque es utilizado cuando no existe ningún componente dentro del servidor AEM que tenga una funcionalidad similar al componente que debemos desarrollar. Básicamente consiste en construir el componente desde el inicio, en la carpeta correspondiente a nuestro sitio dentro de *apps*.

4.5.3.2 Herencia

AEM ofrece el concepto de herencia entre nodos, con especial énfasis en los componentes [35]. Es un concepto similar a la herencia en el paradigma de programación orientada a objetos. Un componente puede basar su comportamiento en otro, e implementar cierta funcionalidad específica diferente, que se considere necesaria.

4.5.3.3 Sobre-escritura

El último enfoque consiste en sobrescribir completamente un componente ya existente. Esto es posible asignándole al nodo del componente definido en la carpeta *apps* el mismo nombre del cual queremos sobrescribir. De esta forma, dentro de nuestro sitio web todas las referencias hechas a dicho componente serán re direccionadas al nodo antes mencionado.

4.5.4 Persistencia

La mayor parte del contenido definido es guardado en la carpeta *content* de AEM como una jerarquía de nodos. En esta carpeta, cada página creada por el autor es persistida en forma de nodo -con nodos hijos que representan a los *parsys* que contiene la misma-. A su vez, dentro de dichos nodos que hacen referencia a los *parsys*, se encuentran los nodos hijos que representan a las instancias de los componentes de contenido que definió el autor dentro de los *parsys* antes mencionados. Por ultimo dichas instancias guardan los valores definidos por los autores en los diálogos como propiedades.

4.5.5 AEM JAVA USE-API

La API que provee AEM permite una comunicación simple desde Sightly a una clase JAVA. De esta forma la implementación de lógica compleja de negocio queda encapsulada en código JAVA, mientras que Sightly solo se encarga de mostrar los diferentes datos.
Una de las características más utilizadas es la clase abstracta WCMUsePojo, la cual brinda diferentes funcionalidades a las que puede acceder el desarrollador. Entre ellas se encuentra la posibilidad de acceder al nodo en el cual se encuentra el componente o a la página, junto a las propiedades de ambos. Como vemos en la Figura 35, para poder acceder desde *Sighlty* a una clase JAVA ésta debe pertenecer a un *bundle* de OSGi y a su vez extender de la clase WCMUsePojo. Cumpliendo ambas especificaciones mencionadas es posible acceder a otras diferentes

funcionalidades de la API como el *PageManager* que permite buscar y obtener páginas dada una ruta específica o el *QueryCreator*, clase que permite realizar búsquedas de nodos o contenido dentro del repositorio JCR. Todo esto sucede del lado del servidor, el resultado final de la interacción entre *Sightly* y el *bundle* es el HTML, estilos y scripts que ve el cliente en su navegador.

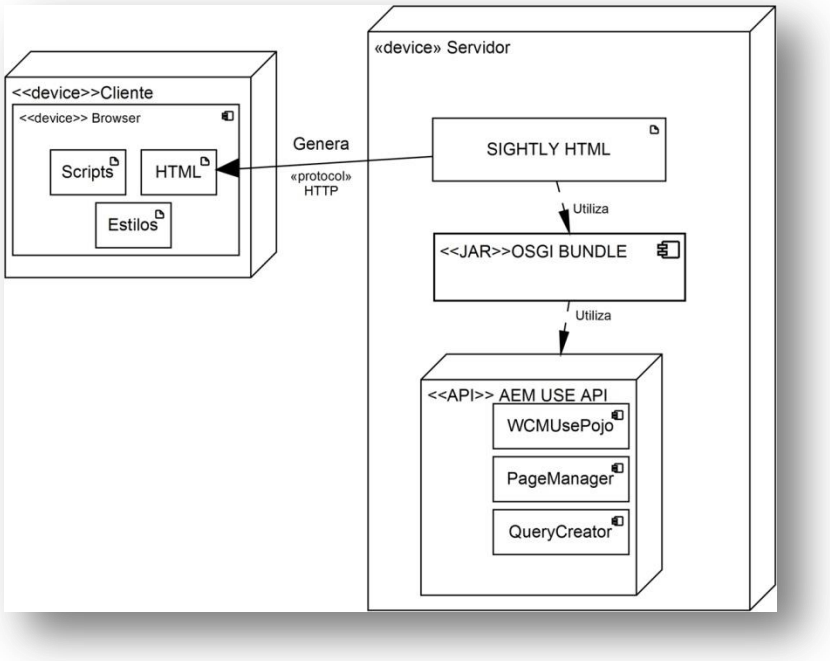


Figura 35 – Modelado de funcionamiento de AEM USE-API. Fuente: Autoría Propia.

4.6 Resumen del capitulo

El capítulo realizó un repaso de los conceptos claves de AEM para lograr comprender la solución del caso de estudio que se presentará en el siguiente capítulo. Entre los conceptos descritos se destacan las páginas del sitio web, las cuales son definidas a partir de *templates* de AEM. A su vez el contenido que despliegan las mismas es dividido en componentes de contenido de AEM, los cuales son los encargados de administrar y desplegar una porción de contenido del sitio web. Estos utilizan archivos de código –HTML, JavaScript, Java, CSS, etc- para lograr su cometido. Por otra parte los autores interactúan con ellos mediante un diálogo el cual es definido para cada componente –puede ser de edición o diseño-. Mediante el dialogo los autores son capaces de crear el contenido que luego es desplegado. Dos de las principales características del uso de los componentes es su reutilización y fácil modificación.

Por último los autores utilizan el ambiente de autor brindado por AEM para interactuar con los componentes. Este ambiente cuenta con tres diferentes modos -edición, diseño y pre visualización-.

Capítulo 5: Implementación del caso de estudio en AEM

En el presente capítulo explicaremos la implementación del caso de estudio en AEM, detallando la creación del *template* de página, del componente *Topnav y Speaker Box* - dado que los mismos cubren gran parte de las técnicas utilizadas para el desarrollo en dicha plataforma-. Para acceder a la totalidad de los componentes restantes, el lector podrá utilizar el *Anexo 3 Solución completa del caso de estudio en AEM***.**

5.1 Estructura de proyecto

La estructura general del proyecto se basa en el arquetipo descripto en el capítulo anterior. Tal como nos señala la Figura 36, el nombre del proyecto es *Summit* -el cual viene ya inicializado con dos carpetas de componentes diferenciando los que son de estructura y de contenido-. El arquetipo crea dos *templates* pre-definidos, denominados *page-content* y *page-home*. Ambos poseen como componente de página al componente *page* que se encuentra en el proyecto bajo la ruta */apps/summit/components/structure/page*, el mismo tiene como súper tipo al componente *page* definido por AEM en la carpeta *libs*.

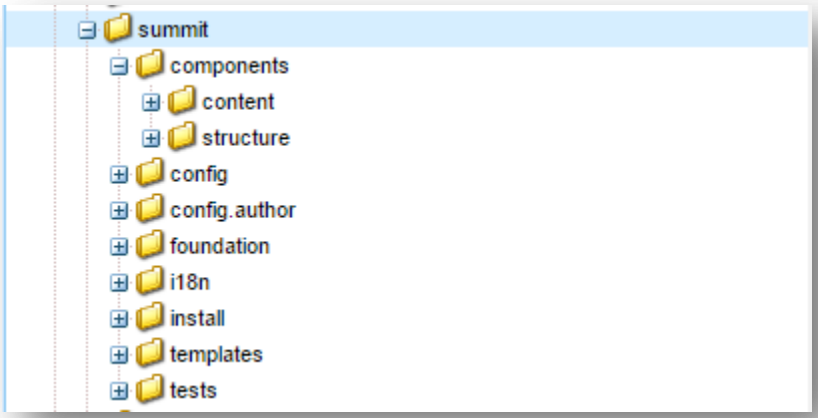


Figura 36 – Captura de pantalla de CRX mostrando estructura del proyecto. Fuente: Autoría Propia.

El arquetipo define también una librería global cuya categoría es *summit.all* y cuyo nodo se encuentra en la carpeta asociada al proyecto bajo */etc,* tal como muestra la Figura 37. También podemos observar que en dicha carpeta creamos los archivos globales de estilo y *scripts* del sitio web.

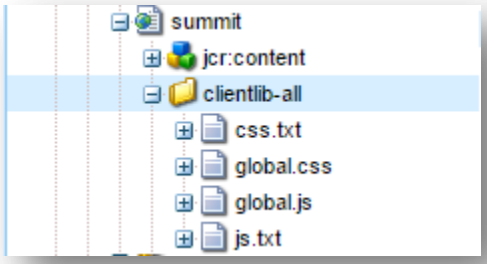


Figura 37 – Captura de pantalla mostrando la librería global del proyecto. Fuente: Autoría Propia.

5.2 Template de página

Si bien no es necesario modificar los nodos *template* definidos por defecto en el proyecto *Summit,* sí lo será a la hora de modificar el archivo HTML del componente de página asociado a los mismos. La propuesta consiste en sobrescribir el archivo encargado del renderizado de la página -*page.html*-, ajustándolo a nuestras especificaciones pero manteniendo las funcionalidades definidas por el componente padre de AEM. Al mismo le agregaremos tres *parsys*: uno llamado *content* -el cual será utilizado por los autores para incluir componentes de contenido a la página-, otro llamado *header* -donde se incluirá estáticamente al componente *Topnav-,* y por último, haremos lo análogo para el *Footer*.

Existen dos importaciones a la librería *Summit.all*: en el *header* del documento podremos importar los estilos mientras que en el *body* importaremos los *scripts*. La sintaxis de *Sighlty* permite diferenciar si las importaciones son de archivos de estilo o archivos de *scripts*. Con la implementación descrita nos aseguramos que todas las páginas creadas a partir de los *templates* de nuestro proyecto tengan incorporado al componente *Topnav* y *Footer*, así como las importaciones de estilos y *scripts* necesarios.

5.3 Componentes estructurales

5.3.1 Topnav

El componente *Topnav* es el encargado de manejar la navegación del sitio web. Para facilitar la utilización del componente por parte de los autores -ilustrado en la Figura 38- se decidió otorgarle la capacidad de cargar de forma automática las páginas de primer nivel creadas bajo la página raíz del sitio web. Dos propiedades incidirán en el comportamiento del componente: el título asociado a cada página (utilizado para renderizar el texto del link en la navegación del componente) y la propiedad *hide in navigation* -la cual por defecto es falsa, pero en caso de ser verdadera la página no será mostrada en el navegador-.
Topnav también debe brindar la funcionalidad de *log in* y *log out* de los usuarios. El componente será incluido en la categoría *hidden* con el fin de mantenerlo oculto, evitando que sea incluido de forma manual por los autores, ya que el mismo es incluido de forma estática en todas las páginas del sitio web.

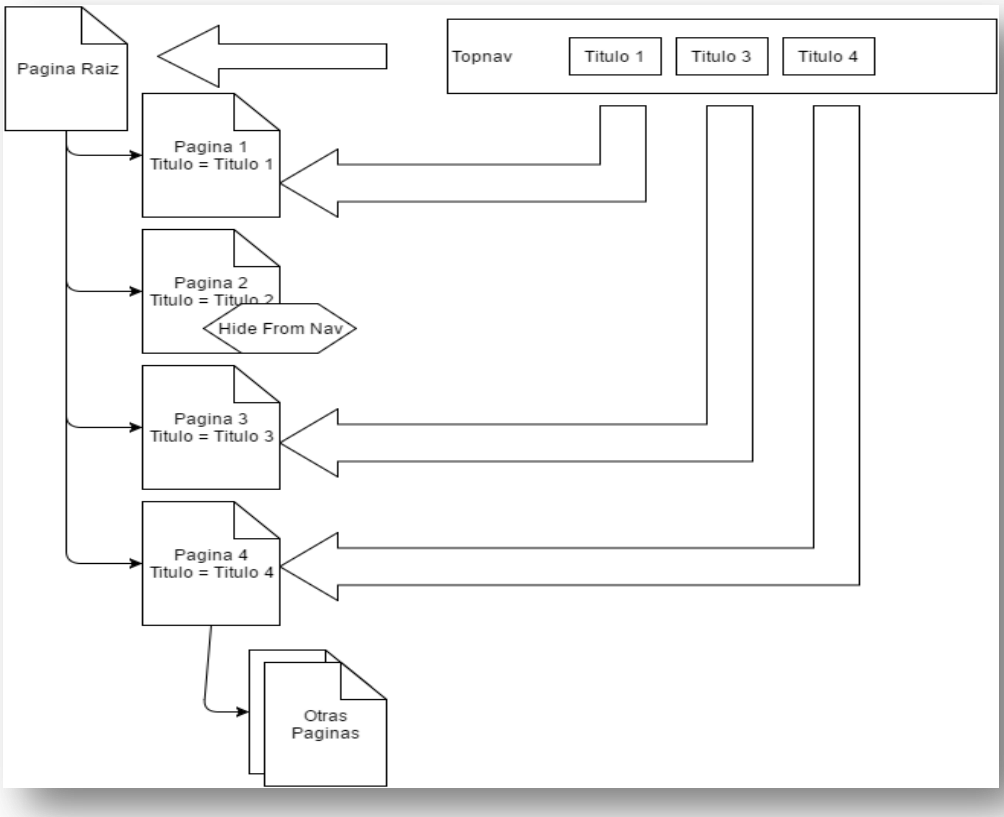


Figura 38 – Esquema del funcionamiento de los link del Topnav. Fuente: Autoría Propia.

5.3.2 *Diálogo del Topnav*

El componente incluirá un diálogo de diseño, esto permite que la configuración definida por los autores sea compartida por todas las instancias del mismo, evitando así que se deba configurar individualmente.

El diálogo está compuesto, a su vez, por ocho *widgets* que definen las siguientes propiedades:

- ***Root Page***

  Definida como un *widget* tipo *path.* La propiedad define la ruta donde se encuentra la raíz de las páginas del sitio web y a partir de ella el componente será capaz de cargar automáticamente las páginas hijas de primer nivel.

- ***Registration Path***

  Definida como un *widget* tipo *path*. La propiedad define la ruta de la página a la cual se debe redireccionar a un usuario del sitio web que busca darse de alta.

- ***Logo Image***

  Definida como un *widget* tipo *image.* La propiedad define la imagen del logo del *Topnav.*

- **Textos asociados a la parte visual**

  Contiene cinco *widgets* tipo texto, los cuales son utilizados para la configuración de los diferentes textos que son visibles dentro del *log in box.*

Para facilitar la utilización del diálogo por el autor, se divide a los *widgets* en dos conjuntos diferentes. Tal como se señala en la Figura 39 cada conjunto es agrupado en un *tab* conteniendo la definición de las rutas, del logo y de los textos respectivamente.

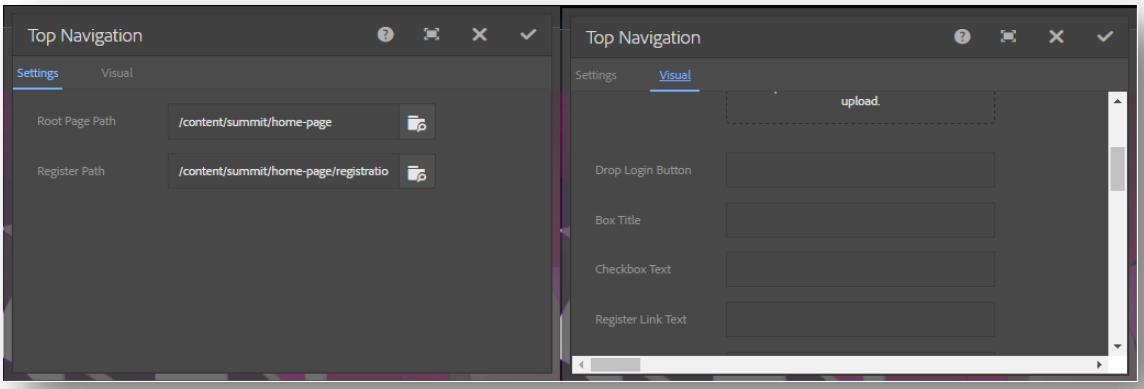


Figura 39 – Captura de pantalla con el dialogo desarrollado para el componente Topnav. Fuente: Autoría Propia.

5.3.3 Backend del Topnav

El componente utilizará una clase de JAVA -creada por nosotros- llamada *MegaMenu*, la cual extiende de la clase *WCMUsePojo*. Cuando el componente es renderizado la vista *topnav.html* se comunica con la clase *MegaMenu* utilizando *Sightly.* Mediante esta comunicación *MegaMenu* devuelve a *topnav.html* los datos que deben ser mostrados en el menú.
MegaMenu utiliza la propiedad *Root Page* definida por el autor y la clase *PageManager* que brinda la API de AEM para obtener todas las páginas hijas de primer nivel de la página raíz. A su vez, las páginas son representadas por la clase *Page* definida por la API de AEM y el conjunto obtenido de las mismas es filtrado por la propiedad *hide in navigation.* Dado que la clase *Page* contiene gran volumen de información que resulta irrelevante para generar el navegador, se mapean los atributos nombre y ruta de las páginas en una clase llamada *MenuItem* -la cual creamos y utilizamos para enviar una lista de la misma con los datos obtenidos al archivo *topnav.html*-.
Las decisiones tomadas durante la implementación del componente *Topnav* se enmarcaron dentro del objetivo de maximizar la reutilización de componentes y propiedades ya brindadas por AEM (Ver Figura 40).

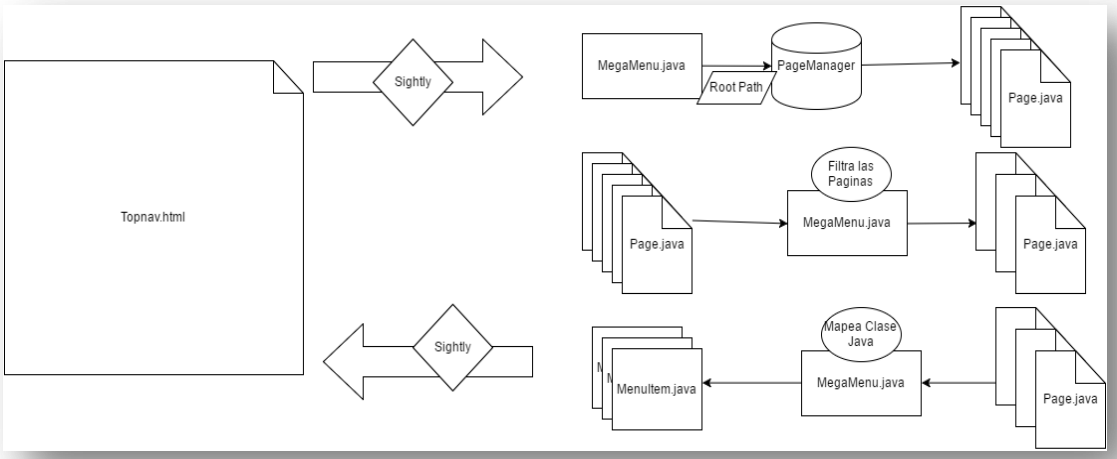


Figura 40 – Esquema que representa el funcionamiento del Topnav en lo que respecta a los links de navegación. Fuente: Autoría Propia.

5.3.4 Log in del Topnav

El componente brinda la posibilidad a los usuarios del sitio web de poder identificarse. Como vemos en la Figura 41 el *log in box* permite al usuario ingresar su nombre de usuario y contraseña. Al hacer clic en el botón de *Log In*, el componente realiza un pedido a través de AJAX utilizando *Sightly* a la clase *UserUtils* –extiende de *WCMUsePojo -,* que creamos específicamente para el manejo de los usuarios del sitio web. Dicha clase utiliza las funcionalidades que ofrece la API de AEM a través del *UserManager* para en este caso, chequear que el usuario y la contraseña recibidos coincidan. Como mencionamos anteriormente, todos los textos del *log in box* son configurables por los autores.

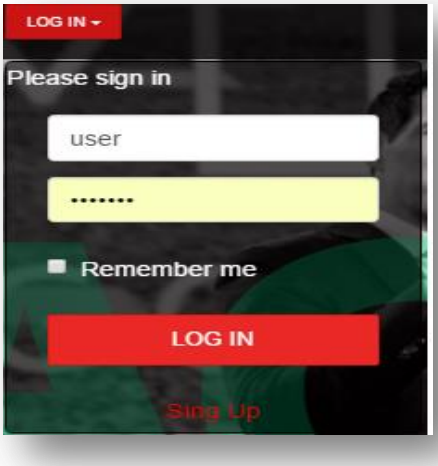


Figura 41 – Captura de pantalla que muestra el log in box del Topnav. Fuente: Autoría Propia.

5.4 Componente de contenido

La categoría de todos los componentes definidos a continuación es *Summit Content.*

5.4.1 Speaker

La creación del componente *Speaker* surge a partir de una decisión de diseño, que fue seleccionada entre dos opciones. Los *speakers* definidos en el caso de estudio son utilizados por dos componentes, *Speaker Box* y *Calendar*. Ahora bien, surge la disyuntiva respecto a la forma en la cual los autores crearán a los *speakers*. En una primeria instancia, se podría pensar que los mismos deben ser creados utilizando los diálogos propios para cada componente, mediante una lista donde cada elemento contenga una imagen y dos *widgets* de texto que definen su foto, nombre y profesión respectivamente–como vemos en la Figura 42-. Esta implementación es similar a la creación de los *links* en el componente *Footer –ver Anexo 3 Solución completa del caso de estudio en AEM-*, sin embargo, se obtendría como resultado datos duplicados en los casos que ambos componentes - *Speaker Box* y *Calendar* - requieran la utilización de un mismo *speaker*. A su vez, introduce el riesgo de generar incoherencia entre los datos, en caso de que por error un mismo *speaker* sea definido de forma diferente entre los componentes.

Figura 42 – Captura de pantalla mostrando el diálogo del componente Speaker. Fuente: Autoría Propia.

El componente *speaker* surge como una alternativa para esta solución y con el objetivo de que sea utilizado por los componentes *Speaker Box* y *Calendar.* Mediante esta decisión de diseño nos aseguramos que los autores creen al *speaker* -representado por un componente- una única vez y de forma centralizada. Para ello, los autores deberán crear una página con la propiedad *hide in navigation* verdadera. Dicha página va a contener las instancias del componente *Speaker* y permanecerá oculta para los usuarios del sitio web.

El componente está compuesto por un diálogo simple que incluye tres *widgets*, uno de tipo imagen para la foto del *speaker*, y los siguientes dos de tipo texto para su nombre y profesión respectivamente.

El componente utiliza una librería de estilos y *scripts* propios cuya categoría es *summit.speaker.*

5.4.2 *Speaker Box*

Speaker Box está compuesto por cuatro instancias del componente pre definido por AEM llamado *Paragraph Reference*, quien permite referenciar a una instancia de otro componente. Mediante los mismos, los autores podrán referenciar las instancias del componente *Speaker* antes mencionado a través de su ruta específica.
Los cuatro componentes *Paragraph Reference* incluidos en *Speaker Box* tienen un diálogo simple que contiene un *widget* de tipo *path.* Éste permite definir la ruta de la instancia de un componente al cual se precise referenciar. A su vez, el propio componente *Speaker Box* provee de un diálogo permitiendo definir el color de fondo con un *widget* de tipo *color picker*.
Por último y para facilitar la edición del componente *Speaker Box*, a la hora de agregar o quitar referencias de los diferentes *Paragraph Reference,* el renderizado se verá de forma diferente dependiendo si se está en modo edición -Figura 43- (en un ambiente de autor) o está siendo visto por un usuario del sitio web. Esto es posible gracias a las funcionalidades que brinda *Sightly*.

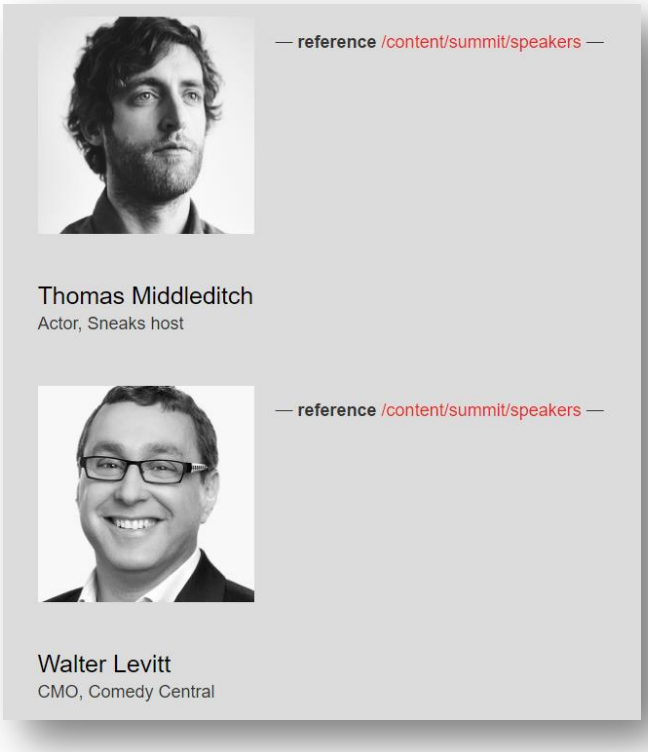


Figura 43 – Captura de pantalla que muestra como son desplegados los *speakers* en el ambiente de autor. Fuente: Autoría propia.

5.5 Manejo global de Librerías

Los componentes *Calendar*, *Speaker*, *User Register* y *Jumbo* definen librerías propias con una categoría específica para cada una de ellas. Dichas categorías son agregadas como dependencias de la categoría global *summit.all.* De esta forma, se realiza una importación de estilos y *scripts* a una única categoría que engloba todas.
Otro enfoque posible sería incluir dentro de *summit.all* a las librerías particulares de los componentes. Si bien su funcionamiento hubiera sido el mismo, creemos que la definición de librerías separadas aumenta la independencia de los componentes y su reutilización en otros proyectos.

5.6 Solución Móvil

AEM tiene integrado PhoneGap [36], un *framework* creado por Adobe utilizado para desarrollar aplicaciones móviles. El desarrollador no necesita tener conocimientos de lenguaje de programación móvil, sino de lenguajes de desarrollo web como HTML, CSS y JavaScript. A partir de estos PhoneGap produce aplicaciones para todas las plataformas de sistemas operativos móviles más populares, como iOS, Android y Windows Mobile OS. El *framework* explota el concepto de Single Page Apps, el cual consiste como sugiere su nombre en crear aplicaciones de una única página web la cual simula una aplicación móvil -ver Figura 44-. De esta forma y mediante JavaScript no perderá los datos que residen en la memoria y se podrá administrar la transición del contenido de un estado visual a otro. AEM ofrece un arquetipo de proyecto móvil el cual utilizamos para crear nuestra aplicación.

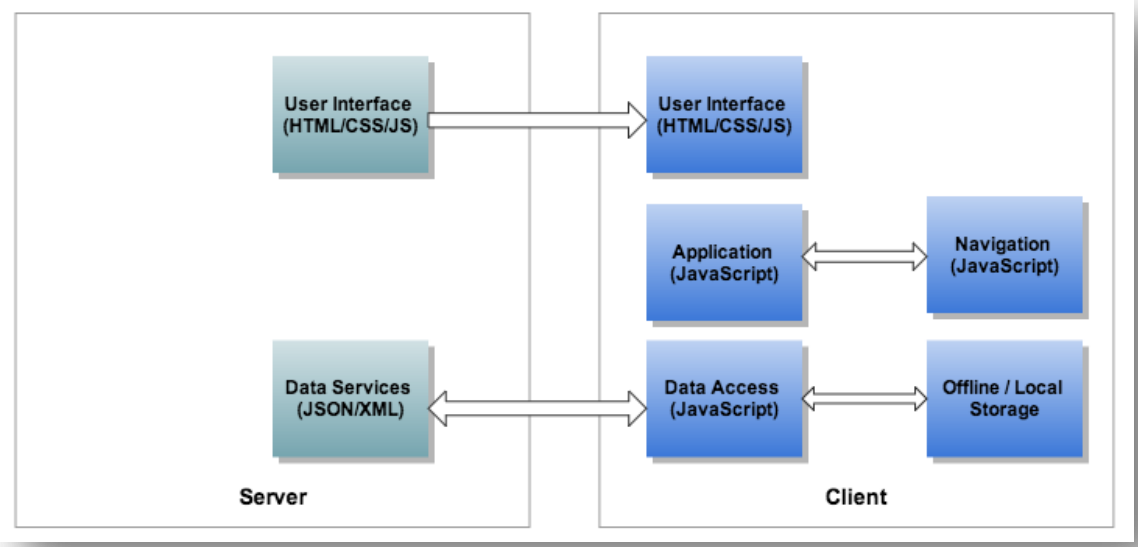


Figura 44 – Esquema de funcionamiento de una aplicación utilizando Phonegap. Fuente: [36].

5.6.1 Servicios

Utilizando la estructura de dicho proyecto tuvimos que crear cuatro servicios en el *bundle* del proyecto del sitio web que van a ser consumidos por nuestra aplicación móvil mediante AJAX:

5.6.1.1 Log In

Es el encargado de validar la combinación de usuario y contraseña ingresados por el usuario en la aplicación móvil.

5.6.1.2 Event List

Obtiene todos los eventos del calendario que comienzan en la fecha enviada en el pedido AJAX como parámetro.

5.6.1.3 Event Detail

A partir de la fecha y el identificador de un evento, ambos enviados en el pedido AJAX, el servicio devuelve los valores de las propiedades del mismo.

5.6.1.4 Event Reservations

El servicio se encarga de crear o eliminar una reserva echa por un usuario a un evento. Los parámetros enviados son: la fecha del evento, el identificador, el nombre de usuario y el tipo de operación –alta o baja- de reserva.

5.6.2 Componentes

Para poder cumplir con los requerimientos de la aplicación móvil creamos tres componentes de contenido –Login, EventList y EventDetail- los cuales interactúan con los servicios. Éstos son similares a los componentes del sitio web en cuanto a desarrollo y utilización por parte de los autores.

5.6.2.1 Login

El log in -Figura 45- de la aplicación móvil envía el usuario y contraseña mediante un servicio REST al servicio de log in antes mencionado, si el usuario y contraseña son correctos se redirecciona a la siguiente vista de la aplicación.

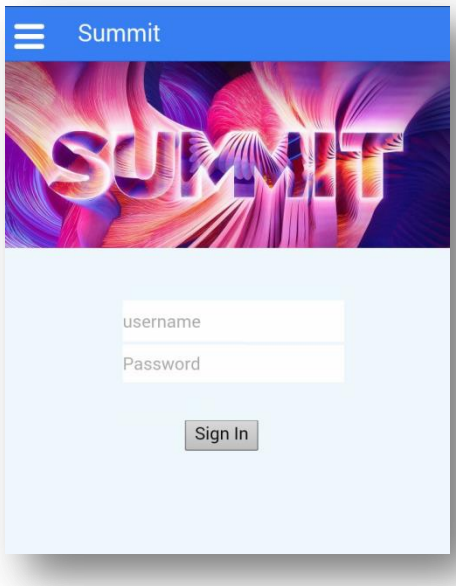


Figura 45 – Captura de pantalla de la aplicación móvil de AEM. Fuente: Autoría propia.

5.6.2.2 Event List

El componente *event list* es el encargado de consultar los eventos que existen en el sitio web desde la aplicación móvil. Para lograr este cometido tiene un campo donde el usuario de la aplicación móvil puede seleccionar una fecha y a partir de esta serán desplegados todos los eventos disponibles -Figura 46-. Por último el usuario podrá seleccionar cualquiera de los eventos desplegados para poder acceder a sus detalles.

5.6.2.3 Event Detail

Por último el componente *event detail* se encarga de desplegar los detalles del evento seleccionado por el usuario de la aplicación móvil -Figura 47-. A su vez le ofrece la posibilidad de realizar o eliminar una reserva del evento desplegado.

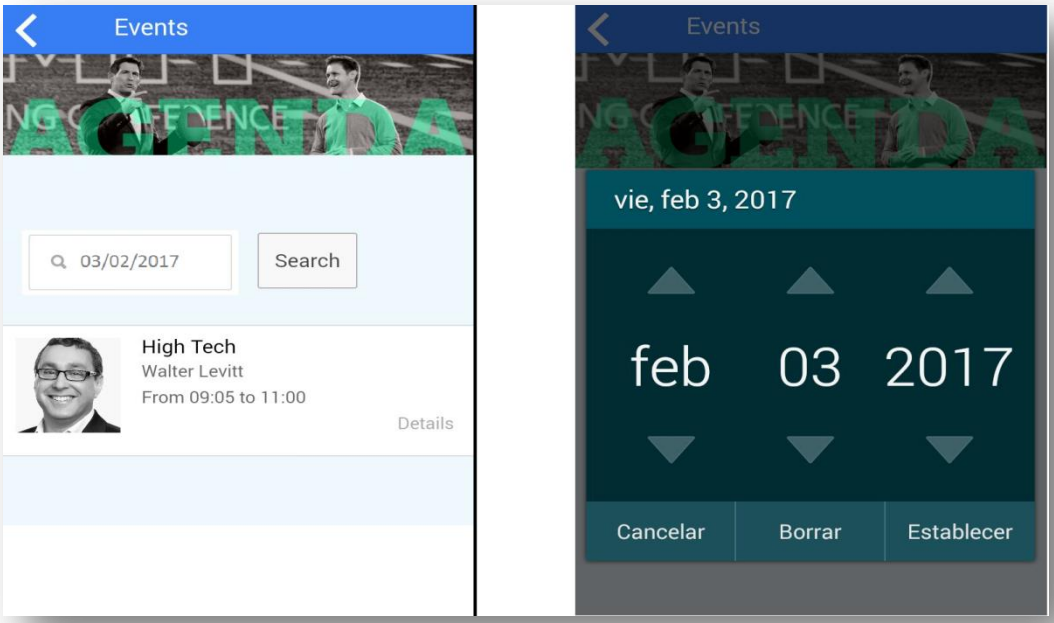


Figura 46 – Dos captura de pantallas diferentes de la aplicación móvil en AEM. Fuente: Autoría propia.

5.7 Creación de contenido

Por último y habiendo implementado los componentes definidos anteriormente, pasamos a la creación de las páginas del caso de estudio. La solución está constituida por seis páginas. En primer lugar, el nodo referente a nuestro sitio –*summit Site-*, con dos páginas hijas: *Home Page y Speakers.* La primera será utilizada como página raíz definida en el componente *Topnav* y en la segunda se incluyen las instancias del componente *Speaker* utilizadas por los componentes *Calendar* y *Speaker Box.* Esta última permanecerá oculta para la navegación y el componente *Topnav.*

Por último se crean las páginas *Adobe Attend*, Agenda y *Registration* como hijas de *Home Page*, donde la página *Registration* se le define la propiedad *Hide in Navigation* en verdadero. Se ilustra la jerarquía de páginas creada en la Figura 48.

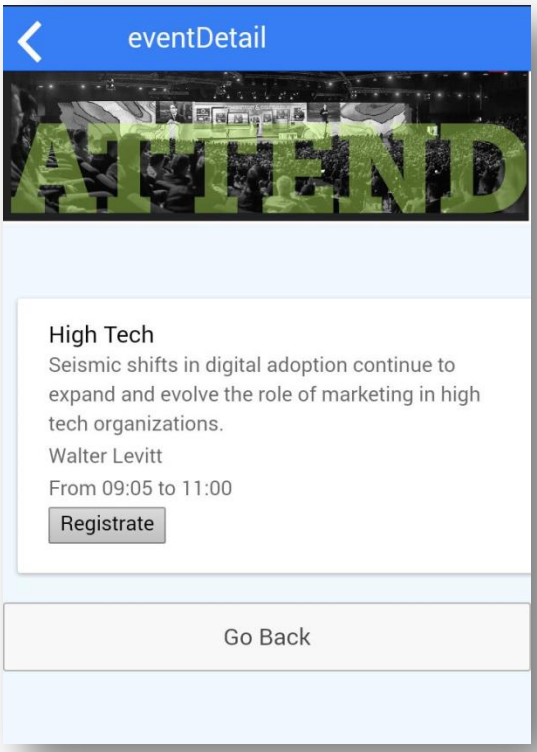


Figura 47 - Captura de pantallas diferentes de la aplicación móvil en AEM. Fuente: Autoría propia.

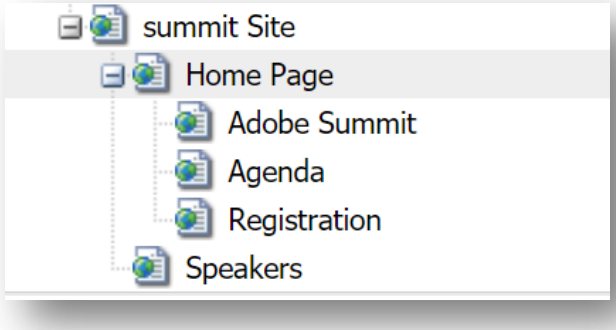


Figura 48 – Captura de pantalla de la jerarquía de páginas creada para el caso de estudio en AEM. Fuente: Autoría propia.

5.8 Resumen del capitulo

Fue posible utilizando la plataforma AEM poder implementar el caso de estudio definido en el capítulo tres. Se logró crear una estructura compleja de componentes reutilizables y fácilmente configurables en distintas secciones del sitio web. Adicionalmente utilizando el framework Phonegap, descubrimos que AEM tiene mucho para ofrecer en cuanto a aplicaciones móviles se refiere, con una excelente integración de tecnologías web y aplicaciones móviles.

Capítulo 6: Introducción a Sitecore

El presente capítulo es un resumen del *Anexo 4 Documentación técnica de Sitecore*.

El objetivo del mismo es presentar una introducción técnica, la cual debe ser suficiente para comprender la implementación del caso de estudio en Sitecore del próximo capítulo.

6.1 Arquitectura

Al trabajar con esta plataforma nos obstaculizó el hecho de disponer de muy poca información sobre su arquitectura. Esto pone de manifiesto la dificultad de cara al aprendizaje de esta plataforma, desconocida hasta el inicio de este proyecto.

6.1.1 Desplegado de contenido

Los sistemas encargados de desplegar o enviar contenido a los usuarios web son:

- ***Rendering Engine***
  Es el encargado de la ejecución dinámica de los archivos HTML o CSHTML que son consumidos por los usuarios a través del navegador, entre otras tareas.
  Realiza una jerarquía de controles de presentación y luego determina si los ejecuta o los recupera de una salida previamente guardada en *caché* bajo las mismas condiciones de procesamiento.

- ***Item Web API***
  Es un módulo cuyo propósito es el de proveer a los desarrolladores el acceso programático a la instancia de Sitecore, pudiendo manipular datos a través de código por medio de pedidos HTTP –servicios *RestFULL*-.

Los visitantes de los sitios publicados acceden a páginas renderizadas, pudiendo invocar al *Item Web API.*
Todas estas técnicas para acceder a los datos gestionados en Sitecore resuelven las llamadas de la *API* contra la capa de abstracción de contenido de Sitecore.

6.1.2 Capa de Persistencia

Cada instancia de Sitecore depende de cierta cantidad de bases de datos relacionales *Microsoft SQL Server* u *Oracle*. Como el código desarrollado es a través de una capa que abstrae el mecanismo de almacenamiento subyacente (a través de una API de Sitecore), se pueden utilizar diferentes bases de datos en distintos ambientes (desarrollo, prueba), a menos que se interactúe directamente con la base de datos.
Sitecore crea por defecto tres bases de datos que cumplen los siguientes propósitos:

- ***Master***

Contiene la totalidad del contenido -incluyendo los cambios que aún no han sido publicados en el sitio en línea-. En esta base de datos trabajan directamente los desarrolladores y autores antes de publicar cualquier modificación al sitio web en producción.

- ***Web***

La base de datos web contiene todo el contenido del sitio web ya publicado y que puede ser accedido por los usuarios.

- ***Core***

  En esta base de datos se mantienen todo lo que no refiere al contenido, sino a la configuración y funcionamiento de la propia plataforma.

6.1.3 Estructura árbol de contenido en Sitecore

En la siguiente Figura 49 se muestra el árbol de contenido en Sitecore, al cual se accede a través del editor de contenido:

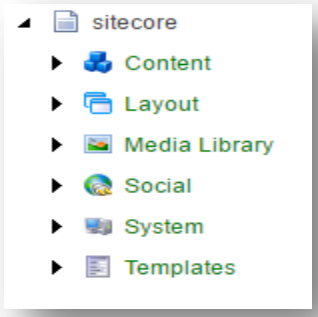


Figura 49 – Captura de pantalla recortada de la interfaz gráfica de Sitecore, muestra su estructura de carpetas. Fuente: Autoría Propia.

- ***Content***

  Se almacenan las páginas y la información de las mismas. La estructura de los ítems representa la estructura del sitio web.
- ***Layout***

  Almacena todo lo concerniente a la presentación de las páginas que componen el sitio web, en particular *layouts* y *renderings* –se definirán más adelante-.
- ***Media Library***

  Persisten todos los archivos multimedia.
- ***Social***

  Almacena ítems relacionados a aplicaciones que utilicen redes sociales.
- ***System***

  Persisten las herramientas estándar para el editor de contenido y la aplicación web, por ejemplo los idiomas disponibles.
- ***Templates***

  Esta carpeta es la encargada de almacenar todos los *data templates* disponibles, tanto los que se crean como los que ya vienen precargados con la instalación de Sitecore.

6.1.4 Tecnologías usadas por Sitecore

6.1.4.1 .Net

Framework que define un ambiente que soporta el desarrollo y la ejecución de aplicaciones altamente distribuidas y basadas en componentes de *software*. Permite a diferentes lenguajes trabajar juntos ofreciendo seguridad, portabilidad, y un modelo común de programación para la plataforma de Windows [37].

6.1.4.2 C#

Se trata de un lenguaje de programación de propósito general y orientado a objetos. Fue creado para trabajar con el *framework* .NET [38], por lo tanto los programas escritos en C# son automáticamente portables a cualquier ambiente .NET.

6.1.4.3 IIS

Sigla correspondiente a *Internet Information Services*. Servidor web para Windows [38].

6.1.4.4 Microsoft Visual Studio

Entorno de desarrollo integrado con todas las características para Android, iOS, Windows, web y la nube [39].

6.1.4.5 Microsoft SQL Server

Sistema de administración y análisis de bases de datos relacionales de Microsoft [40].

6.2 Conceptos de desarrollo en Sitecore

Con el fin de dar una introducción a los conceptos utilizados para la solución del caso de estudio, a continuación se definen los principales elementos que forman parte del desarrollo de un proyecto web en Sitecore.

6.2.1 Ítems

Los ítems de Sitecore representan recursos individuales del WCMS. Un ítem puede representar cualquier tipo de dato: páginas, secciones, metadata, etc.
Haciendo una analogía con el paradigma de programación orientada a objetos, un ítem equivaldría a una instancia de una clase (objeto).
Cada ítem existe dentro de una jerarquía de elementos en la base de datos de Sitecore. La ruta de un ítem identifica su ubicación dentro de la jerarquía y Sitecore asigna un identificador global único (GUID, o simplemente ID) para cada ítem.
Éstos no sólo contienen datos sino que además se comportan como subdirectorios capaces de contener otros ítems.
Todos los ítems presentan cinco propiedades en común:

- ***Name***

Nombre del ítem. Puede no ser único.

- ***Key***

Es generado automáticamente por Sitecore a partir del nombre. Corresponde al nombre asignado al ítem en letra minúscula. Puede no ser único.

- ***Path***

Se define mediante el nombre del ítem y sus ancestros en orden, separados por el caracter '/'. La ruta es construida dinámicamente y puede ser modificada si el ítem es cambiado de ubicación.

- ***ID***

Identificador único de un ítem.

- ***Data Template***

Esta propiedad indica el ID del *data template* a partir del cual fue creado el ítem (será explicado más adelante).

Sitecore organiza todo su contenido en forma de directorio, lo que permite identificar a los ítems de manera sencilla a través de su ruta. También se puede referir a un ítem a través de su ID, ya que el mismo es único para cada instancia de Sitecore.

6.2.2 Data template

Los *Data Templates* (a partir de ahora *templates*) definen la estructura de los ítems. Nuevamente es posible realizar un paralelismo con el paradigma orientado a objetos, donde un *template* ocuparía el lugar de una clase en la cual se definen los atributos. A su vez, los ítems constituirían las instancias de dichas clases donde se definen los valores de los atributos.

Dentro de los *templates* es posible definir cero o más secciones, las cuales son utilizadas para agrupar diferentes campos del *template*. Entre otras, su función es la de proveer mayor facilidad de configuración para los autores, agrupando campos que cumplen una función en conjunto. Podemos ilustrar lo descripto anteriormente en la Figura 50, donde los campos son agrupados en las secciones y a cada uno se le definirá al menos un nombre. De esta manera será identificado por los autores.

Sitecore dispone de una gran variedad de tipos de datos que pueden ser utilizados para definir diferentes campos. Adicionalmente, es posible que los desarrolladores definan tipos de datos específicos. A su vez, observando nuevamente las figuras Figura 50 y Figura 51, podemos corroborar que cada ítem puede asignar valores para los campos definidos en el *template* donde fue creado.

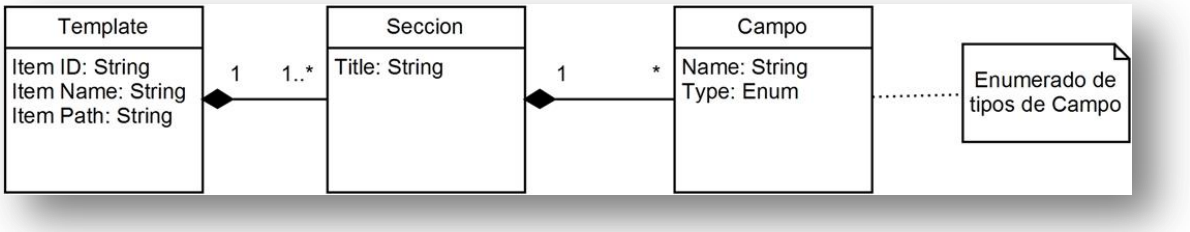


Figura 50 – Modelado de conceptos de *template*, sección, campo e ítem de Sitecore. Fuente: Autoría Propia.

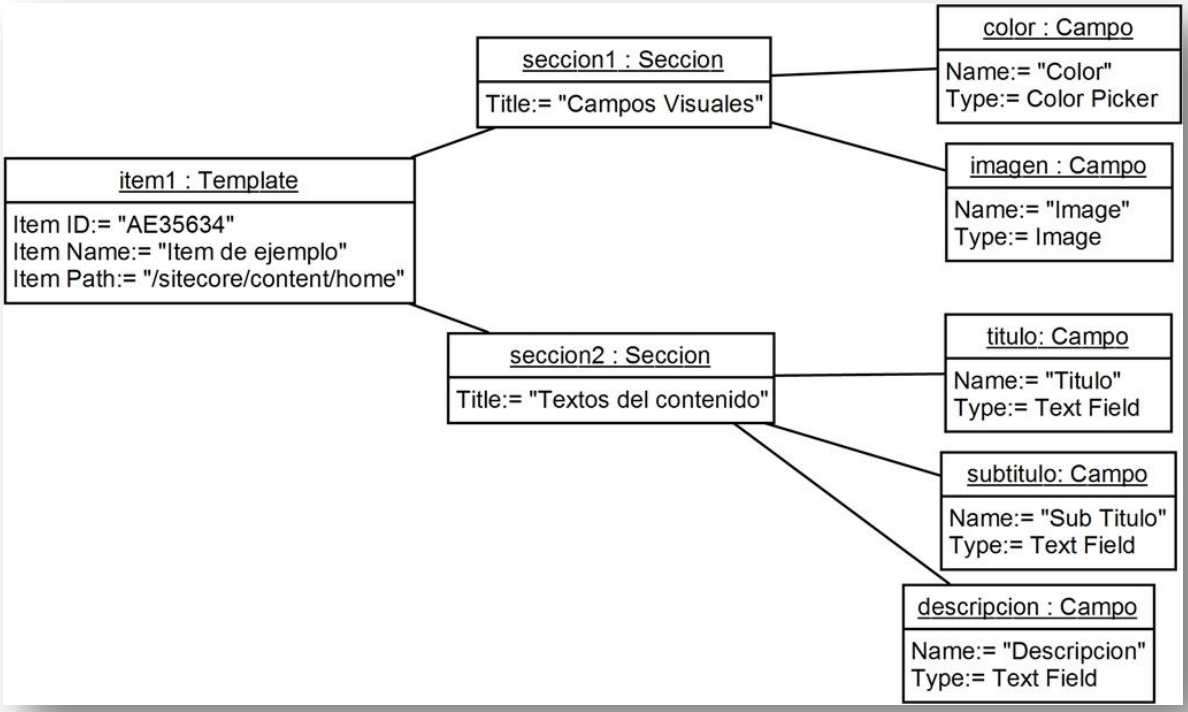


Figura 51 – Diagrama de instancias de conceptos de Sitecore. Fuente: Autoría Propia.

6.2.2.1 Standard values

Los *templates* ofrecen la posibilidad de definir valores por defecto mediante los *standard values*. Esto es posible no sólo para los campos sino también para la capa de presentación – se verá más adelante-. Recurriendo nuevamente al paralelismo con el paradigma de programación orientada a objetos, los *standard values* corresponderían a los valores asignados en la creación por defecto de una instancia.

De esta forma se evita tener que definir manualmente la configuración para cada ítem, sin embargo, los autores tienen la posibilidad de una vez creado el ítem modificar estos valores a otros que prefiera.

Como observamos en la Figura 52, los *standard values* son ítems persistidos como hijos del *template* al cual pertenecen y son creados de forma automática con el nombre __*Standard Values*. Dentro de este ítem el desarrollador puede definir tanto los valores por defecto para cada campo definido en el *template*, como cualquier otro tipo de configuración la cual será replicada a cada ítem que se cree a partir del mismo.

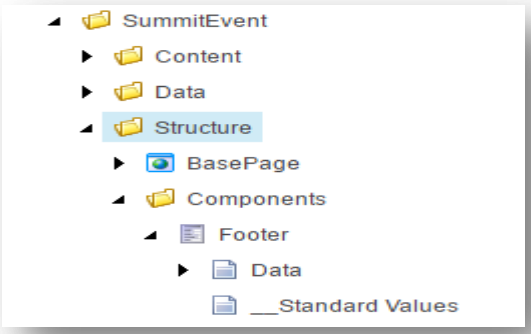


Figura 52 – Captura de pantalla mostrando la estructura de un *template* con su standar values. Fuente: Autoría Propia.

6.2.2.2 Herencia entre Templates

Volviendo a utilizar el paradigma de programación orientado a objetos, los *data templates* de Sitecore tienen la capacidad de heredar las características de múltiples *data templates*. Esto incluye las secciones, campos y s*tandard values*, entre otros.

6.2.3 Layouts Details

Es posible configurarle a cada ítem de Sitecore diferentes aspectos que refieren a su presentación visual, los cuales son denominados y agrupados como *Layout Details.* De esta manera, el desarrollador es capaz de definir la forma en que Sitecore renderiza a los ítems.

Como vemos en la Figura 53, el *Layout Details* de un ítem permite al desarrollador definir diferentes tipos de presentaciones a partir de ítems *Layouts* -que veremos a continuación-, diferenciándolos según los dispositivos desde los cuales se accede.

6.2.4 Layouts

Sitecore provee un *template* llamado *Layout* que permite crear ítems para definir una presentación visual. Estos ítems (a partir de ahora *layouts*) son utilizados para definir el renderizado. Esto es posible a partir de la definición de su campo *path* que especifica la ruta para relacionarlo con un archivo –HTML, CSHTML, ASPX, etc.-. De esta forma el *layout* puede ser utilizado por diferentes ítems asociándolo a través de su configuración de *Layout Details* (ver Figura 54 y Figura 55).

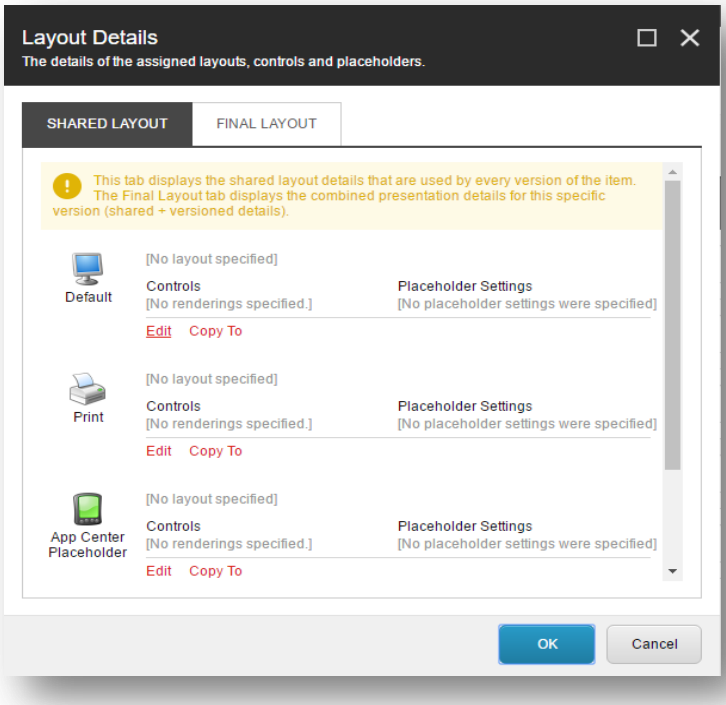


Figura 53 – Captura de pantalla mostrando la interfaz gráfica que ofrece Sitecore para configurar los Layout Detail. Fuente: Autoría Propia.

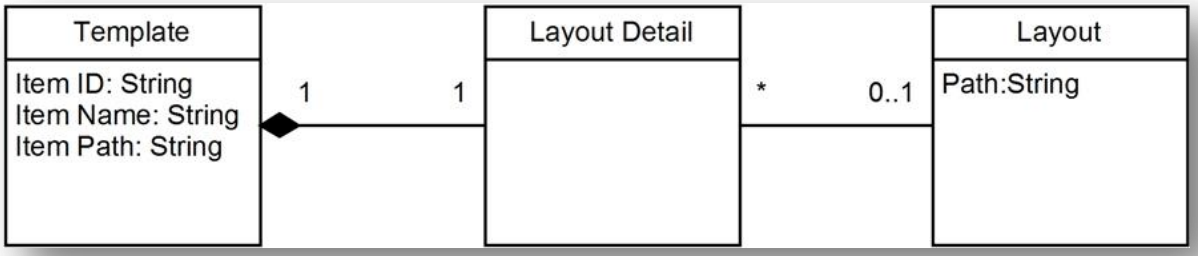


Figura 54 – Esquema de conceptos de la capa de presentación Layout Detail – Layout. Fuente: Autoría Propia.

Para facilitar el uso de nuestra solución queremos asegurarnos que todos los ítems creados a partir de un *template* específico sean inicializados con un *layout* determinado. Esto será posible asociando el *layout* a los *standard values* del *template* en cuestión.

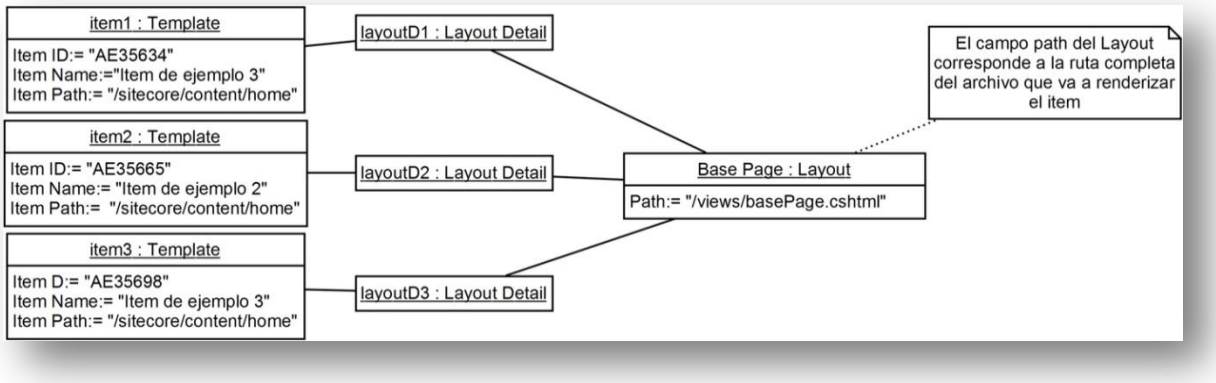


Figura 55 – Diagrama de instancias de conceptos de la capa de presentación *Layout Detail* de Sitecore. Fuente: Autoría Propia.

6.2.5 Renderings

Sitecore brinda un conjunto de *templates* que llamaremos *renderings*. Los mismos son utilizados para construir ítems que representan componentes de presentación individual, cuya función es construir bloques para sitios web. Algunos de sus posibles usos son los de:

- Renderizar contenido en una página, el típico caso de un componente de contenido en un sistema WCMS.
- Obtener datos de sistemas externos.
- Ejecutar lógica de *back end* con componentes no visuales, como solicitudes de registro para analítica web u otros propósitos.

Utilizaremos dos de estos *templates* recién mencionados. Los mismos se adaptan a nuestras necesidades y se basan en el patrón MVC: *View Rendering* y *Controller Rendering*.

6.2.5.1 View rendering

Se utiliza en los casos donde el desarrollo involucrado no contiene operaciones lógicas complejas ni procesamiento de información. Para este tipo de rendering no es necesario utilizar un controlador, por lo que los únicos componentes del patrón MVC usados son la vista y el modelo –opcional-.

El campo fundamental para la utilización de este tipo de *rendering* es denominado *path* -en el cual el desarrollador debe ingresar la ruta relativa a un archivo-. Ver figuras Figura 56 y Figura 57.

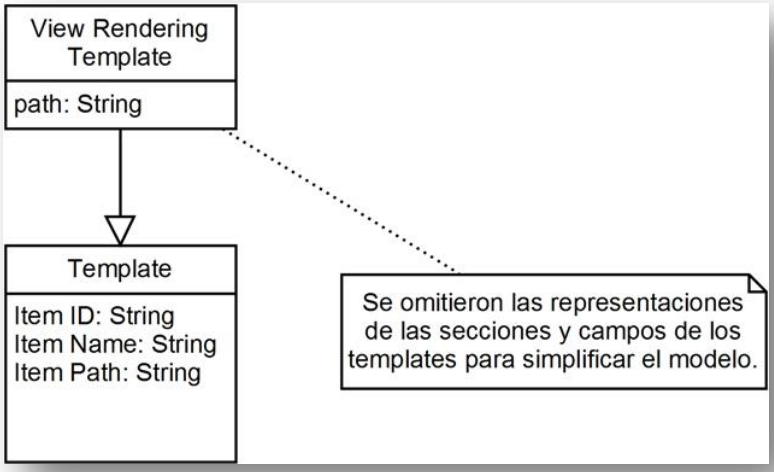


Figura 56 – Esquema de conceptos de *View Rendering*. Fuente: Autoría Propia.

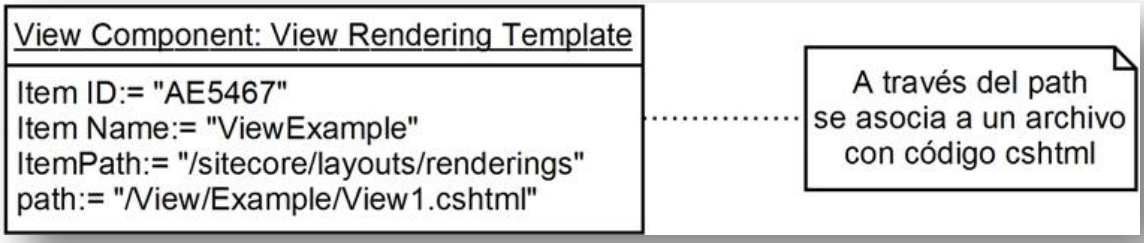


Figura 57 – Esquema de instancias de conceptos de *View Rendering*. Fuente: Autoría Propia.

Además, es posible asociar un modelo específico al *View Rendering* -el cual es enviado automáticamente a la vista -. Con este fin, Sitecore provee un *template* llamado *Model* a partir del cual debemos crear un ítem. El mismo consta de un único campo llamado *Model Type* utilizado para referenciar una clase C# (la que será ejecutada como modelo). La clase debe implementar la interfaz *IRenderingModel* definida por la API de Sitecore la cual cuenta con un método de inicialización que es ejecutado antes de ser enviado a la vista y que debe ser implementado para obtener los valores de los campos del *rendering*. Tal como se ilustra en las figuras Figura 58 y Figura 59, el ítem modelo antes mencionado debe ser asociado con el *View Rendering* a través del campo *Model* ingresando su ID o su ruta completa.

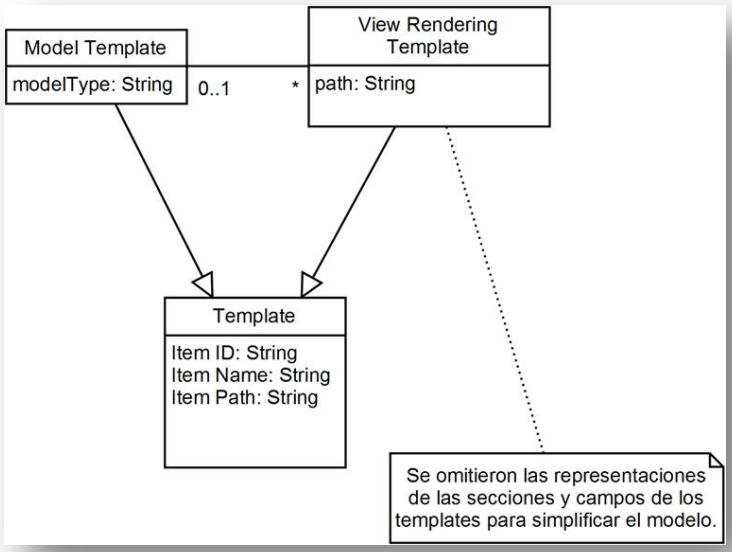


Figura 58 – Esquema de conceptos del *View Rendering* con modelo. Fuente: Autoría Propia.

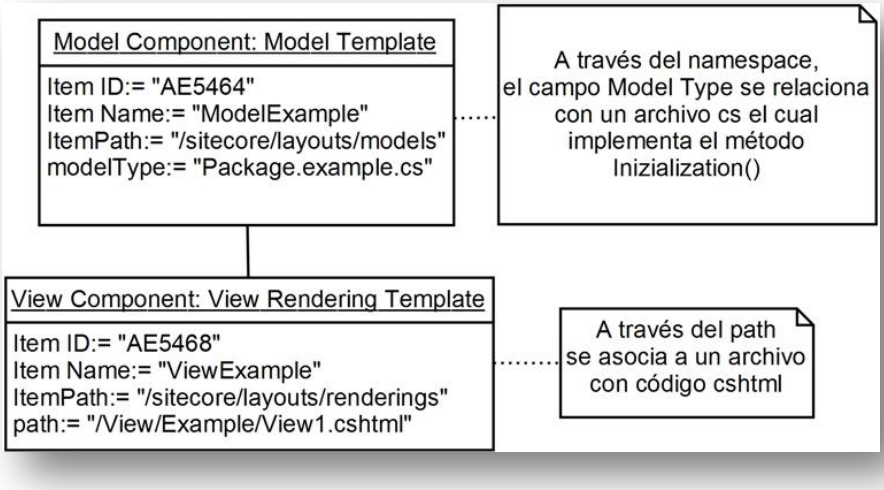


Figura 59 - Esquema de instancias de conceptos del *View Rendering* con Modelo. Fuente: Autoría Propia.

6.2.5.2 Controller rendering

Referencia a una clase utilizada como controlador - archivo de extensión *cs* - y a un método de la misma. Existen dos campos fundamentales para configurar el *Controller Rendering*:

Controller

Por lo general los nombres de los controladores en proyectos ASP.NET MVC *Framework* son de la forma *NombreControladorController*, pero para su referencia desde Sitecore es suficiente con ingresar *NombreControlador* en el campo *controller*.

Controller Action

El campo es utilizado para ingresar el método que será ejecutado cuando se renderice el *Controller Rendering*.

Es recomendada su utilización para desarrollos que requieran de operaciones lógicas, ya sean de mayor o menor complejidad, o de un mayor procesamiento de información.

A su vez, este tipo de *rendering* permite crear un modelo dentro de cada método del controlador y que sea enviado a la vista completando los tres componentes del patrón MVC. Asimismo, el desarrollador puede asociar varios *Controller Renderings* a un mismo controlador y asignarle a cada uno de ellos un método diferente (ver figuras Figura 60 y Figura 61).

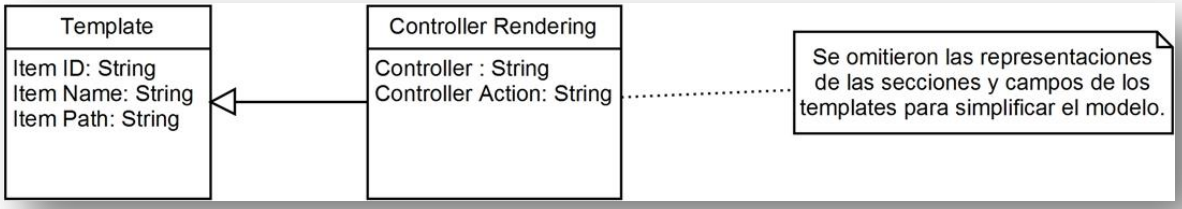


Figura 60 – Esquema de conceptos que participan en un *Controller Rendering*. Fuente: Autoría Propia.

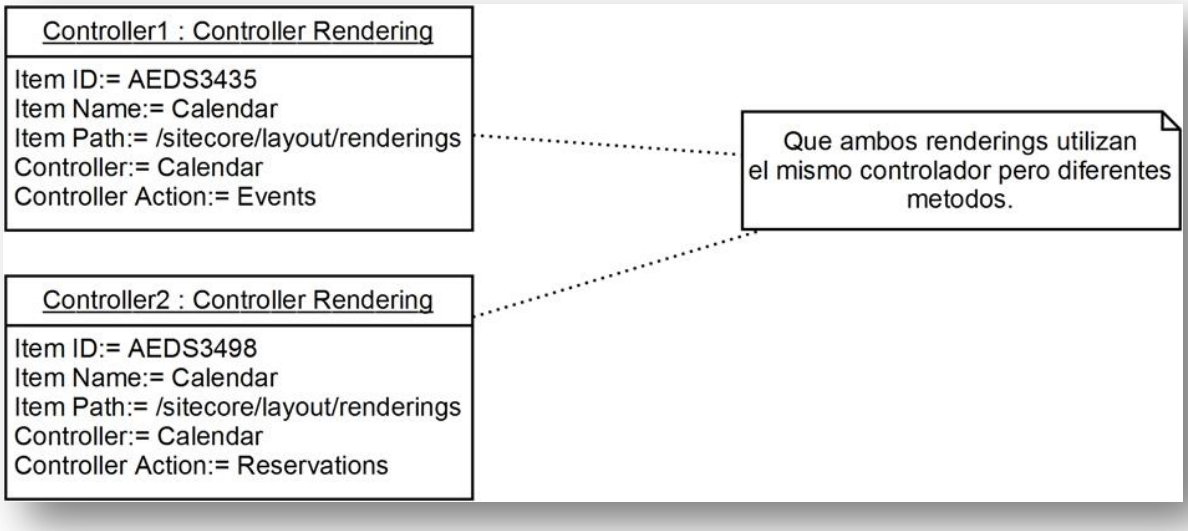


Figura 61 - Esquema de instancias de conceptos que participan en un *Controller Rendering*. Fuente: Autoría Propia.

6.2.6 Diferentes tipos de parametrización que tiene un rendering

En Sitecore existen dos maneras diferentes de parametrizar un rendering. Las mismas se detallan a continuación:

Asignación de un *Parameter Template*

Existen *templates* especiales encargados de definir los campos de configuración o contenido de un *rendering*. Los mismos deben extender del *template Standard Rendering Parameter* que ofrece Sitecore.

El desarrollador define los campos dentro del *template* para luego asociarlos al *rendering* en cuestión que busca parametrizar. Para asociarlos, se le debe asignar la ruta o ID específico del *template* en el campo *parameters template* del *rendering*.

De esta forma, los autores tendrán una ventana específica –similar a un diálogo en AEM- para cada instancia del *rendering*; mediante la misma podrán interactuar e ingresar valores a los campos definidos por el *Standard Rendering Parameter*. Estos valores son independientes de cada instancia del *rendering*.

Asignación de un ítem como fuente de datos.

En la propiedad *data source* del *rendering* se introduce la ruta o ID del ítem que oficiará de fuente de datos.

Dicho ítem puede ser creado a partir de cualquier tipo de *template*.

Desde el punto de vista del autor, al utilizar ítems como fuente datos, es posible realizar edición en línea, es decir, es posible editar ciertos valores sobre la vista propia del *rendering* en el ambiente de autor.

A su vez, esta forma de parametrización permite a los desarrolladores asociar un mismo ítem de fuente de datos a varias instancias del *rendering*, provocando que las mismas compartan los valores de los campos del ítem antes mencionado. Se desprende, por tanto, que esta opción planteada resulta de suma utilidad cuando se necesita desplegar el mismo contenido de un *rendering* en varios lugares del sitio web. Ahora bien, si el autor pretende que los valores de los campos se mantengan independientes en cada instancia del *rendering* deberá crear un ítem de fuente de datos para cada uno.

Por último, es importante destacar que las dos alternativas descriptas pueden combinarse en un mismo *rendering*.

6.2.7 Placeholder

Los *placeholder* –similar a los *parsys* de AEM- son controles de ASP.NET utilizados como contenedores de *renderings*-. De esta forma, los autores son capaces de agregar, mover o quitar dichos *renderings* dinámicamente.

Los *placeholders* son definidos dentro de los archivos HTML, CSHTML, ACXP de renderizado de los *layouts*; así, los desarrolladores son capaces de definir qué área de los archivos será utilizada por los autores para agregar componentes de contenido. Asimismo, los desarrolladores son los encargados de definir una *key* única para cada *placeholder*.

Adicionalmente, es posible definir un ítem a partir del *template Placeholder* definido por Sitecore para configurar el comportamiento del mismo, este consta de tres campos:

- **Placeholder Key:**
  El campo indica la *key* del *placeholder* asociado a la configuración que se está definiendo.
- **Allowed Controls:**
  Este campo define los *rendering* que son permitidos utilizar en el *placeholder*.
- **Description:**
  Dentro de este campo el desarrollador puede escribir información relevante para los autores respecto al uso del *placeholder*.

6.3 Contexto y APIs de Sitecore

6.3.1 Contexto

Define un contexto de procesamiento para cada pedido HTTP. Contiene información sobre el usuario, el ítem requerido, el dispositivo que realizó el pedido, el sitio administrado, además de otros datos asociados con el pedido HTTP correspondiente.
Se citan algunas de las propiedades estáticas más importantes que se pueden consultar a través del contexto de Sitecore:

- **Database**

Base de datos Sitecore asociada al pedido HTTP, o base de datos por defecto para el contexto del sitio.

- **Item**

Ítem solicitado a la base de datos de contexto por el cliente web (basado en la ruta de la URL solicitada).

- **User**

Usuario autentificado o anónimo en el contexto del dominio de seguridad.

6.3.2 API s

Sitecore provee de una completa gama de APIs que permiten realizar una gran cantidad de funcionalidades sobre la instancia. De acuerdo con nuestro objetivo, las operaciones más utilizadas fueron las de acceso, creación y eliminación de ítems. Se enumeran las principales APIs, usadas en este proyecto.

6.3.2.1 Sitecore.Data.Database

La clase Sitecore.Data.Database representa una base de datos en Sitecore, que puede ser la base de datos *Master* o *Core* o una base de datos de destino de publicación.
El acceso a las diferentes bases de datos deberá ser siempre través de esta API y no ejecutando directamente consultas SQL.
Adicionalmente, esta API provee de una función de suma utilidad, la que es responsable de recuperar un ítem: se trata del método *GetItem* y ofrece varias opciones de uso según los parámetros que se le indique a la función. En caso de invocar a la función solamente con un parámetro que corresponde a su ID o ruta, se recupera el ítem en su versión actual.

6.3.2.2 Sitecore.Data.Items.Item

La clase Sitecore.Data.Items.Item representa un ítem en una de las bases de datos de Sitecore.

Entre otras, ofrece las siguientes propiedades:

- **Children**
  Lista los hijos del ítem.
- **Database**
  Referencia a la base de datos que contiene el ítem.
- **Fields**
  Provee el acceso a los campos del ítem.
- **ID**
  Identificador del ítem dentro de la base de datos de Sitecore.
- **Key**
  Nombre del ítem en letra minúscula.
- **Parent**
  Padre del ítem, o retorna el resultado nulo si se trata del ítem raíz.
- **ParentID**
  ID del ítem padre.
- **Template**
  Template asociada al ítem.
- **TemplateID**
  ID del *template* asociado al ítem.
- **TemplateName**
  Nombre del *template* asociado al item.

6.4 Resumen de capítulo

El ítem es la unidad básica en Sitecore, a partir de ella se derivan todos los posibles elementos que se pueden crear y administrar en la instancia.

Para un desarrollo en Sitecore, es necesario el uso de d*ata templates*, *layouts*, *renderings*, *placeholders.* También son de mucha utilidad e importancia el contexto y APIs de Sitecore, en particular para el desarrollo de código.

Capítulo 7: Implementación del caso de estudio en Sitecore

Antes de describir el desarrollo de cada componente haremos una breve introducción a las tecnologías y ambiente de desarrollo utilizados. En este sentido, describiremos las principales decisiones tomadas al respecto, ya que a diferencia de AEM, la creación de un ambiente de desarrollo en Sitecore es una tarea más compleja.

Luego se detallarán los desarrollos correspondientes al *template BasePage*, *Topnav, Footer* y *Jumbo* ya que éstos cubren gran parte de las técnicas utilizadas en el desarrollo de la plataforma. Los componentes restantes se encuentran detallados en el *Anexo 5 Solución completa del caso de estudio en Sitecore.*

7.1 Ambiente de desarrollo

Si bien Sitecore ofrece determinadas recomendaciones carece de un arquetipo. Es por esto que la misma debió ser creada desde cero para la implementación del caso de estudio.

Las principales herramientas que utilizaremos para desarrollar el caso de estudio son:

- IDE *Microsoft Visual Studio* Community 2015 Versión 14.0.2 *Update* 2.
- *Microsoft .NET framework Versión 4.5.*

7.1.1.1 Logview4net Versión 16.8

Herramienta gratuita para el monitoreo y visualización de registros [41].
A través de esta herramienta, y configurando un *listener* de tipo UDP es posible monitorear en tiempo real el registro de Sitecore. Resulta de utilidad para la resolución de problemas, por ejemplo: colocar mensajes (en este caso de tipo de error o advertencia) entre líneas de código a evaluar.

7.1.1.2 Razor versión 3.2.3

Razor [42] es una sintaxis de marcado que permite incorporar código (Visual Basic y C #) al HTML el cual es ejecutado del lado del servidor en las páginas web. Está basado en ASP.NET y diseñado para la creación de aplicaciones web.

7.1.2 Frameworks

7.1.2.1 Glass.Mapper versión 4.2.0.184

Se trata de un *framework* que realiza un mapeo objeto-relacional (ORM en inglés). Permite el mapeo de datos a modelos [43], facilitando la tarea del desarrollador a la hora de obtener los valores de los campos de un ítem.

7.1.2.2 Sitecore Rocks versión 2.0.39.0

Framework que se integra a Visual Studio y provee una experiencia de desarrollo rápido y coordinada [44]. A través de esta herramienta se puede acceder a las tres bases de datos que proporciona Sitecore y por consiguiente, brinda acceso a todo el contenido de la instancia sin tener que utilizar un navegador web para acceder a la misma.

7.1.3 Ambiente de desarrollo integrado

7.1.3.1 Base de datos

Para la base de datos se decidió utilizar *Microsoft SQL Server* con el fin de trabajar con tecnologías del mismo proveedor, evitando así eventuales problemas de compatibilidad.

7.1.3.2 Repositorio

Bitbucket fue el repositorio utilizado para el proyecto. Dicha plataforma ofrece un repositorio privado (a diferencia de *Github* por ejemplo), el cual es gratuito para una cantidad limitada de desarrolladores. Resulta una opción adecuada para nuestro proyecto ya que lo conforma un equipo de dos personas únicamente.

7.1.3.3 Servidor web

Como servidor web se utilizó IIS versión 8.5, ya que es el servidor destinado por defecto para tecnologías Microsoft. Asimismo, es el requerido por Sitecore para su funcionamiento.

7.1.3.4 Trabajo colaborativo en Sitecore

La sincronización de la solución en un proyecto donde coexisten dos o más desarrolladores que cuentan con una instancia local de Sitecore se vuelve una tarea compleja. Esto se debe a que Sitecore no cuenta con una funcionalidad específica o herramienta propia para sincronizar proyectos en una misma instancia.

Por este motivo, debimos buscar herramientas externas a la plataforma para facilitar dicha tareas -la cual se reduce a sincronizar bases de datos relacionales-.

En primera instancia, nos encontramos con una herramienta llamada TDS (*Team Development for Sitecore*) [45] la cual ofrece no sólo la sincronización de proyectos sino además otras funcionalidades específicas. Sin embargo, la misma fue descartada ya que se trata de una herramienta propietaria y, por tanto, requiere de una licencia para su uso. Otra posible solución es que todos los desarrolladores pertenecientes al proyecto configuren las instancias locales para compartir una misma base de datos alojada en la nube. Esto último también fue descartado ya que no contamos con dicha infraestructura.

El resultado de la búsqueda se redujo entonces a tres herramientas *open source.* Las mismas proveen funcionalidades específicas o parciales para la sincronización de proyectos en Sitecore y son las siguientes:

7.1.3.4.1 Sitecore Ship

Sitecore Ship [46] permite instalar paquetes que contienen diferentes objetos de Sitecore tales como ítems, *templates* y *renderings* con sus valores correspondientes a través de un pedido HTTP. Esta funcionalidad resultaría más útil en caso de tener un servidor remoto únicamente para el proyecto, sin embargo no ofrece opciones respecto a la sincronización, por lo que resulta recomendable la utilización de dicha herramienta en combinación con otras.

7.1.3.4.2 Sitecore Courier

Sitecore Courier [47] tiene como objetivo acortar la brecha entre el desarrollo y los entornos de producción cuando la construcción de los sitios web es realizada con Sitecore. La herramienta permite crear paquetes analizando artículos de Sitecore serializados y empaquetando únicamente los que fueron modificados. Estos paquetes son persistidos de forma independiente al código, limitando el control y versionado de los mismos.

7.1.3.4.3 Unicorn versión 3.3.1

Unicorn [48] resuelve el problema de la sincronización escribiendo copias serializadas de los elementos de Sitecore en el disco junto al código. De esta manera, una copia de los elementos de la base de datos -necesarios para la solución- forma parte del código fuente del proyecto. Los mismos son versionados en el repositorio GIT, lo cual ofrece una ventaja fundamental de administrar y versionar los archivos que representan los datos de Sitecore, por la cual decidimos utilizar Unicorn versión 3.3.1 como herramienta de sincronización. Por otra parte, también proporciona una interfaz web integrada con Sitecore para sincronizar o serializar los ítems que los desarrolladores consideren necesarios.

7.2 Estructura del proyecto

La estructura fue creada a partir de un proyecto MVC vacío en el cual se agregaron las carpetas *App_Config* y *SummitEvent*.

Las carpetas *Controllers*, *Models* y *Views* tienen una estructura común y constan de dos subcarpetas llamadas *Content* y *Structure*. *Content* agrupa todo lo relacionado a los componentes específicos cuyo contenido varía según cada instancia.
Por otra parte, en la carpeta *Structure* se encuentran los componentes cuyo contenido es el mismo a través de todo el sitio web. A su vez, cada carpeta citada contiene una subcarpeta denominada *Components*, la cual contiene controladores, modelos o vistas según su carpeta padre.
En la Figura 62 se ilustra lo descrito anteriormente para el caso de los modelos.

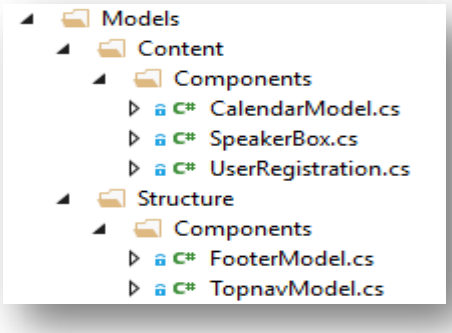


Figura 62 – Captura de pantalla de la estructura del proyecto en Visual Studio. Fuente: Autoría Propia.

La misma estructura es aplicada al directorio en donde se encuentran los ítems de Sitecore que son sincronizados a través de Unicorn.

Se realizó esta organización para mantener un mapeo más sencillo entre el proyecto de Visual Studio y la sincronización de la instancia Sitecore.

Del lado de la instancia de Sitecore, se realiza una organización similar para las carpetas *Renderings*, *Layouts* (a su vez dentro de *Layout*), *Templates* y *Models*.
Dentro de las citadas carpetas, se crea otra con el nombre del proyecto, en este caso *SummitEvent*, para separar de forma clara los datos referido al proyecto (puede haber varios en una misma instancia) del contenido propio de Sitecore. Como se ve en la Figura 63 –para el caso de *Renderings*- la estructura interna a la carpeta citada es la misma que se utilizó en Visual Studio.

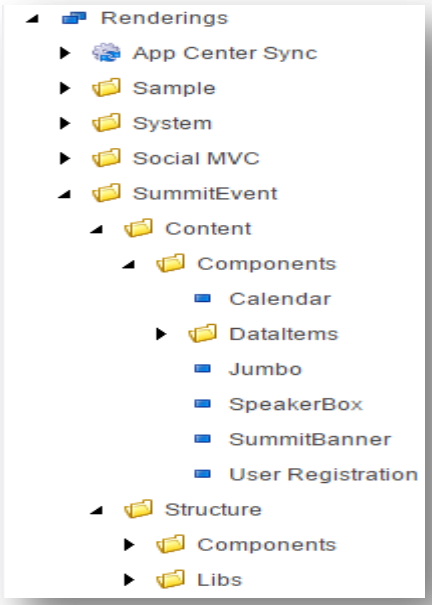


Figura 63 – Captura de pantalla de la estructura del proyecto en Sitecore. Fuente: Autoría Propia.

7.3 Componentes

De manera similar a AEM y utilizando los mismos criterios, decidimos dividir los componente en estructurales y de contenido.

Como ya se mencionó al comienzo del capítulo, vamos a detallar la implementación del *template* de página –BasePage-, los componentes Topnav, Footer y Jumbo.

7.3.1 Componentes Estructurales

7.3.1.1 BasePage

Se creó un *template* llamado BasePage así como un *layout* con el mismo nombre, los cuales están relacionados y juntos representan a todas las páginas que integran el sitio web. El *template* cuenta con un campo booleano de tipo *checkbox* llamado *Hide in Topnav* que va a ser utilizado por los autores para mantener las páginas que prefiera ocultas del navegador, a su vez, el *layout* tiene asociado un archivo *BasePage.cshtml* que contiene la estructura HTML principal de la página del sitio web e incluye tres *placeholders*. Los mismos son *topnav, main y footer,* a su vez dicho archivo

tiene asociado de manera estática dos *view rendering* -Footlibs y Headlibs- los cuales son encargados de importar las diferentes librerías de estilos y *scripts* de todo el sitio.

Con el objetivo que todos los ítems creados a partir del *template* BasePage contengan una configuración pre definida, se utiliza la funcionalidad de *Standard Values* y mediante la misma se asocia el *layout* antes mencionado. A su vez se incluyen los *renderings TopNav* y *Footer* en los *placeholders topnav* y *footer* respectivamente de manera inicial. De esta forma todos los ítems creados a partir del *template BasePage* van a contar con un *layout*, tres *placeholders* y cuatro *renderings* encargados de las libreras, navegación y *footer*.

7.3.1.2 Headlibs y Footlibs

Es importante notar que tanto *Headlibs* como *Footlibs* se definieron como *View Renderings*, ya que al no aplicar ningún tipo de lógica ni de manejo de datos (solo se realizan importaciones), no son necesarios ni un modelo ni un controlador.
En el cabezal (sección *head*) del archivo *BasePage.cshtml* se realizan las importaciones de archivos de estilos CSS mediante la asociación estática del *view rendering Headlibs*. El mismo en su campo *path* hace referencia a otro archivo cshtml el cual contiene las importaciones propiamente dichas.
De forma análoga pero dentro del cuerpo del HTML del archivo *BasePage.cshtml* se realiza la importación de *scripts*, esta vez mediante el *view rendering Footlibs*.
En la siguiente figura -Figura 64- se muestra la relación entre el archivo correspondiente a *BasePage*, los *View Renderings Headlibs* y *Footlibs* y sus correspondientes archivos cshtml:

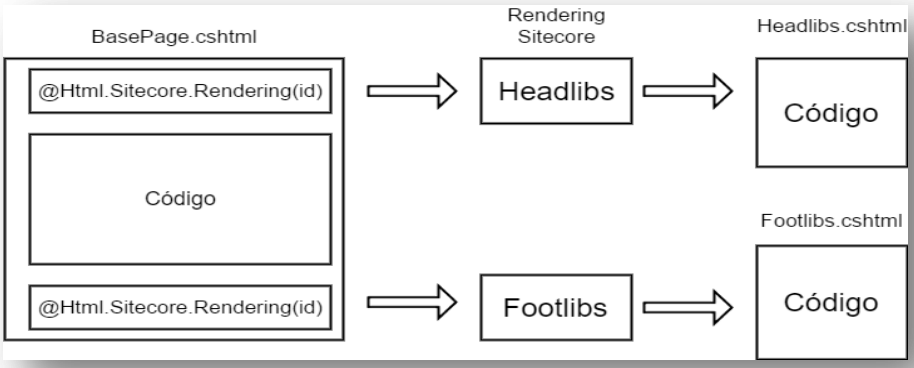


Figura 64 – Relación de archivos que involucran a los renderings Headlibs y Footlibs. Fuente: Autoría Propia.

7.3.1.3 Topnav

Topnav muestra los enlaces a todas las páginas hijas de primer nivel que no tengan el *checkbox Hide in Topnav* activado. El procesamiento de las páginas hijas que serán visibles en el Topnav requiere de algunas operaciones lógicas, por lo tanto es necesario el uso de un controlador que las realice.

En consecuencia, el desarrollo se hace a partir de un *Controller Rendering*, referenciando al controlador y método ambos de nombre *Topnav* en el proyecto. Este tipo de *rendering* no permite

la asociación de un modelo en Sitecore, por lo tanto el mismo será instanciado a través del controlador.
Para la parametrización se utilizó un ítem como fuente de datos. El ítem en cuestión es el llamado *TopNavDS*. El mismo contiene los campos de información del Topnav y fue instanciado a partir del *template TopNavData* creado por nosotros, conteniendo los campos que se observan en la Figura 65.

La utilización de un ítem como fuente de datos busca mantener una misma configuración para todas las instancias del *Topnav* que se creen a través del sitio web.
Se obtiene como resultado la centralización de toda la configuración en el ítem *TopNavDS*.

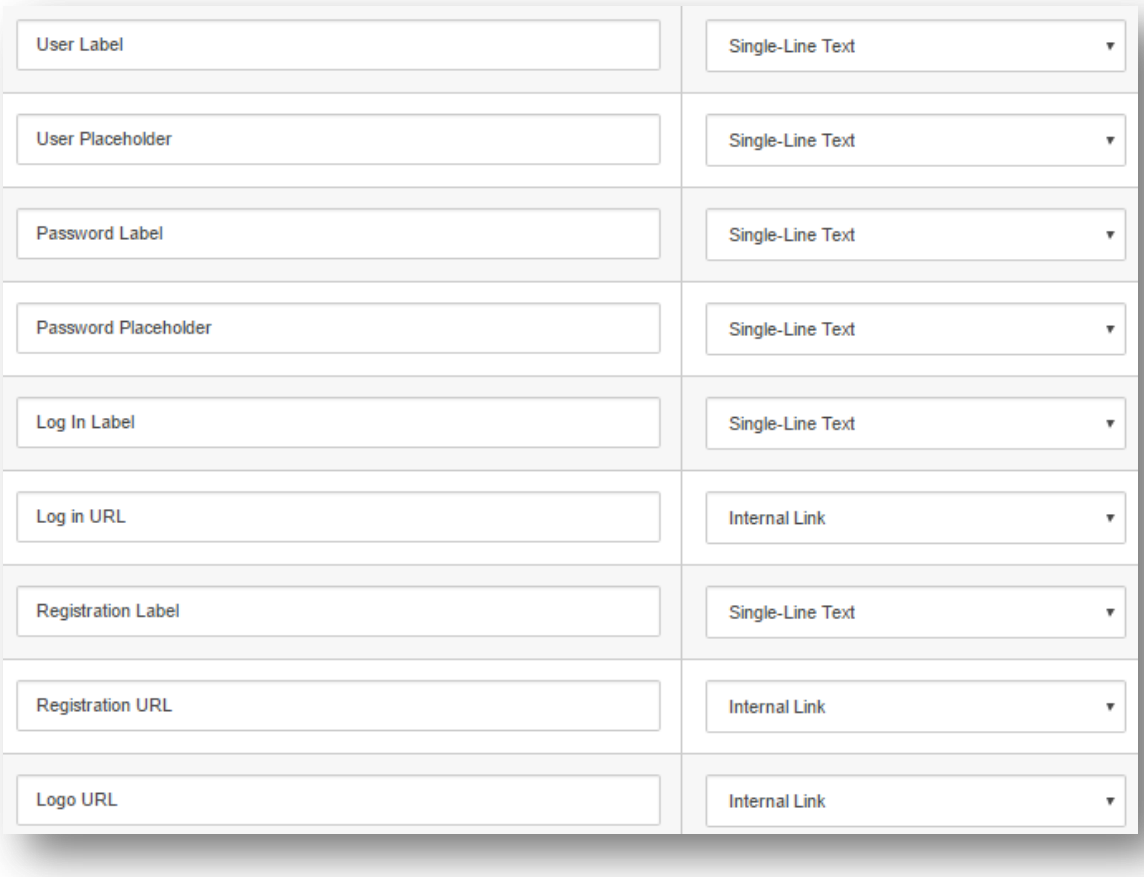


Figura 65 – Captura de pantalla que muestra los campos del Topnav. Fuente: Autoría Propia.

7.3.1.3.1 Log in del Topnav

Brinda la posibilidad a los usuarios del sitio web de poder identificarse o también ser redirigidos a la página de registro (componente *Register*). Como vemos en la Figura 41, el *log in box* permite ingresar nombre de usuario y contraseña. Al hacer clic en el botón de *Log In* se comunica utilizando AJAX con una clase especifica que creamos para el manejo de usuarios.

7.3.1.4 Footer

Este componente no requiere de la implementación de lógica de negocio, por lo tanto es implementado por un *View Rendering,* el mismo utiliza una vista y un modelo.

Para la parametrización se utilizó un ítem como fuente de datos. El ítem en cuestión es el llamado *FooterDS*. El mismo es el que contiene los campos de información del *Footer* y fue creado a partir del *template FooterData*.
La única particularidad de este componente es que uno de sus campos de información es de tipo *multilist,* es utilizado para listar y seleccionar los posibles *links* que el autor quiera incluir en el *rendering*. Para cumplir con dicho objetivo debemos especificar la ruta correspondiente a la carpeta que contiene los ítems *Static Links* que representan *links* estáticos. De esta manera se pueden agregar, quitar y editar los links disponibles desde el editor de contenido y así modificar la configuración del ítem *FooterDS*, obteniendo como resultado que la presentación de todas las instancias del *rendering Footer* sea actualizada.

7.3.1.4.1 Static Links

El *template Static Link* representa *links* estáticos. Los ítems creados a partir de dicho *template* no tienen ningún archivo u elemento relacionado con código.
Es necesario usar el editor de contenido de Sitecore para crearlos y contienen dos campos:

- URL de tipo *path*.
- *Text* de tipo texto.

Los ítems son almacenados en la ruta *sitecore/content/Summit/GenericData/StaticLinks*.

7.3.2 Componentes de contenido

A partir de tener la estructura del sitio web se pasa a construir los componentes directamente relacionados al contenido.

7.3.2.1 Jumbo

El componente Jumbo fue implementado de dos maneras distintas.
En primera instancia se desarrolló utilizando un *Controller Rendering* el cual utiliza una vista, un modelo y un controlador, dado que aunque el *rendering* no implementa lógica de negocio si debe procesar un conjunto considerable de campos.
Para la parametrización se utiliza un *Parameter Template*, que es referenciado en las propiedades del *rendering*. Se pueden observar sus campos en la Figura 66:

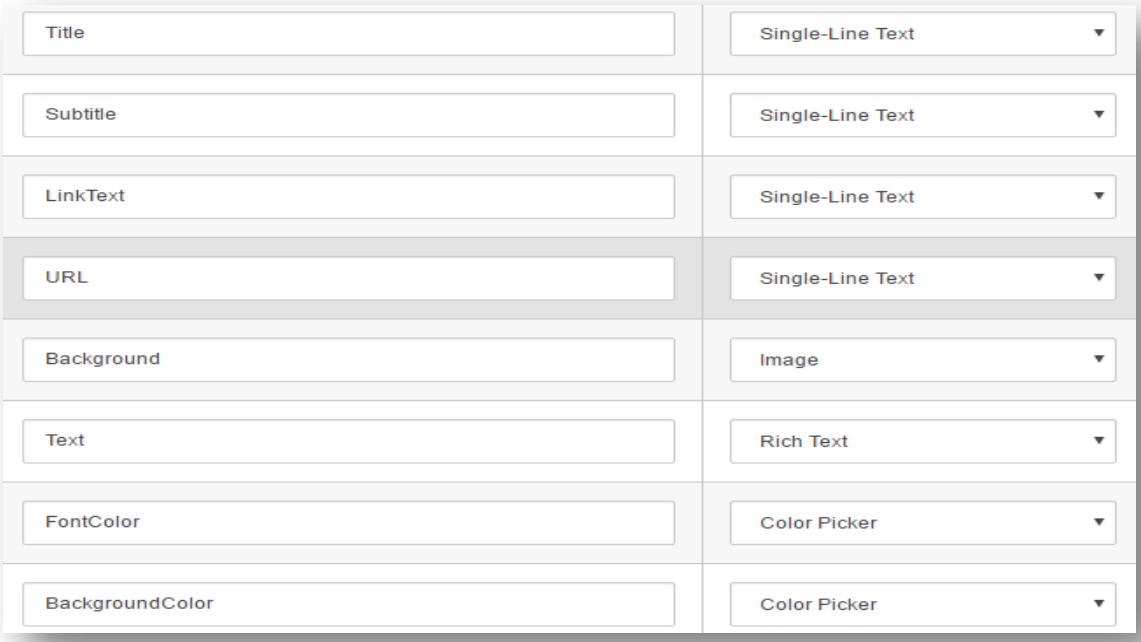


Figura 66 – Captura de pantalla que muestra los campos del Jumbo. Fuente: Autoría Propia.

La segunda alternativa surgió a partir de la interacción con un arquitecto de software con experiencia en Sitecore. El mismo nos sugirió investigar el *framework* Glass.Mapper [43]. Éste permite mapear los valores de los campos asociados a los ítems de Sitecore a una clase C# mediante notaciones sencillas y sin tener que crear código para obtener los valores de los campos. A partir de esto decidimos incluir al menos un *rendering* utilizando dicho *framework* a pesar de que la implementación del caso de estudio en Sitecore ya había sido finalizada.

La utilización del *framework* permite disminuir la cantidad de archivos que deben ser desarrollados eliminando así el controlador, por este motivo esta segunda implementación del componente fue realizada con un *View Rendering*, el cual ni siquiera fue necesario asociarle un modelo, ya que el *framework* se encarga de inicializarlo y mapearlo. De esta forma se reduce de manera significativa la complejidad y cantidad de código que debe ser generado para desarrollar un *rendering* con el mismo comportamiento.

7.4 Creación de contenido

Por último y habiendo implementado los componente definidos anteriormente, pasamos a crear las páginas del caso de estudio e incluir los componentes requeridos en cada una. La misma va a estar constituida por seis páginas, como vemos en la Figura 67, primero tenemos el nodo referente a nuestro sitio –*Summit*- que representa a la página raíz de nuestro sitio -*Home Page*-. Cuenta con tres páginas hijas, *Adobe Attend, Agenda* y *Registration.* A su vez existe una carpeta llamada *GenericData* donde son persistidos los ítems de los *speakers*, *static links* y eventos de calendario.
La página *Registration* tiene definido *Hide in TopNav* en verdadero, para que no esté disponible desde el *Topnav*.

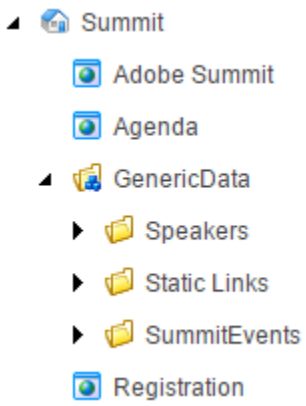


Figura 67 – Captura de pantalla mostrando las páginas y carpetas creadas para el caso de estudio. Fuente: Autoría Propia.

7.5 Solución móvil

7.5.1 Xamarin

Xamarin [49] es una *cross platform* basada en .NET, lo cual significa que programando en C# se pueden implementar aplicaciones móviles tanto para *Android*, *iOS* como *Windows Phone*.

En abril del año 2015 Sitecore anuncia la asociación con Xamarin con el objetivo de expandir su oferta en desarrollo móvil. De esta forma Sitecore ofrece una API -*Sitecore MobileSDK Xamarin*- para poder comunicarse con sus servidores desde aplicaciones creadas utilizando *Xamarin*.

Para la solución móvil no fue necesario el desarrollaro de nuevos componentes, ya que la API ofrecida para trabajar con *Xamarin* no maneja *renderings,* sino que es utilizada para obtener ítems y sus datos. Se desarrolló un proyecto para la familia de sistemas operativos *Android*. No se realizó para otros sistemas operativos (*iOS*, *Windows Phone*) ya que no se disponía de dispositivos con esos sistemas operativos (necesario para el caso de *iOS*).

Se hizo uso de la API antes mencionada en su versión 1.3.0, la cual permite una gran variedad de funciones en lo que respecta a ítems. Las más importantes para llevar a cabo este proyecto fueron en particular la lectura, creación y borrado de ítems (también imágenes para el caso de la lectura). La aplicación está compuesta por tres actividades de *Android* que son representadas mediantes clases de C# (ver Figura 71):

7.5.2 Funcionalidades

7.5.2.1 Inicio de sesión

Es la actividad inicial de la aplicación móvil -Figura 68-. Se permite el acceso, búsqueda y registro/baja de eventos a los usuarios registrados en el sitio web. La misma se comunica con el servidor para validar la combinación de usuario y contraseña ingresada.

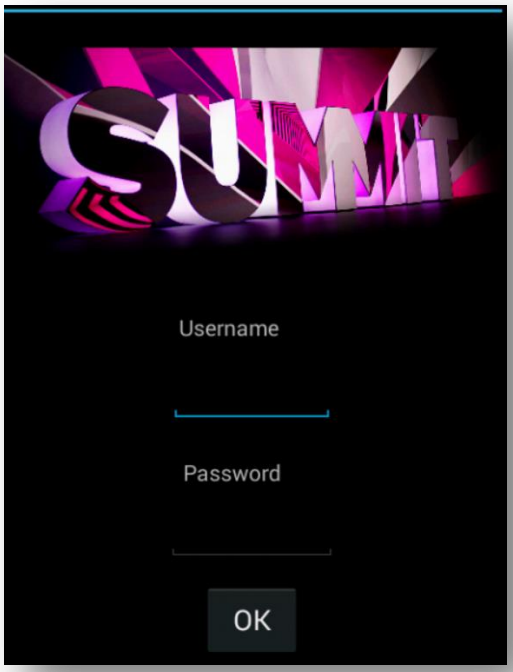


Figura 68 - Captura de pantalla de la aplicación móvil desarrollada. Fuente: Autoría propia.

7.5.2.2 Búsqueda de eventos por fecha

Una vez logueado el usuario es enviado a la actividad de búsqueda de eventos -Figura 69-, la cual provee de un calendario que filtra por fecha los eventos y los despliega por nombre en orden alfabético.

7.5.2.3 Inscripción/baja de evento

Esta actividad se despliega al seleccionar un evento en el calendario descrito anteriormente y muestra la información más relevante del mismo -Figura 70-. Se ofrece la opción de darse de baja o registrarse, dependiendo del caso. Utilizando la API *Sitecore MobileSDK Xamarin* creamos un ítem de reserva desde la aplicación móvil en la instancia de Sitecore o eliminando la reserva en caso que el usuario quiera darse de baja.

En la Figura 71 se puede observar un resumen de la interacción de los usuarios con la aplicación y el servidor de Sitecore. Para que el usuario pueda llegar a realizar una reserva en un evento debe pasar por un conjunto de tres pasos secuenciales. El usuario comienza identificándose con un usuario y contraseña, el siguiente paso consiste en seleccionar una fecha para que la aplicación despliegue los eventos disponibles. Finalmente el usuario selecciona uno y realiza o elimina una reserva.

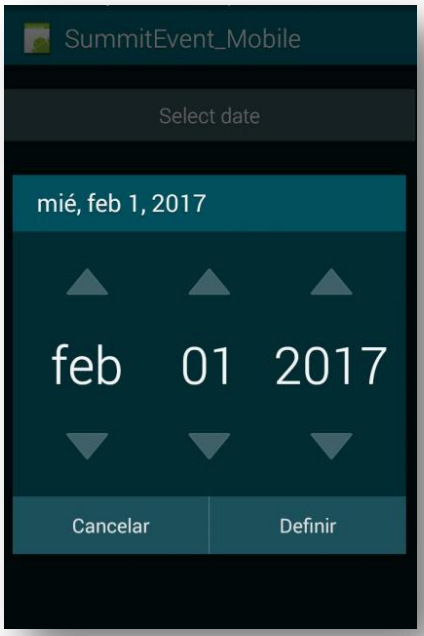


Figura 69 - Captura de pantalla de la aplicación móvil desarrollada, calendario para búsqueda de ventos. Fuente: Autoría propia.

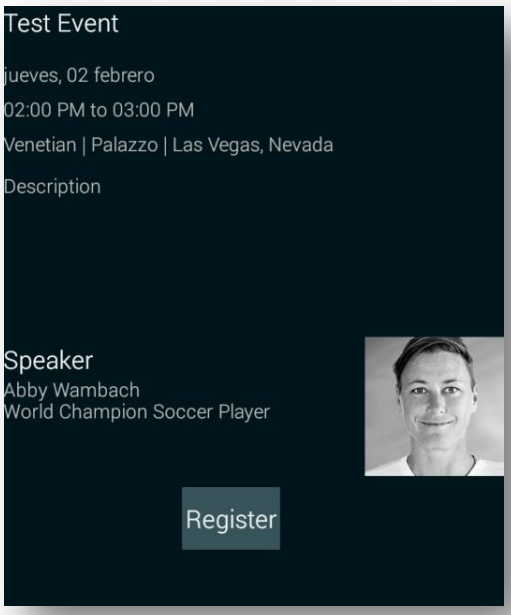


Figura 70 - Captura de pantalla de la aplicación móvil desarrollada. Fuente: Autoría propia.

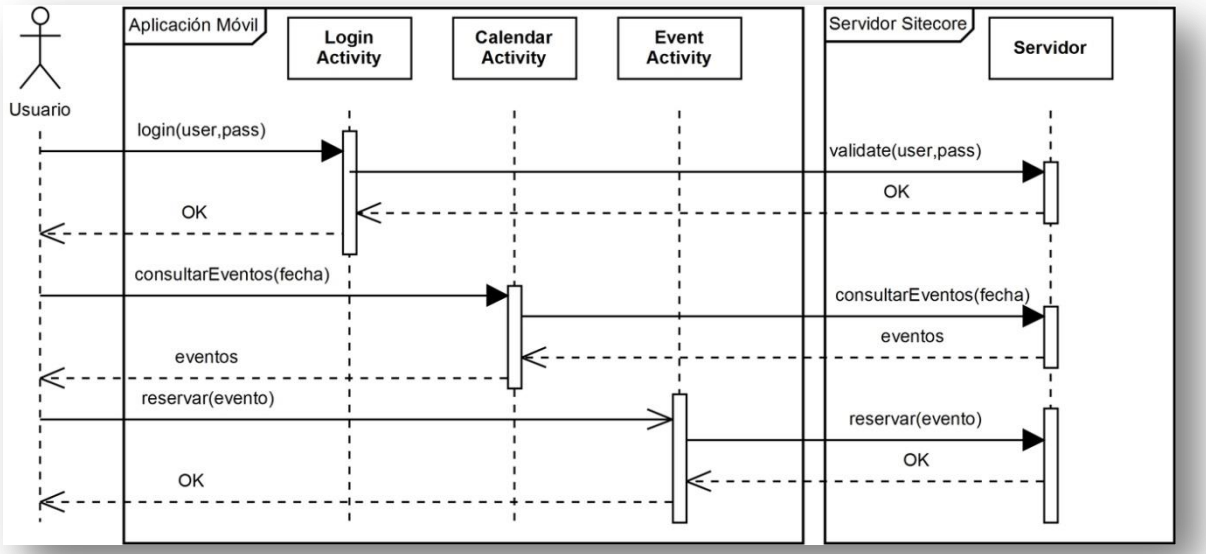


Figura 71 – Diagrama de secuencia de la comunicación de los usuarios con la aplicación y ésta con el servidor. Fuente: Autoría Propia.

7.6 Resumen del capítulo

A través de la implementación del caso de estudio nos fue posible comprender el funcionamiento de Sitecore. A su vez descubrimos las diferencias que tiene con la plataforma AEM desde diferentes puntos de vista -autores y desarrolladores-. Además fue posible implementar una aplicación móvil utilizando *Xamarin* y que la misma se comunique con nuestro sitio web hecho en Sitecore.

Capítulo 8: Comparación de Plataformas –AEM vs Sitecore-

En el presente capítulo compararemos las plataformas en las áreas abarcadas dentro del caso de estudio, nombrando las ventajas y desventajas que ofrece cada una. Para esto vamos a utilizar dos puntos de vista, el de un autor y el de un desarrollador, ya que son los roles más preponderantes que utilizan la plataforma.

8.1 Acceso a la documentación

Una de las primeras actividades que realiza cualquier desarrollador interesado en el uso de Sitecore o AEM es la búsqueda de información. La documentación oficial brindada por ambos productos resulta insuficiente a la hora de poner en práctica los conocimientos brindados para implementar componentes de contenido. Ambas plataformas cuentan con documentación oficial en sus sitios webs, aunque sólo Sitecore ofrece bibliografía exclusiva para desarrolladores [50]. Esto le otorga una pequeña ventaja respecto al acceso de la información.

AEM, por su parte, brinda bibliografía pero ésta se encuentra enfocada a los autores. En lo que respecta a los cursos, ambas plataformas ofrecen entrenamientos para desempeñar los distintos roles que participan en el producto –desarrolladores, arquitectos, autores, vendedores, etcétera-. La totalidad de dichos cursos son realizados en Estados Unidos y Europa.

De esta manera, la mayor parte del conocimiento adquirido para el desarrollo en dichas plataformas fue generada a partir de una combinación de la información brindada por sus empresas junto a las comunidades activas de desarrolladores en foros oficiales y no oficiales; sumados a la experiencia de Conexio y experimentación del uso de las plataformas.

Por último, cabe destacar que la curva de aprendizaje es alta en ambas herramientas.

8.2 Desde un punto de vista del desarrollador

Además de las diferencias que presentan *per se* las tecnologías JAVA y .NET, podemos comparar las distintas soluciones que proponen las plataformas para resolver los problemas que atañen a las herramientas WCMS.

8.2.1 Persistencia de contenido

Sitecore utiliza una base de datos relacional para persistir sus datos, provocando un problema de sincronización de instancias de la solución de un proyecto. Esto genera una dificultad a la hora de implementar un ambiente colaborativo en un proyecto distribuido donde participan diferentes actores. Para poder solucionar el inconveniente se debe recurrir a herramientas externas, ya que Sitecore no brinda una solución propia. Sin embargo, AEM lo soluciona de una manera simple, unificando el contenido y los datos de la instancia junto con el código fuente en formato XML. Todos los actores que pertenezcan al proyecto y desplieguen la solución en sus servidores locales generarán los mismos datos.

8.2.2 Herramientas de desarrollo

Visual Studio combinado con el *plug-in Sitecore Rocks* brindan todo lo necesario para desarrollar nuevas funcionalidades, siendo innecesario para los desarrolladores acceder a la instancia web de

Sitecore –exceptuando el testeo de componentes –. No obstante, esto no sucede en AEM donde el desarrollador se ve forzado a utilizar varios IDEs para poder implementar nuevas funcionalidades –en el caso de estudio se utilizaron tres IDEs diferentes-. El *plug-in* ofrecido en Eclipse resulta insuficiente lo que obliga a utilizar el ambiente web que ofrece AEM para crear nodos y componentes.

8.2.3 Extensión y Personalización

Sitecore ofrece a los desarrolladores la capacidad de sobrescribir o extender procesos propios a través de los *pipelines*. Esto amplía la posibilidad de modificar tareas vitales de la plataforma, como por ejemplo el renderizado de ítems. En este punto, Sitecore posee una capacidad de personalización y extensión de procedimientos internos sensiblemente mayor que AEM. Siendo AEM una plataforma en la cual la tarea de interferir o personalizar sus procesos internos resulta bastante más compleja y limitada para el desarrollador.

8.2.4 Patrones de Diseño

Sitecore utiliza MVC como patrón base para desarrollar soluciones en la plataforma, esto ofrece un camino claro a la hora de tomar decisiones de diseño por parte de los desarrolladores. A su vez, proporciona una marcada independencia entre la capa de presentación y datos, resultando más fácil representarlos mediante ítems cuya única función es contener información.

Por otra parte, en AEM resulta más difusa la creación de nodos específicos para la representación de datos. Por ejemplo, en el caso de los *speakers* los datos debieron ser representados mediante componentes -los cuales, a su vez, deben ser instanciados en una página-. Si bien AEM no ofrece explícitamente un patrón de diseño para desarrollar soluciones en su plataforma, sí brinda una gran cantidad de componentes y ejemplos de sitios web que vienen preinstalados en la plataforma. Los mismos resultan de gran ayuda para los desarrolladores, quienes pueden utilizar dichas prestaciones para implementar nuevas soluciones.

8.2.5 Administración de Librerías

En los proyectos web de gran porte siempre existen una gran cantidad de librerías de estilos y *script*s externas. Su administración se vuelve una tarea engorrosa incluso para los desarrolladores más experimentados.

AEM ofrece un mecanismo de manejo de librerías de estilos y *scripts* que resulta muy beneficioso para la organización de la estructura del proyecto. AEM fomenta así la creación de componentes modulares, donde cada uno tiene asociado las librerías que utiliza para su funcionamiento, facilitando su exportación a otras soluciones. Este mismo mecanismo permite administrar las librerías globales del sitio. Por su parte, Sitecore no ofrece ninguna funcionalidad específica al respecto. Esto genera que el desarrollador tenga que ingeniárselas para poder administrar de forma eficaz las diferentes librerías –estilos y *scripts*- utilizados en el proyecto.

8.2.6 Desarrollo de aplicaciones móviles

Sitecore introdujo *Xamarin SDK API* hace menos de dos años brindando la posibilidad a los desarrolles móviles de comunicarse desde la aplicación a la instancia del servidor. No obstante,

todo lo que refiere a la presentación visual de la aplicación móvil se encuentra encapsulado dentro del código de la misma. En consecuencia los autores se encuentran limitados en este punto. Lo que ofrece la API termina siendo una forma sencilla de consumir datos de un servidor Sitecore desde una aplicación móvil, pero lejos está de ofrecer una solución que permita adaptar contenido y crear componentes específicos.

En este punto AEM se fortalece, integrando el *framework Phonegap* –creado por Adobe- el cual permite crear una aplicación en diferentes plataformas a partir de código HTML, CSS y Javascript. Esto tiene como resultado que el ambiente utilizado por los autores es el mismo para aplicaciones móviles y sitios web. Asimismo los desarrolladores son capaces de crear componentes para aplicaciones móviles que resultan similares a los utilizados en sitios web. Concluyendo que es posible crear aplicaciones de mayor complejidad utilizando Xamarin que Phonegap, pero la utilización de Phonegap ofrece un mayor poder de personalización para los autores y simplicidad a la hora de cambiar contenido.

8.3 Desde un punto de vista del autor

8.3.1 Configuración de componentes

AEM ofrece la posibilidad de crear diálogos simples y comprensibles para los autores a la hora de crear contenido para sus componentes. Los desarrolladores cuentan con una amplia variedad de configuraciones para desplegar estos diálogos. Esto permite que la tarea de configuración de componentes resulte simple. A su vez, los componentes pueden ser agrupados mediante categorías -lo cual facilita la búsqueda de los mismos-. Las ventanas para configurar el contenido de los *renderings* -análogos a los componentes de contenido de AEM- en Sitecore son menos flexibles, pero son más simples de implementar. En casos complejos esto puede generar que la configuración resulte más tediosa y menos intuitiva para los autores respecto de AEM.

Asimismo, la utilización de ítems como fuentes de datos permite a los autores editar en línea –es decir, en la vista HTML- no sólo el texto de los componentes sino también las imágenes. Por otra parte, en AEM los autores tienen únicamente la capacidad de editar en línea un campo de texto por componente.

8.3.2 Facilidad de uso

El ambiente de autor ofrecido por AEM resulta intuitivo y simple de utilizar incluso para los autores menos experimentados. Es posible navegar a través de los diferentes modos de una forma sencilla. Además, ofrece un *sidekick* –menú vertical- que despliega en forma de íconos los componentes autorizados a utilizar en la página donde se encuentra el autor (simplemente debe arrastrarlos al *parsys* para incluirlos).

Por otro lado, el ambiente de autor de Sitecore resulta menos intuitivo, ya que muchas de las funcionalidades disponibles son poco utilizadas en la creación de contenido. La inclusión de componentes resulta un poco más tediosa que en AEM, dado que el autor debe hacer clic en el *placeholder* donde se desea agregar el contenido, y luego seleccionar entre los *renderings*

disponibles. Esta selección mostrará todos los *renderings* de la solución, siempre y cuando el desarrollador no limite el *placeholder.*

Lo mencionado anteriormente se puede ver reflejado en varios sitios donde las distintas comunidades de usuario realizan evaluaciones de ambas plataformas [51]. En este aspecto la mayoría destaca la facilidad que tienen los autores en AEM para administrar y crear contenido.

8.4 Comparativa de Sitecore y AEM

A partir de la implementación del caso de estudio se puede concluir que las ventajas de Sitecore contra AEM son -Tabla 1-:

- Clara definición entre la capa de presentación, datos y lógica que se basa en el patrón MVC, el cual oficia de guía a la hora de tomar decisiones de diseño sobre el desarrollo de soluciones.
- Mayor flexibilidad a la hora de extender y generar funcionalidades específicas de procesos internos de la plataforma.
- Los desarrolladores pueden realizar todas las tareas que necesitan dentro de un único IDE –Visual Studio-.
- Los diálogos para configurar los *renderings* resultan más fácil de implementar que AEM.
- Con Xamarin es posible desarrollar aplicaciones móviles más complejas, pero los autores no tiene la posibilidad de modificar su presentación desde Sitecore. Esto se debe a que la estrategia de Sitecore es ofrecer una API la cual es utilizada desde las aplicaciones únicamente para consumir datos del servidor.
- Sitecore ofrece un libro para desarrolladores, el cual brinda un punto de partida para comenzar a utilizar dicha plataforma.

En cuanto a AEM, las ventajas obtenidas con respecto Sitecore son:

- Ofrece una forma eficiente y ordenada para administrar las librerías de estilos y *scripts*.
- Provee una mayor flexibilidad en lo que respecta a la creación de diálogos de componentes, pudiendo crearlos de forma personalizada, lo que genera diálogos más fáciles de usar para los autores. En contraparte éstos tiene una implementación más compleja.
- Ambiente de autor más rico e intuitivo, brindando funcionalidades que facilitan las tareas de edición. Un ejemplo de esto es el menú vertical que despliega en forma de iconos todos los componentes disponibles para usar y se pueden incluir en la página simplemente arrastrándolos a un *parsys*.
- La sincronización del proyecto resulta más fácil, ya que los datos de la instancia son persistidos junto con el código fuente. De esta forma se resuelven la mayoría de los problemas que surgen cuando se trabaja en equipos de desarrollo de varios integrantes.
- Los autores tienen la capacidad de agregar o quitar componentes en las soluciones móviles creadas en AEM –integrado con PhoneGap-, de esta forma son capaces de modificar la presentación visual de la aplicación. Por otro lado la complejidad de las mismas se ven limitadas debido al lenguaje que debe ser utilizado para su desarrollo –HTML, JavaScript, CSS-.
- AEM ofrece más de 40 componentes incorporados –*out of the box*- y prontos para usar, los cuales sirven como base para poder desarrollar otras funcionalidades.

Tabla 1 – Tabla de comparación entre las plataformas AEM y Sitecore. Fuente: Autoría Propia

	AEM	Sitecore
Clara definición de capa de presentación y datos, de contenido y componentes.	✗	✓
Administrador propio de librerías de Estilos y Scripts	✓	✗
Sincronización de datos de la instancia de la plataforma sin utilización de herramientas externas	✓	✗
Componentes de contenido pre definidos incluidos en la plataforma	✓	✗
Facilidad a la hora de extender o modificar funcionalidades propias de la plataforma	✗	✓
Desarrollo de aplicaciones móviles y administración de contenido embebido en la plataforma	✓	✗
Bibliografía exclusiva para desarrolladores	✗	✓

8.5 Selección de plataforma a partir de diferentes puntos de vista

A partir de la comparación de ambas plataformas, pudimos generar arboles de decisiones los cuales pueden ser utilizados por los diferentes actores que utilizan la plataforma. Los cuales son los desarrolladores -Figura 72-, los autores -Figura 73 - y desde el punto de vista de un cliente -Figura 74-. El objetivo es a partir de ciertas preguntas, poder tomar una decisión a en cuanto a la plataforma a utilizar –Sitecore o AEM-, no se tiene en cuenta los costos de las licencias de ambas plataformas y solo se enfoca en la administración de contenido. A continuación se va a realizar una breve justificación de cada hoja del árbol.

8.5.1 Punto de vista del desarrollador

La Figura 72 muestra el árbol de decisión desde el punto del desarrollador y tiene 11 hojas numeradas cuyas justificaciones son:

1. AEM trabaja con JAVA.
2. Sitecore trabaja con .NET.
3. AEM viene con más de 200 componentes predefinidos de ejemplos y varios sitios web.
4. Sitecore tiene mayor documentación e incluso tiene un libro para desarrolladores.
5. AEM es menos flexible en cuanto a modificar su funcionalidad.
6. Sitecore provee varios mecanismos para modificar su funcionalidad.

7. Sitecore promueve el desarrollo de componentes de contenido basado en el patrón de diseño MVC.
8. Varias funcionalidades de AEM, incluyendo el desarrollo móvil requieren de conocimientos de tecnologías *front end.*
9. Sitecore solo funciona en Windows.
10. AEM funciona en todos los SO.
11. Si el lector llego a este punto le es indiferente la utilización de cualquiera de las plataformas.

8.5.2 Punto de vista del autor

La Figura 73 muestra el árbol de decisión desde el punto del desarrollador y tiene 10 hojas numeradas cuyas justificaciones son:

1. AEM ofrece un ambiente de autor más intuitivo y fácil de utilizar sin experiencia previa.
2. AEM permite arrastrar los componentes de contenido para incluirlos en las páginas del sitio web.
3. Sitecore ofrece la posibilidad de elegir los componentes de contenido a partir de un listado.
4. Sitecore permite realizar cambios manuales de modos de autor modificando la URL, brindando mayor flexibilidad.
5. AEM ofrece una interfaz gráfica simple que permite cambiar de modo en el ambiente de autor fácilmente.
6. La modificación de contenido en línea permite en Sitecore, no solo modificar texto, sino también imágenes.
7. La modificación en línea de contenido de AEM se limita solo a texto.
8. Sitecore brinda la posibilidad a los autores de administrar el contenido de forma independiente a los componentes.
9. En AEM el contenido es creado dentro de los componentes.
10. Si el lector llego a este punto le es indiferente la utilización de cualquiera de las plataformas.

8.5.3 Punto de vista de una posición de responsabilidad del lado del cliente

La Figura 74 muestra el árbol de decisión desde el punto del desarrollador y tiene 6 hojas numeradas cuyas justificaciones son:

1. Sitecore ofrece licencias limitadas para proyectos pequeños.
2. AEM tiene embebida la visualización del contenido móvil en su plataforma web.
3. AEM ofrece un ambiente de autor más intuitivo y fácil de utilizar sin experiencia previa.
4. Sitecore trabaja con .NET.
5. Si el lector llego a este punto le es indiferente la utilización de cualquiera de las plataformas.
6. AEM trabaja con JAVA.

8.6 Resume de capítulo

Se realizó comparativa de ambas plataformas a partir de la implantación del caso de estudio. Los aspectos a destacar serian la desventaja de AEM frente a Sitecore a la hora de crear un ambiente colaborativo, así como la desventaja de no tener un ambiente para la creación de contenido móvil embebido en la plataforma, como si tiene AEM. Por otra parte Sitecore ofrece una clara división entre el contenido y los componentes, cuya base es fundamentada en el patrón de diseño MVC. De todas formas, se destaca que ambas plataformas son excelentes alternativas para proyectos WCMS. Por último se definieron arboles de decisión para los distintos actores, a partir de la comparación de ambas plataformas realizada en este capítulo.

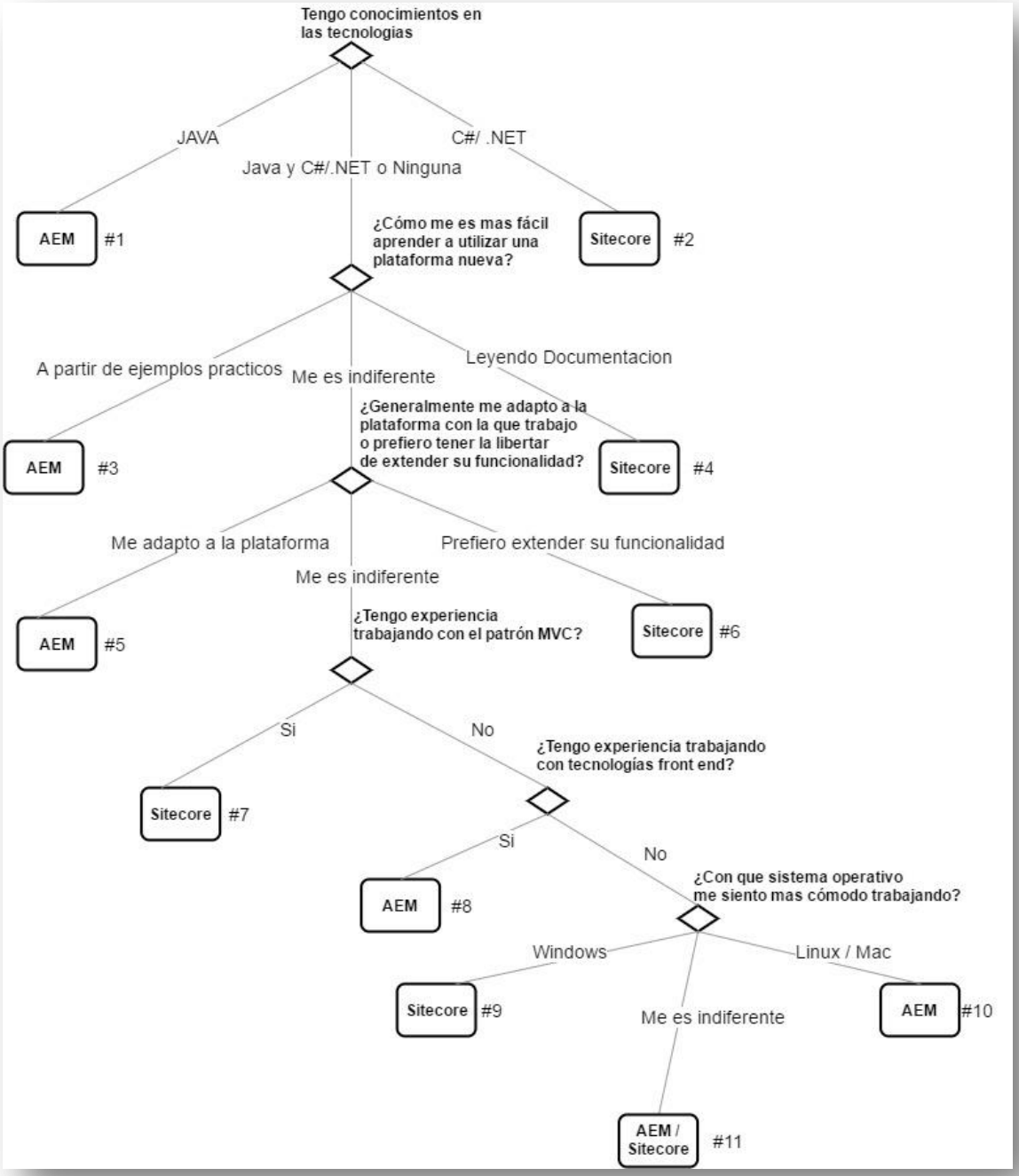


Figura 72 – Árbol de decisión desde el punto de vista de un desarrollador. Fuente: Autoría Propia.

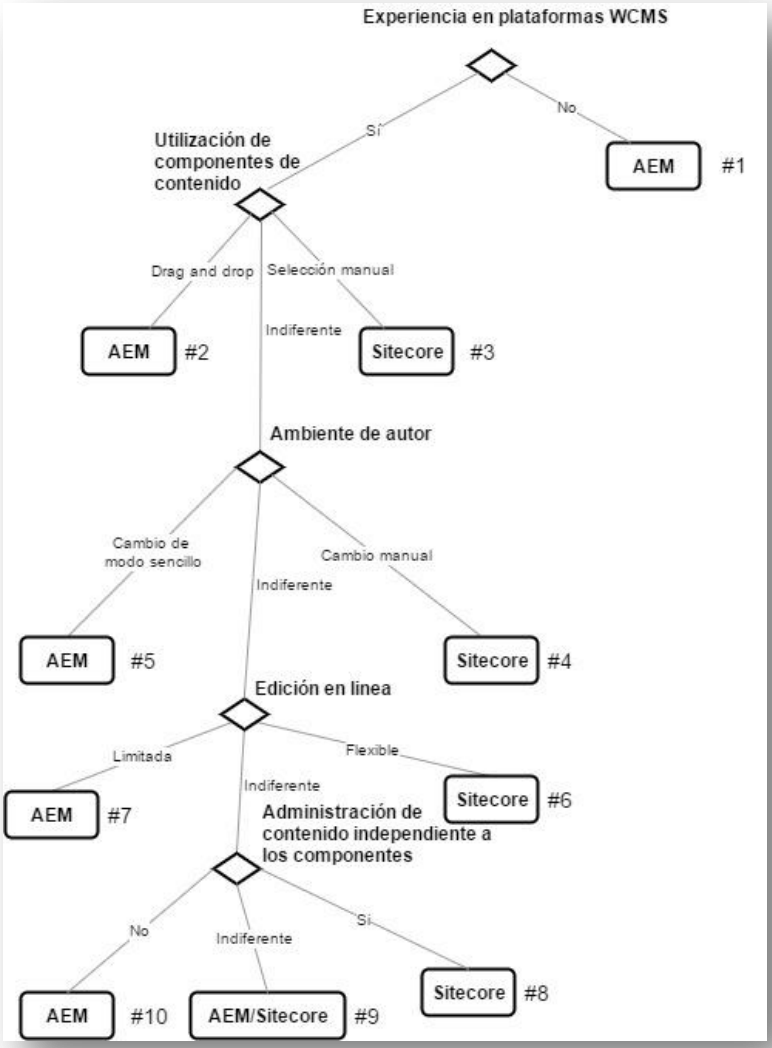


Figura 73 – Árbol de decisión desde el punto de vista de un autor. Fuente: Autoría Propia.

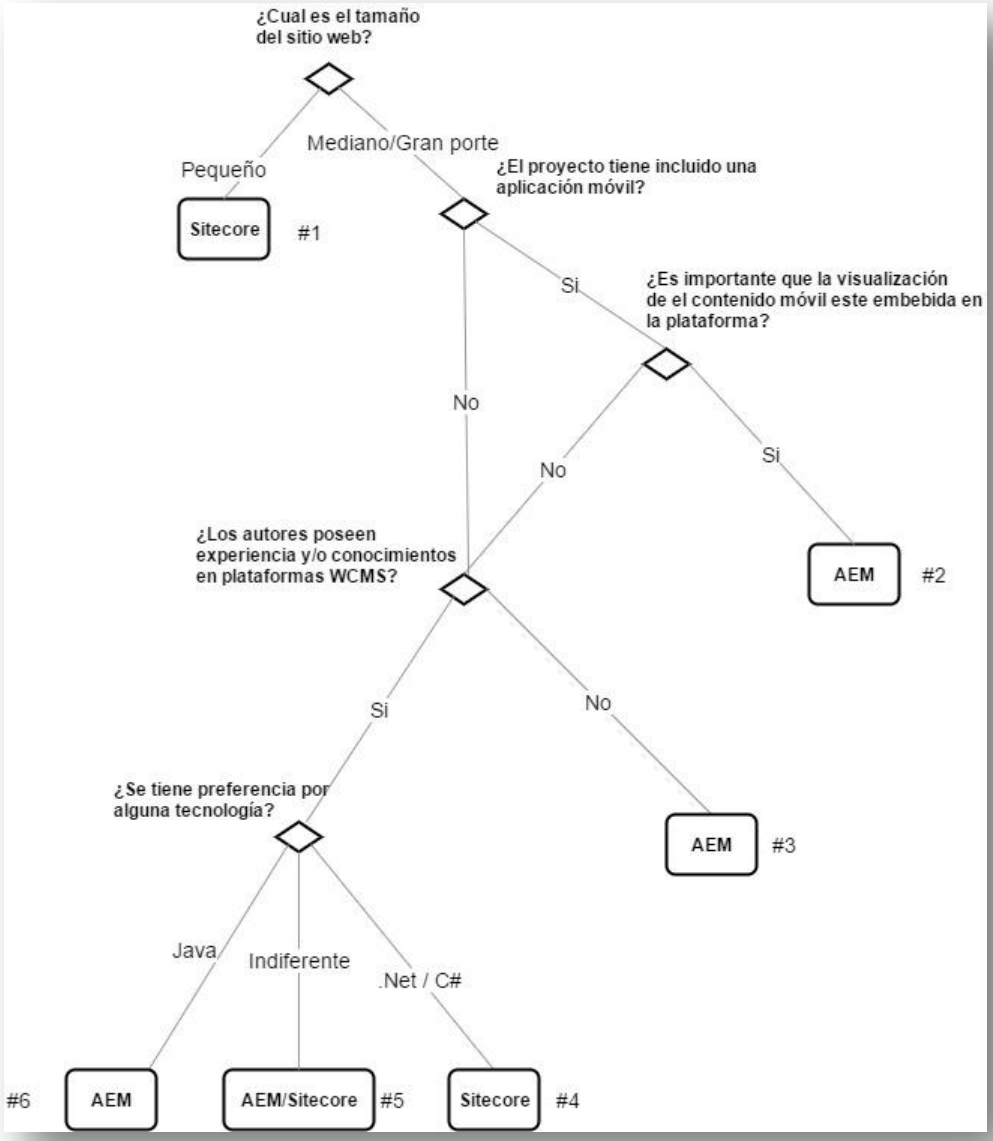


Figura 74 – Árbol de decisión desde el punto de vista de un tomador de decisiones del lado del cliente. Fuente: Autoría Propia.

Parte 3

En esta parte del informe nos enfocaremos en la metodología de desarrollo con plataformas WCMS. Se incluye esta parte para aportar una documentación de valor para Conexio, de forma que pueda utilizarla para futuros proyectos comerciales. También se pretende dar un marco teórico sobre el desarrollo con plataformas WCMS, ya que tiene algunas particularidades si se compara con desarrollo web tradicional, y que consideramos importante exponer.

Capítulo 9: Metodología teórica de desarrollo en un WCMS

El proceso descrito a continuación tiene como objetivo brindar un marco teórico, abarcando todas las posibles etapas, describiendo el proceso de desarrollo en un WCMS [5]. Como se mencionará posteriormente, existen tres tipos de implementaciones, las cuales depende del tipo de proyecto a llevar a cabo. El desarrollo de un proyecto WCMS se puede dividir en dos grandes etapas, pre implementación e implementación, con sus sub etapas correspondientes. Se mostrará en primer lugar una línea de tiempo con las actividades incluidas en cada etapa Figura 75, luego se detallará cada una de ellas.

9.1 Tipos de implementaciones en un WCMS

Las implementaciones WCMS se pueden agrupar en tres categorías basadas en la relación con el sitio web, las mismas son:

9.1.1 Solo WCMS o forklift

El objetivo en este caso no es modificar el sitio web. El diseño, contenido y la arquitectura de la información serán iguales, la diferencia radicará en que el WCMS será cambiado por otro o el sitio web estático se migrara por primera vez a un WCMS.
Para este tipo de implementaciones, el sitio web actual es el modelo para el nuevo sitio.

9.1.2 WCMS más reemplazo de presentación o reorganización

Esta es una extensión de la implementación antes mencionada. Aplica para casos donde es necesaria una nueva organización del sitio.
Algunos aspectos del sitio web se verán modificados, pero dichos cambios se limitan a los estilos y edición de contenido. Por lo tanto el sitio web actual es todavía relevante como un modelo.

9.1.3 Construcción completa

En estos casos, el sitio web es reconstruido por completo. El mismo tendrá un nuevo diseño, contenido, arquitectura de la información y nuevas funcionalidades (puede permanecer parte del anterior sitio).
La implementación a menudo viene luego de una gran estrategia de contenido y fase de planificación de la experiencia del usuario. Claramente el sitio web existente es irrelevante como modelo del nuevo sitio.
Previamente a la etapa de implementación (pre implementación), hay varias etapas que deben llevarse a cabo las cuales se detallan a continuación.

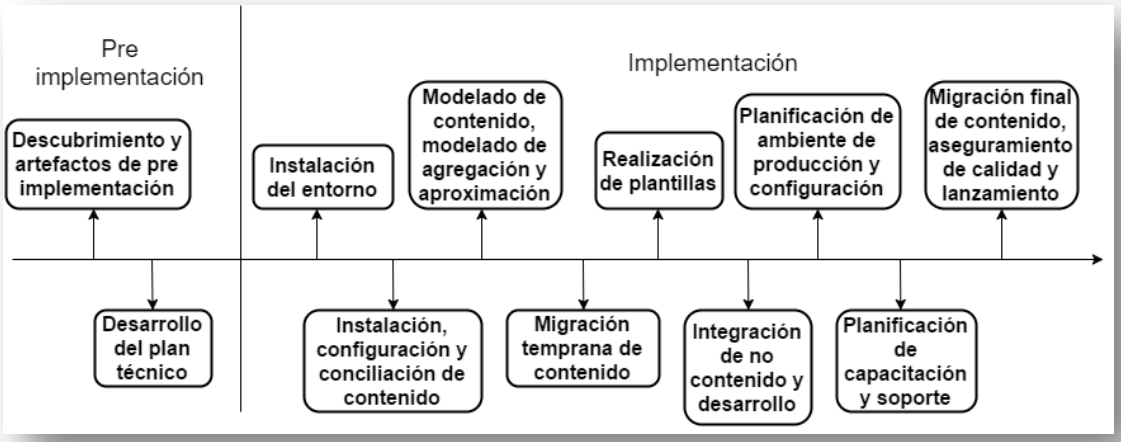


Figura 75 – Línea de tiempo de las actividades a llevar a cabo en un proyecto WCMS. Fuente: Autoría Propia.

9.2 Pre implementación

9.2.1 Descubrimiento y artefactos de pre implementación

La cantidad de documentación necesaria depende en gran medida del nivel de cambio planeado para el sitio web. Para la implementación *forklift,* existe la posibilidad de usar el sitio web actual como el documento de requerimientos de facto.

Cuando solamente se cambia el WCMS en un sitio web ya existente, el objetivo del proyecto puede ser simplemente hacer que las cosas funcionen como lo hacían anteriormente. Será suficiente que el equipo comprenda los trabajos internos a realizar y tener disponible a un integrante del equipo de autores para evacuar dudas.

En el caso que el sitio web sea cambiado de apariencia o reorganizado, se deben tener algunas precauciones con respecto a cómo estos afectarán al WCMS. Si los cambios se limitan solamente al diseño y al contenido y ninguno de estos cambios requieren de una modificación al modelo de datos, entonces no será relevante el esfuerzo requerido para el desarrollo.

Para una construcción completa, se requiere de documentación básica para llevar a cabo una buena implementación. Como mínimo, el equipo de desarrollo necesitará los siguientes documentos:

- Conjunto de *wireframes* que muestren el diseño de cada componente principal, incluyendo los elementos más relevantes. Un *wireframe* -Figura 76- es una imagen de cómo se vería el futuro sitio en construcción.
- Un conjunto de requerimientos funcionales (o anotaciones adjuntas a los *wireframes*) que expliquen como las funcionalidades no visuales deberían comportarse, en particular las relacionadas a la navegación y al contexto en el entorno.
- Un mapa del sitio web mostrando a grandes rasgos como el contenido está organizado.

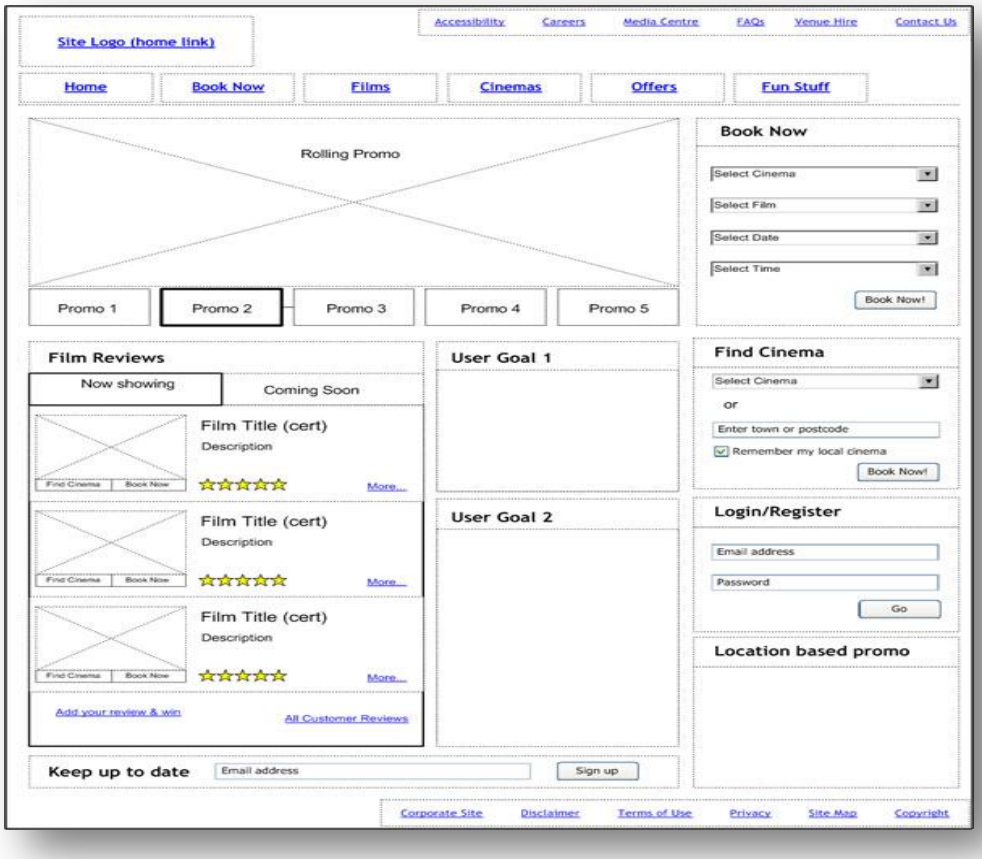


Figura 76 – Wireframe. Fuente: [52].

Dependiendo del alcance de responsabilidades del equipo de desarrollo, también se podría necesitar documentación como:

- Un diseño completo del renderizado con todos los archivos de arte correspondientes.
- Archivos HTML/CSS de la implementación de diseño.

En la actualidad, el desarrollo *front end* se ha convertido en una disciplina en sí misma. Anteriormente las implementaciones se solían llevar a cabo con un conjunto reducido de habilidades de desarrollo. Sin embargo, con el advenimiento del diseño responsivo y el crecimiento del uso de tecnologías del lado del cliente, el desarrollo *front end* es a menudo encargado a un equipo en particular.
Además, en muchos proyectos es común que el desarrollo *front end* sea completado antes que la implementación del WCMS comience.
La fase de descubrimiento de un proyecto WCMS es normalmente gestionada por un equipo de especialistas en contenido o de experiencia de usuario, quienes trabajan con la organización para determinar las necesidades del proyecto.

9.2.2 Desarrollo del plan técnico

El equipo de desarrollo necesitará revisar los artefactos de pre implementación y obtener planes técnicos para todo lo contenido en ellos.

Aunque resulte tentador tratar de planificar un sitio desde arriba hacia abajo, es mejor comenzar de abajo hacia arriba. Esto significa paginar a través del conjunto de *wireframes* (y componentes) y hacer varias consultas acerca de cada uno. Estas preguntas deben ser contestadas de forma previa al desarrollo. En algunos casos el desarrollador escribirá un plan formal de implementación, en otros casos, los *wireframes* son simplemente revisados en una reunión para asegurar el entendimiento de los mismos y confirmar que todo lo especificado es implementable.

Para cada componente se realizan preguntas de diversos aspectos, tanto de su usabilidad a través del sitio, como de su contenido, sus características visuales, propiedades de todo tipo (versionado, idioma, seguridad) entre varias otras cuestiones. Las direcciones de las conversaciones tienen igual importancia a las respuestas conseguidas y a menudo revelan motivaciones o aspectos que no habían sido tenidos en cuenta previamente. Además las entrevistas ayudan a dar contexto, lo cual en el futuro brindará un soporte a los desarrolladores cuando tengan que tomar decisiones de implementación sobre desarrollos más intrincados.

A medida que se despejan las dudas de los *wireframes* uno por uno, surgen preguntas más generales. En particular luego de que cada interfaz haya sido revisada, surgen preguntas sobre el modelo de datos, propiedades específicas de los componentes y definición de WCMS, entre otras.
Todas las respuestas forman parte del plan técnico, el cual guiará el proceso de implementación, así como su alcance y su presupuestación.
Es importante resaltar que las respuestas obtenidas previamente no son universales, sino que están enmarcadas en el contexto de una combinación entre la organización en cuestión, el equipo, y el plan a largo plazo del sitio web a desarrollar.
El equipo en este momento debe tomar decisiones importantes en cuanto a la organización del proyecto. Algunas de las más relevantes son:

- Conocer el presupuesto disponible.
- Conocer la experiencia y curva de aprendizaje de los autores con respecto a conceptos técnicos.
- Estimar la proporción del presupuesto destinada para cada funcionalidad a realizar.
- Determinar el tiempo de uso del proyecto a realizar (permanente o temporal).
- Definir el plan de desarrollo.

9.3 Proceso de implementación

9.3.1 Instalación del entorno

En la mayoría de los casos los desarrolladores trabajarán en sus estaciones de trabajo locales. Subirán su código a un repositorio central, el cual es una plataforma que administra el código fuente –sus siglas son SCM - como pueden ser Git o Subversion.
Varios desarrolladores pueden subir su código, el cual es combinado y desplegado en un servidor de integración para revisación y prueba. Continúan un ciclo que consiste en desarrollar nuevas funcionalidades, subir su código al repositorio central y descargar el código subido por los demás desarrolladores para tener sus estaciones de trabajo locales actualizadas.
El proceso de desplegado del código en los servidores se conoce generalmente como "*building*". Es llevado a cabo usualmente por herramientas llamadas "*building servers*", las cuales consisten en software ejecutando en el servidor de integración que monitorea el repositorio. Cuando detecta que se ha subido código nuevo, lanza un proceso que comprueba el código y lleva a cabo las tareas necesarias para que se termine ejecutando en el servidor – compila el código, borra archivos fuente, copia el código al directorio del servidor web, agrega archivos de licencia, etc.-.

Además del servidor de integración, a menudo se utiliza un servidor de prueba para proveer un ambiente más estable.
Mientras el sitio web es construido en el servidor de integración, el código nuevo es desplegado en el servidor de prueba con menos frecuencia para mantener un ambiente medianamente estable. El servidor de prueba usualmente tiene un *build server* propio pero es activado manualmente o conectado a una rama distinta del repositorio donde el código es actualizado con menos frecuencia.

9.3.2 Instalación, configuración y conciliación de contenido

Una vez que todos los ambientes han sido creados, se procede a la instalación y configuración del WCMS. En muchos casos consiste en un doble clic y en otros casos se requiere del despliegue de archivos a la raíz del servidor web y utilizar un instalador.
Una vez que se completa la instalación, el sitio web resultante (WCMS) tendrá que ser comprobado dentro del SCM, desplegado para su integración y luego comprobado por los otros desarrolladores.
La reconciliación de la instalación y su contenido entre todos los desarrolladores que trabajan en el proyecto puede resultar engorrosa. Una de las cuestiones principales es como se administrará la base de datos que potencia a la mayoría de los WCMSs, si cada usuario debe mantener su propia copia, o si todos los desarrolladores deben interactuar con la base de datos central. La misma disyuntiva se debe determinar para las bases de datos que serán utilizadas como los servidores encargados de la realización de la integración y prueba.
En cuanto al proceso de reconciliación de contenido es un punto muy importante para los desarrolladores. Algunos sistemas han desarrollado tecnología para la migración de cambios en el contenido entre ambientes, en otros sistemas esto debe realizarse manualmente. Para otros WCMS, es necesaria la utilización de aplicaciones externas que se especializan en resolver este tipo de problemas. Un ejemplo claro de esto es la herramienta Unicorn utilizada en Sitecore para resolver el problema del ambiente colaborativo.

9.3.3 Modelado de contenido, modelado de agregación y aproximación

A esta altura del proyecto deben ser creados los tipos de contenidos para el modelo requerido.
Los tipos deben ser definidos y las propiedades agregadas, además de sus correspondientes reglas de validación.
Además del modelo para cada tipo, se deben definir las relaciones entre tipos y objetos. Esto puede implicar la creación de contenido que luego no será usado, pero para el momento presente es de utilidad para poder continuar con el desarrollo.
Hay también otras tareas de modelado, que se describen a continuación:

- **Definir permisos a agregaciones de contenido**
  Se restringe, por ejemplo, que un elemento solo puede tener hijos de un determinado tipo.
- **Refinamiento de la interfaz editorial**
  La interfaz editorial necesita ser considerada desde el punto de vista de la experiencia del usuario.
  Algunos posibles puntos a considerar pueden ser:
  - Ítems etiquetados apropiadamente.
  - Los textos resultan de ayuda.
  - Elementos innecesarios en la interfaz han sido eliminados.
  - Opciones avanzadas ocultas para usuarios sin conocimiento o que no deberían tener acceso.

- **Definición de permisos**
  Requiere de una mínima asignación de permisos, variando los permisos a los grupos correspondientes según el contenido.

Cuando el modelado del contenido esté completo, la estructura básica estaría disponible y debería ser resiliente. Los datos deberían ser validados correctamente, así como las jerarquías y las relaciones, y la interfaz editorial debería ser intuitiva y segura para los autores.

9.3.4 Migración temprana de contenido

A partir de la migración de contenido, es posible que surjan varios problemas.
El contenido migrado en esta etapa no tiene que ser completo ni perfecto, pero necesita ser introducido al sistema.
La migración temprana de contenido puede ser considerada como una "aproximación extendida de contenido". Haciendo una analogía con la filosofía de la integración continua que busca integrar código rápido y de forma seguida, lo mismo busca la "migración continua" con el contenido.
El riesgo de no hacer esto es que el equipo de desarrollo solo trabaje con una parte teórica de contenido y no con el contenido real. El contenido teórico se ha introducido porque es necesario para continuar el desarrollo pero que no es el actual ni el real.
Para evitar males mayores, hay que evitar que los desarrolladores trabajen de forma aislada. Para esto el desarrollador debe verificar si falta información, formatos, relaciones, entre otros posibles problemas.

9.3.5 Realización de Componentes de contenido

A esta altura del proyecto, el ambiente se encuentra configurado, el WCMS instalado, el modelo de contenido creado, y una primera aproximación del mismo está disponible.
El siguiente paso es el de la implementación de los componentes de contenido, donde se generan las presentaciones correspondientes al contenido que se está implementando.
La realización de componentes de contenido se divide en dos maneras:

- **Externa –Pagina-**
  Se refiere a la página que va a contener a los diferentes componentes de contenido.

- **Componente de contenido**
  Se refiere al componente de contenido específico que se quiere presentar.
  Es importante notar que los componentes de contenido que se quiere desplegar son inyectados o anidados dentro de las páginas.

9.3.5.1 Realización de plantillas de páginas

En la mayoría de los casos en la plantillas de página debe funcionar todo el contenido.
También lo más común es que haya solo una plantilla para este caso, teniendo una arquitectura que la haga lo suficientemente flexible para adaptarse a todos los tipos de contenido.

9.3.5.2 Realización de componentes de contenido

Los componentes de contenido no tienen que funcionar para todo tipo de contenido –a diferencia de las plantillas de página- puede funcionar solamente para un tipo. Además por lo general no tiene ninguna dependencia con la plantillas de páginas, como sí ocurre a la inversa.

En algunos equipos, el trabajo de desarrollo es dividido según si el código es correspondiente al *front end* (HTML, CSS) o al *back end*.

9.3.6 Integración de no contenido y desarrollo

Es normal que existan proyectos WCMS que no se limiten solo al WCMS, sino que intervengan también sistemas e información externos, o elementos programados a medida que no interactúen con el contenido –sistemas legados-. Un claro ejemplo de lo descrito es el de un banco y la utilización de una aplicación a medida para los préstamos.
La habilidad para conseguir acoplar funcionalidades no concernientes al WCMS es variable. Algunas alternativas son más flexibles que otras al respecto.
Un desarrollador siempre tiene la posibilidad escribir código por fuera del sistema y de ejecutar una aplicación o acceder a archivos sin invocar el WCMS, pero varias funcionalidades relacionadas al WCMS se perderían. La mejor alternativa es que el código pueda ejecutar dentro del alcance del WCMS y aprovechar las ventajas de la plantillas de código reutilizables, manejo de URLs, manejo de permisos, entre otras.
Sin embargo, puede ocurrir que se debe realizar una migración de un sistema que ejecutaba en un *stack* tecnológico diferente al actual. Para estos casos la aplicación debe ser reescrita para que pueda ser compatible con el nuevo *stack* tecnológico, lo que podría resultar muy costoso, incluso más que la implementación en el WCMS.

9.3.6.1 Integración de contenido

Este tema está relacionado a la integración de contenido ya mencionada. Consiste en el proceso de combinar información de fuentes externas con contenido en una instalación WCMS. Hay muchas maneras de poder llevar esto a cabo, que cuentan con ventajas y desventajas, tomando la opción que mejor convenga según los requisitos del proyecto (donde se encuentra la información, acceso a la información, latencia y velocidad de acceso, etc.).

9.3.7 Planificación de ambiente de producción y configuración

Durante el proceso de desarrollo, el ambiente de producción necesita ser planificado, creado y probado. También el sitio web en desarrollo debería ser desplegado.

Surgen cuestiones básicas que deben ser contestadas, sobre localizaciones físicas/lógicas, relaciones del cliente con el ambiente -tanto técnica como administrativamente-, así como los parámetros técnicos del ambiente.

9.3.7.1 Modelos de alojamiento

- **Alojamiento en las propias instalaciones del cliente**
  Se lleva a cabo en el data center de la propia cliente.
- **Alojamiento en un tercero bajo control de la cliente**
  La organización crea una cuenta de alojamiento en una plataforma (como Microsoft Azure [53] o Amazon Web Services [54]) y administra el ambiente.
- **Alojamiento en un tercero bajo control externo**
  Igual al caso anterior con la diferencia que el cliente cede el control del alojamiento a otra organización. Algunas empresas que ofrecen estos servicios son Rackspace [55], 3sharecorp [56] y Xumak [57].

9.3.7.2 Diseño del ambiente de alojamiento

- **Tolerancia a fallas y redundancia**
  La redundancia perfecta y sin problemas es claramente la situación ideal, pero rara vez es totalmente aplicable y a menudo es costoso.
- **Recuperación de fallos y desastres**
  En caso de presentarse fallas, se debe determinar cómo restaurar el sistema. Se presentan las alternativas de restaurar el ambiente, una es a través de un respaldo y la segunda es mantener un segundo ambiente con una versión del contenido.
- **Performance**
  Se deben evaluar el tráfico que manejará el sitio, las pruebas de carga realizadas, la rapidez en la escalabilidad.
- **Seguridad y acceso**
  Se debe determinar quién tiene acceso al servidor, como desplegar el código nuevo en el sitio web, así como quien puede aprobar dichos despliegues.

Luego de que el ambiente de producción esté listo, resta asegurarse que los despliegues de código se hayan realizado de forma correcta y se haya verificado que el sitio sea funcional.

9.3.8 Planificación de capacitación y soporte

Tanto los autores como los administradores deben ser capacitados para operar en el nuevo WCMS. Existen dos tipos diferentes de entrenamiento:

- **Capacitación WCMS**
  Es la capacitación en el WCMS específico, generalmente brindada por el vendedor del WCMS.
- **Capacitación de implementación**
  Es la capacitación concerniente al sitio web desarrollado. Involucra el entendimiento de como el WCMS está relacionado a los requisitos, así como el entendimiento de los conceptos y estructuras que puedan existir en el sitio. Esta capacitación debe ser dada por alguien familiarizado con la implementación, ya que el vendedor no tiene conocimiento de cómo este producto fue desarrollado.

9.3.9 Migración final de contenido, aseguramiento de calidad y lanzamiento

En esta etapa se debe introducir el contenido que permanecerá en el sistema. El mismo debe tener asegurada su calidad y eventualmente puede ser editado considerablemente.
En cuanto al lanzamiento del sitio web, el mismo puede tomar el lugar de un sitio web ya existente en un ambiente de alojamiento ya utilizado o puede ser desplegado en un nuevo ambiente. Para la primera opción hay que bajar el anterior sitio, realizar una prueba de instalación y regresión para luego si poder liberar el nuevo sitio web.

9.4 Resumen de capítulo

Un proyecto en una plataforma WCMS consiste en dos grandes etapas: la pre implementación y la implementación. En la pre implementación es importante destacar el enfoque hacia componentes de contenido que se realiza, el cual es un concepto clave para el desarrollo en este tipo de plataformas. En el marco de proyectos WCMS se desarrollan componentes de contenido que formarán las páginas del sitio web. Cada página está formada por varios componentes de contenido.

Lo descrito contrasta con el desarrollo web más tradicional, el cual desarrolla las páginas web de forma más “monolítica”, sin dividirla en partes o componentes.

La planificación de los componentes de contenido a realizar en el proyecto se determina en la etapa de pre implementación. Luego en la etapa de implementación, se desarrollan los componentes de contenido planificados para dar paso la construcción del sitio web con dichos componentes de contenido, además de la migración o creación de contenido necesaria para terminar de dar forma al sitio web proyectado.

Capítulo 10: Metodología empírica de desarrollo en un WCMS

En el presente capítulo, se especifica una metodología empírica para proyectos que involucren una plataforma WCMS, fruto de la experiencia del trabajo de Conexio con varios clientes, combinando y tomando las mejores prácticas de cada proceso. El resultado que se describe a continuación intenta plantear el mejor proceso a partir de las fuentes citadas anteriormente. El mismo fue validado previamente por Conexio, que cuenta con una vasta experiencia en el campo de desarrollo de aplicaciones con WCMS utilizando la plataforma AEM.

10.1 Conceptos de la metodología

En esta sección se definirán los conceptos básicos necesarios para comprender la metodología utilizada para un proyecto WCMS. La misma se basa en metodologías agiles utilizando SCRUM-se definirá más adelante-.

10.1.1 Historia de usuario

Las historias de usuario son utilizadas como base para gestión de proyectos y son una descripción de la funcionalidad (valor) para los usuarios o propietarios de negocios. Las historias de usuario no deben especificar la implementación técnica.

Se destaca que no es lo mismo una historia de usuario que un caso de uso. Como se mencionó, las historias de usuario describen al usuario lo que puede ser capaz de realizar, mostrando el valor que agrega la utilización del sistema, sin especificar lo que el sistema debe hacer. Los casos de uso describen interacciones entre el usuario y el sistema, haciendo hincapié en el contexto orientado al usuario.

Desde el punto de vista del WCMS, la realización de un componente de contenido es el equivalente a la implementación de una historia de usuario.

10.1.1.1 Ciclo de vida

Cada historia de usuario pasa por cinco diferentes estados a través de su proceso de desarrollo:

- **A realizar**
  Es la etapa inicial de una historia de usuario, en este momento puede no estar asignada a ningún integrante del equipo.
- **En progreso**
  Cuando un integrante comienza a trabajar en una historia de usuario, la misma pasa automáticamente a este estado, el cual indica que se está trabajando en la historia.
- ***Code Review***
  El desarrollador considera que finalizó la historia de usuario que estaba implementando y envía los cambios realizados al repositorio central del código fuente del proyecto. El código es revisado por otro integrante del equipo o por el arquitecto del proyecto. En caso que los cambios no sean aprobados vuelve al estado “en progreso”.
- **Revisión QA**
  La historia es aprobada y sus cambios son combinados con el código fuente del proyecto, ésta es testeada por un equipo QA independiente –en el mejor caso- al equipo de

desarrollo. En caso que se encuentre un error la historia vuelve al estado “a realizar” con una prioridad alta para que sea solucionado el problema.

- **Finalizada**
  Este estado indica que la historia fue finalizada correctamente.

10.1.2 Scrum

Scrum es un método ágil general [58]. Se concentra en la administración iterativa del desarrollo, y no en enfoques técnicos específicos para la ingeniería de software.
Existen tres fases, la primera es la creación del bosquejo donde se establecen los objetivos generales del proyecto y el diseño de la arquitectura de software.
A esto le sigue una serie de ciclos llamados sprint, donde cada uno desarrolla un incremento del sistema –historias de usuario-. Finalmente, la fase de cierre del proyecto concluye el mismo, completando la documentación requerida, como los marcos de ayuda del sistema y los manuales del usuario. Además el equipo que participó en el proyecto realiza una valoración de las lecciones aprendidas.
La metodología Scrum no especifica como se deben describir las tareas que se deben realizar, nosotros utilizaremos las historias de usuarios para describirlas.

10.1.2.1 Sprint

El corazón de Scrum es el *Sprint*. Es un bloque de tiempo de dos semanas o más durante el cual se crea un incremento de producto, utilizable y potencialmente desplegable.
Cada *sprint* tiene un ciclo de vida -ver Figura 77- y debe definir qué se va a construir, un diseño y un plan flexible que guiará la construcción, el trabajo y el producto resultante.

10.1.2.2 Lista de Pendientes del Sprint

En la lista de pendientes se encuentran todas las historias de usuarios que fueron definidas y aún no han sido desarrolladas.

10.1.2.3 Reunión diaria

La reunión diaria de un sprint es un bloque de tiempo de 15 minutos para que el equipo de desarrollo sincronice sus actividades y cree un plan para las siguientes 24 horas. Esto se lleva a cabo inspeccionando el trabajo avanzado desde la última reunión diaria y haciendo una proyección acerca del trabajo que podría completarse antes de la siguiente

10.1.2.4 Planificación del sprint

A diferencia de los desarrollos tradicionales, en las metodologías ágiles generalmente los tiempos son estimados a través de puntajes asignados a cada tarea. El equipo tiene una capacidad de desarrollo por sprint que se mide a través de estos puntos y son utilizados para determinar cuántas tareas van a ser realizadas en el sprint. Si el equipo de trabajo cuenta con experiencia en otros proyectos, se puede usar dicha información como base para estimar la cantidad inicial de puntos que puede absorber en el comienzo del proyecto. A medida que avanza el proyecto la cantidad de puntos que un equipo de desarrollo puede implementar por sprint puede ser ajustada.

10.1.2.5 Estimación de puntos por historias de usuario (Poker planning)

Cabe aclarar que existen otros métodos de estimación de historias de usuario, pero a los efectos de la metodología planteada, *poker planning* será el utilizado.

La estimación de puntos es realizada en la reunión de planificación del sprint y se estima cada historia de usuario individualmente. Cada integrante del equipo de desarrollo asigna un puntaje, para esto cuentan con cartas cuyos valores son 1, 2, 3, 5, 8, 13 –pueden haber más valores-. Los integrantes desconocen cuantos puntos le asignaron a la historia de usuario sus compañeros de equipo. Luego de que cada integrante del equipo finalizó, se muestran todos los puntajes al mismo tiempo. Si hay un consenso, se le asigna el valor resultante. En caso contrario, se lleva a cabo una evaluación donde generalmente los dos integrantes que seleccionaron la puntuación más baja y más alta ponen en común sus puntos de vista, a partir de esto se vuelve a seleccionar el puntaje. Se repite el proceso hasta llegar a un consenso o en caso contrario se difiere su estimación hasta contar con mayor información sobre lo que se debe hacer.

10.1.2.6 Revisión de Sprint

Al final del *sprint* se lleva a cabo una revisión para inspeccionar el incremento desarrollado, en la misma participa el cliente del proyecto el cual hace una devolución de lo demostrado. A partir de dicha interacción con el cliente pueden surgir nuevas necesidades o correcciones, las cuales deben ser tenidas en cuenta y agregadas la lista de tareas pendientes si fuese necesario.

10.1.2.7 Retrospectiva de Sprint

Es una oportunidad para el Equipo Scrum de inspeccionarse a sí mismo y crear un plan de mejoras que sean abordadas durante el siguiente *sprint*.
La misma tiene lugar después de la revisión de *sprint* y antes de la siguiente reunión de planificación.

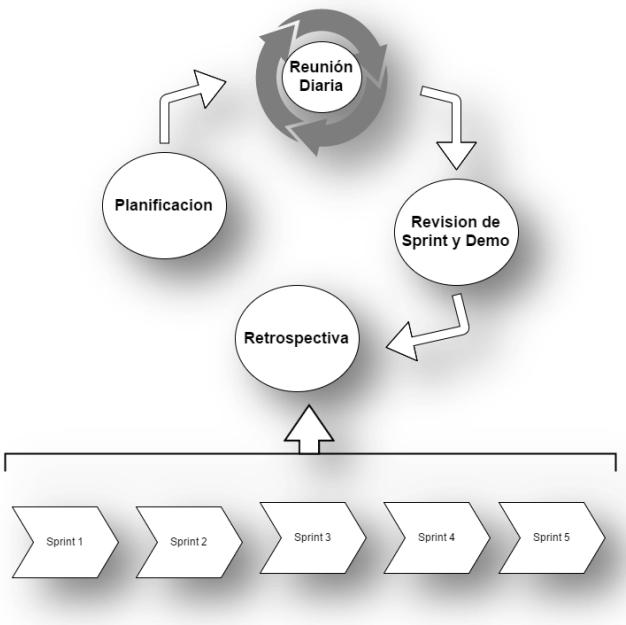


Figura 77 – Esquema de ciclo de *sprints*. Fuente: Autoría Propia.

10.2 Fases de un proyecto

El proceso de desarrollo implica cinco etapas, más la semana cero, que da forma al inicio del proyecto. Las mismas son las fases de descubrimiento, definición, diseño, implementación y despliegue.

En primer lugar, para cada fase, se presenta un diagrama de flujo conteniendo las actividades de cada fase.

Luego se explican los conceptos involucrados en cada diagrama de flujo presentado. Los diagramas utilizados para representar las actividades concernientes a cada fase están basados en los propuestos en la metodología PSP (Personal Software Process) [59]. Lo que más caracteriza este tipo de diagramas es su estructura y sus frases breves, para que sea de fácil comprensión y ejecución de lo descrito en ellos. Cada diagrama está compuesto por cuatro secciones, las cuales son:

- **Propósito**
  Indica el propósito de la fase.
- **Entrada**
  Documentación de entrada para la fase.
- **Paso, actividades y descripción**
  Se especifica el paso, para cada paso puede haber más de una actividad, que tiene una descripción de las sub actividades que la componen. Los diagramas presentados a continuación, se elaboraron haciendo una correspondencia de una actividad por paso.
- **Salida**
  Documentación generada fruto de la ejecución de la fase.

10.2.1 Semana 0

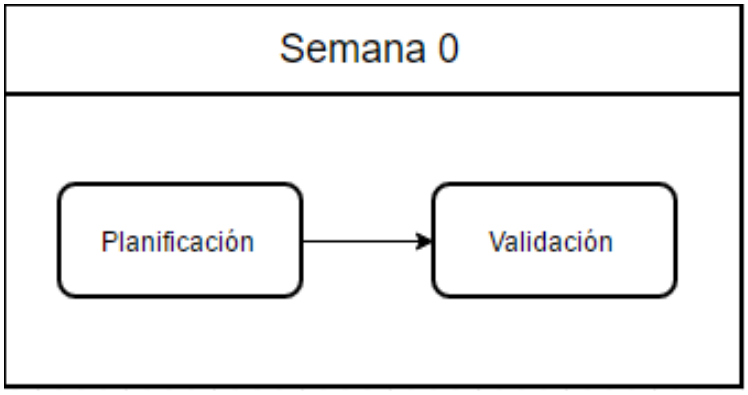


Figura 78 – Esquema de actividades de la Semana 0. Fuente: Autoría Propia.

Propósito		- Asegurar la preparación del inicio del proyecto entre el cliente y el Equipo.
Entrada		No tiene.
Paso	**Actividades**	**Descripción**
1	Planificación	- Determinar herramientas y estrategias para la comunicación- Skype, Google chat, Slack, etc.- entre los equipos de desarrollo del proyecto.
2	Validación	- Validar que el conjunto de herramientas esté listo e instalado.
Salida		No tiene.

10.2.2 Fase de descubrimiento

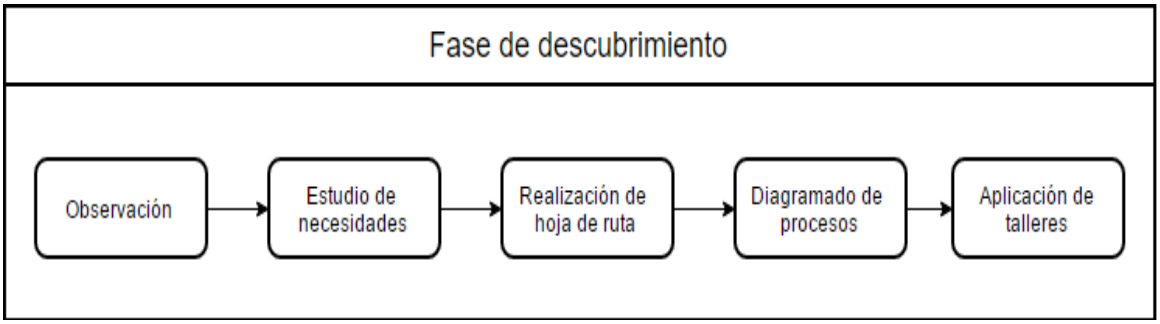


Figura 79 – Esquema de actividades de la fase de descubrimiento. Fuente: Autoría Propia.

Propósito		- Entender, alinear, recolectar, analizar las necesidades del cliente. Validar los requerimientos del mismo.
Entrada		- No tiene.
Paso	**Actividades**	**Descripción**
1	Observación	- Observación etnográfica –prácticas y cultura- de los usuarios.
2	Estudio de necesidades	- Estudio en profundidad del cliente, cubriendo comportamientos, tecnologías, necesidades de lógica de negocio, preferencias y todo lo concerniente al negocio del cliente en cuestión.
3	Realización de hoja de ruta	- Plan estratégico que guía al cliente en los pasos a seguir en los próximos 6 a 18 meses.
4	Diagramado de procesos	- Análisis y posterior re diagramado de procesos los cuales debe indicar los pasos a seguir para llegar de una etapa del proyecto a otra, siendo validados luego por el cliente.
5	Aplicación de talleres	- Se aplican talleres con el cliente los cuales ayudan a clarificar y encontrar ideas innovadoras en áreas de mayor dificultad.
Salida		- Reporte de descubrimiento.

10.2.3 Fase de definición

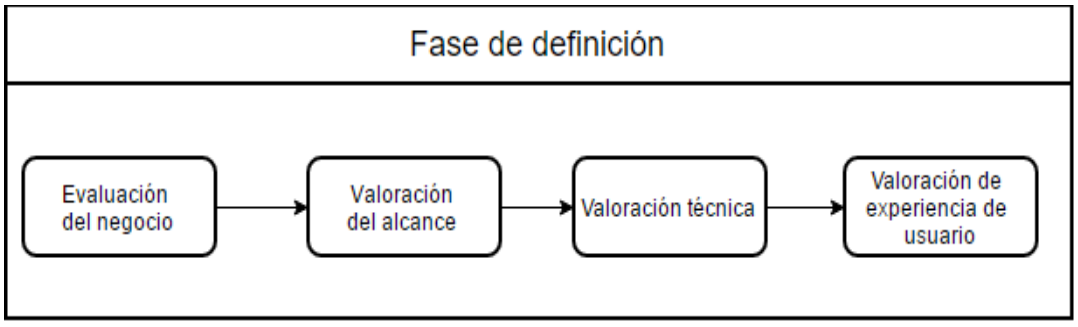


Figura 80 – Esquema de actividades de la fase de definición. Fuente: Autoría Propia.

Propósito	- Adentrarse en los desafíos y problemas del negocio, entender la

		visión y oportunidades del proyecto, y llegar a un acuerdo sobre el alcance.
Entrada		- Reporte de descubrimiento.
Paso	**Actividades**	**Descripción**
1	Evaluación del negocio	- Recolección de información del negocio concerniente al cliente vital para el éxito del proyecto, esta información será utilizada a lo largo de la fase.
2	Valoración del alcance	- Establecer expectativas sobre la línea de tiempo, los recursos y validar o actualizar cualquier estimación generada en instancias previas.
3	Valoración técnica	- Realizar evaluaciones de infraestructura, estrategia de arquitectura y análisis de visión, de rendimiento y escalabilidad para resaltar las consideraciones técnicas de diseño que en última instancia definen el panorama técnico general y las metas de negocio para la solución deseada.
4	Valoración de experiencia de usuario	- Absorber todos los insumos disponibles y revisar los elementos clave en un diálogo abierto para definir con precisión una estrategia centrada en el usuario en términos objetivos.
Salida		- Reporte de resumen de compromiso e informe de definición. - Propuesta de fase de diseño Aproximación de alto nivel para la fase de diseño. Estado de trabajo de la fase de diseño.

10.2.4 Fase de diseño

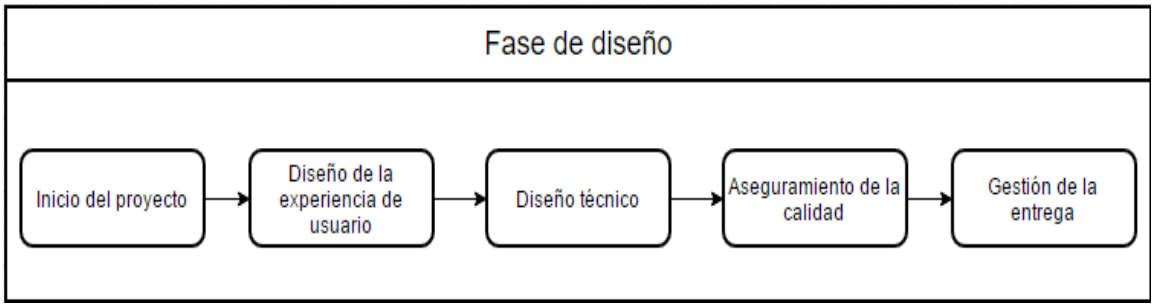


Figura 81 – Esquema de actividades de la fase de diseño. Fuente: Autoría Propia.

Propósito		- Planificar cómo será su arquitectura, diseño, desarrollo y despliegue.
Entrada		Documentación generada en etapas anteriores.
Paso	**Actividades**	**Descripción**
1	Inicio del proyecto	- Planificación para configurar el entorno de colaboración del proyecto, coordinar el lanzamiento, alineación interna del equipo que llevara a cabo el diseño, identificación de dependencias, evaluación de riesgos y planificación de la entrega. - Alineación de resultados clave de la fase de definición, actividades colectivas, entregas e hitos clave.
2	Diseño de la	- Investigación y entrevistas, personajes y escenarios contextuales.

	experiencia de usuario	- Ejecución de *sprints* de diseño de experiencia de usuario: *wireframes* (baja y alta fidelidad), *framework* de diseño (diseño gráfico, interfaz de usuario, patrones de diseño, diseño de interacción, animaciones, coreografía de movimiento y transición de estados) y prototipo de visión. - Colaboración y discusiones de factibilidad. - Prueba de usuario, iteración de diseño, preparación de desarrollo.
3	Diseño técnico	- Arquitectura y planificación tecnológica. - Identificación y ejecución de pruebas técnicas. - Modelado técnico (en conjunción con escenario de contexto y validación de *wireframes*). - Modelado de objetos (basado en escenarios contextuales). - Definición de componentes a partir de los *wireframes* validados. - Creación y refinamiento de historias del usuario. - Estimación de puntos de historias de usuario. - Planificación del entorno.
4	Aseguramiento de la calidad	- Definición de pruebas de aceptación (en conjunto con las historias de usuario). - Planificación de pruebas.
5	Gestión de la entrega	- Plan de liberación. - Proyecto semanal de los *stakeholders* e informes financieros. - Planificación de acercamiento para el desarrollo y desplegado. - Aseguramiento de la calidad de la actividad de integración y sincronización.
Salida		- Conjunto de historias de usuarios vinculadas a todos los activos a los que hacen referencia. - Diseño técnico Modelo de objetos. Diagramas de secuencia. Arquitectura. - Diario de diseño Personas. Escenario de contexto. *Wireframes.* *Frameworks* de diseño. - Prototipo de visión. - Guía de estilos. - Plan de liberación de alto nivel.

10.2.5 Fase de desarrollo

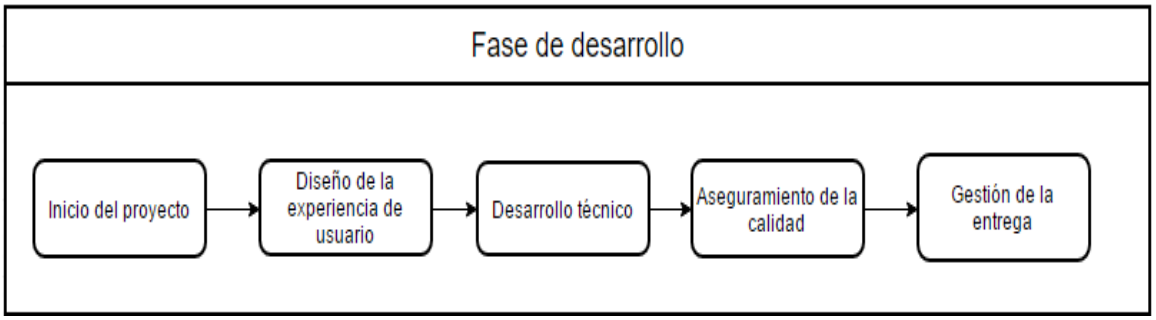


Figura 82 – Esquema de actividades de la fase de desarrollo. Fuente: Autoría Propia.

Propósito		- Desarrollo y prueba del producto diseñado previamente.
Entrada		- Documentación generada en etapas anteriores.
Paso	**Actividades**	**Descripción**
1	Inicio del proyecto	- Configuración de plataforma para seguimiento y gestión de proyectos. Coordinación de inicio, alineación de equipos internos, identificación de dependencias, evaluación de riesgos y planificación de entrega. - Alineación de los principales resultados del diseño, actividades colectivas, entregas e hitos clave.
2	Diseño de la experiencia de usuario	- Soporte a la iteración Representación de la planificación de la iteración. Creación de activos y apariencia de la aplicación. Proveer artefactos de diseño y estilo a equipo de desarrollo, previamente desarrollados en la fase anterior. Supervisión de ajuste y acabado antes del lanzamiento.
3	Desarrollo técnico	- Configuración de integración continua y de creación. - Planificación de *sprint* (descomposición de tareas, asignaciones, estimación, seguimiento diario y reuniones diarias). - Desarrollo iterativo. - Pruebas unitarias. - Identificación, resolución y gestión de defectos y problemas. - Coordinación con equipos de QA y aceptación de usuarios. - Pre y post carga y pruebas de rendimiento. - Transferencia de conocimiento.
4	Aseguramiento de la calidad	- Logística y coordinación de las pruebas de aceptación de los usuarios y del aseguramiento de la calidad. - Pruebas de iteración. - Coordinación, resolución y gestión de defectos.
5	Gestión de la entrega	- Reuniones de equipo diarias de *scrum*, rastreo de tareas, velocidad y asignaciones. - Facilitación semanal de la reunión del proyecto y notas. - Configuración de plataforma para seguimiento y gestión de proyectos y gestión colaborativa. - Facilitación de la planificación de iteraciones.

		- Integración de actividades y sincronización de la de experiencia del usuario y tecnología.
Salida		- Planes de liberación e iteración en evolución (plataforma para seguimiento y gestión de proyectos). - Repositorio de código fuente. - Solución completamente implementada. - Creación de documentación de procesos y scripts de implementación. - Métricas de rendimiento antes y después de la publicación e informes.

10.2.6 Fase de despliegue

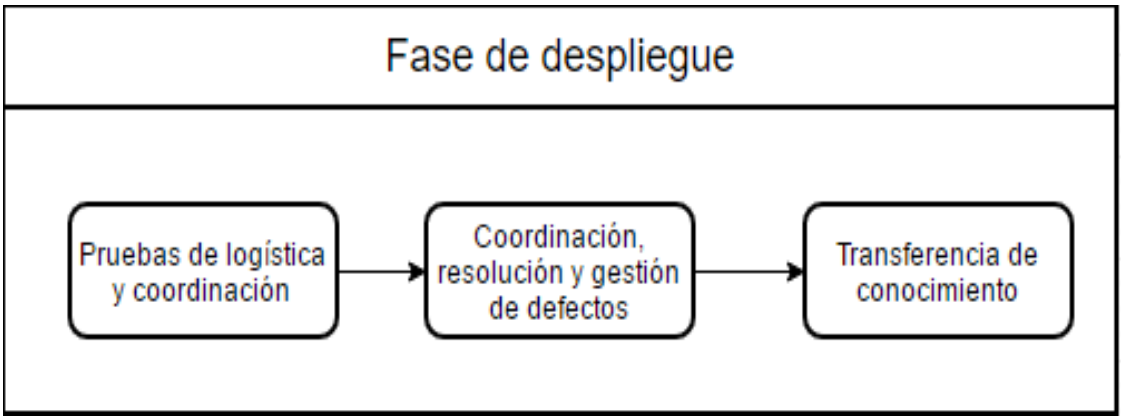


Figura 83 – Esquema de actividades de la fase de despliegue. Fuente: Autoría Propia.

Propósito		- Despliegue en producción de la solución realizada y testeada previamente.
Entrada		- Documentación generada en etapas anteriores.
Paso	**Actividades**	**Descripción**
1	Pruebas de logística y coordinación	- Resolución y proceso de gestión para impulsar el logro de la aprobación formal para la liberación.
2	Coordinación, resolución y gestión de defectos	- Proceso para mantener un control sobre los posibles defectos detectados.
3	Transferencia de conocimiento	- Revisiones de código en curso y/o al final del proyecto. - Base de conocimiento de colaboración del proyecto.

		- Sesiones programadas.
Salida		- Repositorio de código fuente. - Solución completamente implementada. - Métricas de rendimiento antes y después de la publicación e informes.

10.3 Resumen del capítulo

Se planteó una metodología de desarrollo para plataformas WCMS basada en la experiencia de trabajos realizados por Conexio, además del marco teórico con el que se cuenta. Está basada en la metodología de desarrollo ágil Scrum y consta de cinco fases –más la semana 0-, las cuales son: descubrimiento, definición, diseño, desarrollo, despliegue. Cada fase está formada por una cantidad determina de actividades, y a su vez, cada actividad está compuesta por una cantidad de sub actividades.

Parte 4

En esta parte se presentarán las puntos finales del proyecto documentado en este informe, como los son los desafíos a los que nos enfrentamos en este proyecto, el trabajo futuro que ha quedado por fuera del alcance de este proyecto, y por último las conclusiones finales.

Capítulo 11: Dificultades afrontadas durante el proceso

A lo largo del proceso de implementación surgieron numerosas dificultades que se fueron sorteando a lo largo del camino. El primer obstáculo y el cual nos acompañó a lo largo del proyecto fue la curva de aprendizaje que requiere este tipo de plataformas para lograr utilizarlas. Esto se hizo más evidente con Sitecore ya que nuestro cliente Conexio no tenía ningún conocimiento sobre dicha plataforma. Además de la curva de aprendizaje mencionada, se suma la poca documentación que brindan Sitecore y AEM a los desarrolladores, la cual al ser escasa tuvimos que recurrir a la comunidad de desarrolladores utilizando foros en varias ocasiones. Para la comprensión del funcionamiento y uso de Sitecore, que nos permitió estar en condiciones de realizar desarrollos de componentes de contenido, nos requirió un tiempo aproximado de unos cuatro meses.

En cuanto a las dificultades técnicas, el crear un ambiente colaborativo en Sitecore fue uno de los mayores desafíos. Para esto no solo fue necesario comprender la plataforma, sino que también investigar herramientas externas, compararlas y tomar una decisión a la hora de seleccionar una que cumpla con las necesidades del proyecto. Además surgieron diferentes desafíos técnicos a partir de la gran cantidad de tecnologías *back end, front end* y móvil que la implementación del caso de estudio debió utilizar, en particular comprender la interacción entre el WCMS y el código desarrollado fuera del mismo –tanto *back end* como *front end*- resulte en la visualización de los componentes de contenido desarrollados, y por ende, del sitio web. En particular, a nivel de *back end*, al principio del proyecto poseíamos muy poca experiencia en el desarrollo en .NET/ C# -lenguaje necesario para el desarrollo en Sitecore-. El aprendizaje de este lenguaje de programación estuvo incluido dentro del tiempo dedicado a Sitecore –ya mencionado previamente-. Con respecto a la parte móvil, debimos aprender desde cero a utilizar Xamarin, Phonegap y Android –no poseíamos experiencia ni conocimiento previo de estas herramientas. El tiempo que insumió poseer conocimientos para poder realizar desarrollos en conjunto con los WCMS y Android –Xamarin con Sitecore y Phonegap con AEM- ha sido de un mes aproximadamente. También hubo que aprender algunos conceptos concernientes al *front end*, como lo son los estilos web, archivos HTML, el *framework* Angular –utilizado en Phonegap-. Cabe mencionar que las tecnologías *front end* no son la especialidad de los integrantes de este grupo. Sin embargo, la combinación de todos estos desafíos técnicos nos brindó un panorama general de cómo funcionan este tipo de plataformas.

Otro desafío el cual tuvimos que enfrentar en el proyecto fue la negociación con nuestro cliente Conexio en lo que respecta al alcance del caso de estudio. Conexio en un principio planteó la utilización de otros módulos en el caso de estudio que utilizaran analítica de datos –*analytics*-. De esta forma Conexio buscaba generar una implementación que resulte más atractiva para poderla utilizarla para demostraciones a futuros clientes. En este marco se dieron diferentes encuentros y negociaciones que resultaron satisfactorios y finalmente dieron como resultado que dentro del alcance del proyecto no se hayan incluido módulos de analítica de datos, lo cual queda postergado como trabajo futuro a desarrollar –detallado más adelante-.

Siguiendo con el marco de las negociaciones cabe destacar que también tuvimos que afrontar las dificultades a la hora de especificar una metodología de desarrollo consensuada por Conexio y los diferentes proyectos que lleva adelante. Incluso recurrimos a un docente de facultad de ingeniería a modo de consulta sobre el enfoque a trabajar en el tema y que notación utilizar para especificar una metodología para llevar a cabo.

Siguiendo con la temática de la metodología de desarrollo en proyectos WCMS, la poca bibliografía sobre la temática fue una dificultad constante que debimos afrontar. El marco teórico descrito en el informe está basado en un libro reciente –publicado a mediados del 2016- el cual brinda un claro panorama de cómo se afronta un proyecto WCMS

Capítulo 12: Conclusiones Finales

Dentro de los logros obtenidos a través del proyecto, fue posible comprender lo que tienen para ofrecer las diferentes plataformas WCMS, esto incluye la administración de enormes cantidades de contenido y su efectiva distribución a través de diferentes canales –móvil, web, etc.-. Esto además fue posible llevarlo a la práctica en el caso de estudio. Adicionalmente fue posible obtener diferentes análisis de mercado que realizan consultoras de nivel internacional – Gartner y Forrester- donde se comprobó la hipótesis de la que partimos, la cual afirmaba que Sitecore y AEM son los líderes del mercado.

Por otra parte podemos resaltar que junto con nuestro cliente Conexio fue posible definir un caso de estudio lo suficientemente amplio para comprender y finalmente dominar la plataforma Sitecore. A su vez el mismo nos permitió comprender la estrategia que utiliza la plataforma para el desarrollo de aplicaciones móviles. Por otra parte, también logramos comprender la estrategia y tecnologías utilizadas por AEM para el desarrollo tanto de sitios web como aplicaciones móviles. Todo lo mencionado anteriormente nos permitió generar una documentación técnica de suma utilidad para nuestro cliente a la hora de expandir su oferta de desarrollo. Esto se ve reflejado en los diferentes anexos presentados en el proyecto –más de 200 páginas de anexos-, entre los que se encuentran una introducción técnica a cada plataforma –AEM y Sitecore- junto con la solución completa del caso de estudio para cada una de ellas. Conexio podrá hacer usufructo de los mencionados anexos para formar futuros empleados o incluir en su proceso de inducción de personal.

No solo se generó documentación, sino que fue posible implementar un caso de estudio con ambas plataformas que resulta comercialmente atractivo para realizar demostraciones a potenciales clientes de Conexio. De esta forma se logra el objetivo más importante planteado por nuestro cliente, el cual se refiere a su deseo de expandir su oferta de servicios a la plataforma Sitecore. El caso de estudio –el cual abarca desarrollo web y móvil- también tiene el propósito de ejemplarizar el uso de la plataforma y que sirva de inducción para futuros nuevos desarrolladores en Sitecore o AEM que trabajen en Conexio, mejorando así el proceso de inducción de personal.

Asimismo fue posible comparar ambas plataformas –AEM y Sitecore-. Además logramos comprender las distintas estrategias utilizadas para resolver los mismos problemas que ambas enfrentan. Podemos afirmar entonces que para el propósito general ninguna aventaja claramente sobre la otra. En suma, se concluye que ambas plataformas son excelentes herramientas.

A partir de los diferentes proyectos llevados a cabo por Conexio, sumado al marco teórico generado logramos especificar una metodología adaptada al proyecto WCMS. De esta forma, Conexio podrá utilizar dicha documentación, formalizando su proceso de trabajo. En cuanto a la metodología propiamente dicha, creemos que las plataformas WCMS generan una instancia clave de diseño en la cual los involucrados deben dividir parte de los requerimientos funcionales en componentes de contenido.

Creemos que el proyecto mejoró nuestra comprensión global del modelo de negocio y las enormes ventajas que ofrecen los WCMS. Podemos afirmar que hoy entendemos por qué las empresas deciden gastar enormes cantidades de dinero en la implementación de proyectos con este tipo de plataformas, claramente su retribución es mayor. Por otro lado adquirimos conocimientos técnicos en ambas plataformas –Sitecore y AEM- las cuales hoy son nichos no solo para las empresas que ofrecen estos servicios, sino también para los desarrolladores debido a la poca cantidad de personas capaces de utilizarlas.

Por otra parte, logramos aportarle a Conexio el conocimiento y documentación suficiente para expandir sus servicios a Sitecore en el mercado estadounidense. A lo largo del proyecto Conexio utilizó el caso de estudio implementado como *demo* a varios clientes potenciales. A su vez Conexio, con un interés cada vez mayor en trabajar con clientes finales, recibió una especificación completa de una metodología a utilizar en sus futuros proyectos.

Por último el proyecto aporta una comparación de dos plataformas líderes como AEM y Sitecore desde un punto de vista técnico y de experiencia de autor, la cual no nos consta que exista hasta el momento. Esto ya despertó el interés de la comunidad de desarrolladores, tanto de Sitecore como de AEM y recibimos la sugerencia de realizar un blog sobre la temática, el cual llevaremos a cabo como aporte a la comunidad.

Capítulo 13: Trabajo futuro

Analytics constituye una de las áreas que mayor fuerza ha tomado en los últimos tiempos para toda la web. Las plataformas WCMS ofrecen cada vez más funcionalidades para explotar la analítica web, ofreciendo desde evaluaciones sobre la eficacia de los sitios web hasta una amplia gama de datos y estadísticas acerca de la interacción de los usuarios con los mismos. Adicionalmente se cuenta con otras herramientas que incrementan el valor potencial de una organización, cómo lo son herramientas para la creación de campañas publicitarias, para la prueba de distintas combinaciones de contenidos comprobando la más efectiva entre los visitantes del sitio web –pruebas A/B y multivariantes-, para personalizar el contenido desplegado dependiendo del usuario o su región geográfica. Esto permite modificar aspectos que incrementan la explotación comercial de los sitios web, además de mejorar la experiencia del usuario. Es por esto que las plataformas WCMS constituyen la piedra angular de un conjunto de herramientas que utilizan las empresas para cumplir sus objetivos, abriéndose camino como trabajo futuro a profundizar. Es importante mencionar que Sitecore ya brinda como parte del producto ofrecido, una serie de herramientas para un completo manejo de la parte de *marketing*. Sin embargo, la *suite* de *marketing* que ofrece AEM debe adquirirse por separado de la licencia de la plataforma propiamente dicha. Por lo tanto consideramos que a futuro sería de suma utilidad la realización de pruebas de analítica en sitios web, tanto para comprender el funcionamiento de dichas herramientas así como para comprobar las funcionalidades y poder realizar el análisis de los resultados que presenta. Además de las herramientas de *analytics*, en una futura instancia sería relevante la utilización, entre otras, del motor de reglas o del manejo de *workflows*. Lo anterior no ha sido utilizado en el marco de este proyecto por quedar fuera de alcance o por no ser necesario para el caso de estudio implementado.

Aunque fue posible establecer un ambiente colaborativo para Sitecore, que luego de una serie de configuraciones funciona de buena forma (Unicorn), creemos que si se dan las condiciones apropiadas, sería importante poder probar el funcionamiento de la herramienta TDS (*Team Development for Sitecore*) y poder realizar una comparación entre ambas herramientas.

Desde el punto de vista tecnológico, en particular desde Sitecore, quedó pendiente la implementación de *pipelines, hooks*, o *event handlers*. Los mismos son métodos o clases –en el caso de los *hooks*- que pueden extender las clases existentes de Sitecore, agregando o modificando funcionalidades y comportamientos de la propia plataforma.

Si bien la solución del caso de estudio incluyó el desarrollo de aplicaciones móviles, pensamos que en el futuro se podrían profundizar el estudio de las opciones que ofrecen ambas plataformas al respecto; así como las diferentes estrategias que se pueden aplicar cuando la aplicación móvil no posee conexión con el servidor. Teniendo en cuenta que la aplicación móvil desarrollada ha sido de pequeño porte, de cara al futuro se puede considerar la ampliación de la misma, agregando nuevas funcionalidades a las ya provistas –también aplica perfectamente para el sitio web implementado-. También resulta importante tener un seguimiento de la API brindada para el desarrollo móvil para Sitecore, ya que de momento la API ofrecida básicamente realiza la consulta

y obtención de ítems del repositorio de la instancia de Sitecore, no pudiendo por ejemplo, obtener y manipular componentes de contenido –*renderings*-.

En el marco de trabajo de este proyecto, el sitio web desarrollado fue publicado –accesible desde la web- solamente de forma local, o sea, desde el propio equipo que realiza el desarrollo del mismo. Consideramos importante en una próxima instancia, la profundización y puesta en práctica de la publicación de un sitio web a un servidor público desde estas plataformas, en particular desde Sitecore, ya que para AEM, Conexio ya cuenta con experiencia en ese campo. A su vez, el hecho de no haber publicado el sitio web desarrollado en un servidor público, desestimaba la investigación y utilización de la herramienta de *analytics* –ya mencionada previamente- porque no se podrían contar con datos realistas de la interacción de usuarios con el sitio web, y por lo tanto poder sacar conclusiones de valor de dicha herramienta. Relacionado con lo mencionado, también será importante de cara al futuro poder establecer claramente la infraestructura necesaria –servidores, capacidad computacional- para tener un sitio web en estado de producción.

Uno de las principales tareas en el desarrollo con plataformas WCMS es la administración de contenido, y sobre todo el de la migración del mismo, la que puede resultar muy costosa en términos de trabajo y tiempo. En el proyecto realizado, la mayor parte del mismo fue migrado de forma manual, realizando el desarrollo de componentes de contenido. Es importante mencionar que para el caso de estudio realizado, al ser de pequeño porte, no ameritó la investigación y utilización de posibles herramientas y métodos que faciliten la migración de contenido –en particular para AEM y Sitecore-. Sin embargo, pensamos que es muy importante un estudio futuro de lo expuesto anteriormente.

Referencias bibliográficas

[1] http://www.conexiogroup.com/. Último acceso: 23/2/2017.
[2] https://docs.adobe.com. Último acceso: 26/1/2017.
[3] http://www.sitecore.net/. Último acceso: 29/1/2017.
[4] Bob Boiko. *Content Management Bible, second edition.* Wiley Publishing, Inc, 2005.
[5] Deane Baker. Web Content Management. Systems, features, and best prantices. O´Reilly, 2016.
[6] https://www.drupal.org/. Último acceso: 29/1/2017.
[7] http://www-03.ibm.com/software/products/es/category/enterprise-content-management. Último acceso: 29/1/2017.
[8] http://www.opentext.com/what-we-do/products/enterprise-content-management. Último acceso: 29/1/2017.
[9] http://documentum.opentext.com/. Último acceso: 29/1/2017.
[10] https://wordpress.com/. Último acceso: 27/1/2017.
[11] https://techversyssolutions.wordpress.com/tag/wordpress-cms-2/. Último acceso: 31/1/2017.
[12] https://builtwith.com Último acceso: 6/12/2016.
[13] https://www.joomla.org/. Último acceso: 24/2/2017.
[14] https://www.blogger.com/. Último acceso: 24/2/2017.
[15] http://www.gartner.com/. Último acceso: 29/1/2017.
[16] https://go.forrester.com/. Último acceso: 29/1/2017.
[17] Ted Schadler. The Forrester Wave™ : Web Content Management Systems, Q1 2015. Forrester Research, Inc., 2015.
[18] Mark Grannan. The Forrester Wave™ : Web Content Management Systems, Q1 2015. Forrester Research, Inc., 2017.
[19] Mick MacComascaigh, Jim Murphy. *Magic Quadrant for Web Content Management*. Gartner, Inc., 2015.
[20] Mick MacComascaigh, Jim Murphy. *Magic Quadrant for Web Content Management*. Gartner, Inc., 2016.
[21] http://summit.adobe.com/na/. Último acceso: 14/8/2016.
[22] https://docs.adobe.com/docs/en/aem/6-2/deploy.html#What%20is%20AEM?. Último Acceso 21/10/2016.
[23] https://docs.adobe.com/docs/en/cq/5-6-1/exploring/architecture-overview.html. Último acceso 21/10/2016.
[24] https://docs.adobe.com/docs/en/aem/6-2/develop/the-basics.html. Último acceso 21/10/2016.
[25] https://www.osgi.org/developer/architecture/ Último acceso: 27/01/2017.
[26] https://docs.adobe.com/docs/en/spec/jcr/2.0/ Último acceso: 27/12/2016.
[27] http://jackrabbit.apache.org/jcr/index.html Último acceso: 27/12/2016.
[28] https://sling.apache.org/ Último acceso: 27/01/2017.
[29] http://jackrabbit.apache.org/jcr/node-types.html. Último Acceso 21/10/2016.
[30] https://docs.adobe.com/docs/ko/aem/6-1/develop/components.html#Paragraph System. Último acceso 21/10/2016.
[31] https://helpx.adobe.com/aem-forms/6-2/geometrixx-finance-reference-site-walkthrough.html Último acceso: 27/01/2017.
[32] https://docs.adobe.com/docs/en/dev-tools/aem-eclipse.html. Último Acceso 21/10/2016.
[33] https://github.com/Adobe-Marketing-Cloud/aem-project-archetype. Último acceso

21/10/2016.
[34] https://docs.adobe.com/docs/en/aem/6-1/develop/the-basics/clientlibs.html. Último acceso 21/10/2016.
[35] https://docs.adobe.com/docs/en/aem/6-2/develop/platform/sling-resource-merger.html. Último acceso 21/10/2016.
[36] http://stage.docs.phonegap.com/tutorials/develop/02-single-page-architecture/. Último acceso: 6/12/2016.
[37] Herbert Schildt. *C# 4.0: The Complete Reference.* McGraw-Hill. 2010.
[38] Joseph Albahari, Ben Albahari. *C# 5.0 in a nutshell.* O'Reilly Media, Inc. 2012.
[39] https://www.visualstudio.com/es/vs/. Último acceso: 17/10/2016.
[40] https://msdn.microsoft.com/es-es/library/bb545450.aspx. Último acceso: 17/10/2016.
[41] http://logview4net.com/. Último acceso: 17/10/2016.
[42] http://www.w3schools.com/asp/razor_intro.asp. Último acceso: 17/10/2016.
[43] http://www.glass.lu/. Último acceso: 1/2/2017.
[44] http://vsplugins.sitecore.net/. Último acceso: 17/10/2016.
[45] http://www.teamdevelopmentforsitecore.com/. Último acceso: 5/2/2017.
[46] http://docs.sitecoreship.apiary.io/ Último acceso: 01/02/2017.
[47] https://github.com/adoprog/Sitecore-Courier Último acceso: 01/02/2017.
[48] https://github.com/kamsar/Unicorn Último acceso: 01/02/2017.
[49] https://www.xamarin.com/ Último acceso: 01/02/2017.
[50] John West. *Professional Sitecore Development.* John Wiley & Sons, Inc, 2012.
[51] https://www.trustradius.com/compare-products/adobe-experience-manager-vs-sitecore-web-content-management Último acceso: 01/02/2017.
[52] http://www.experienceux.co.uk/faqs/what-is-wireframing/. Último acceso: 30/1/2017.
[53] https://azure.microsoft.com/es-es/ Último acceso: 10/02/2017.
[54] https://aws.amazon.com/es/ Último acceso: 10/02/2017.
[55] https://www.rackspace.com/ Último acceso: 10/02/2017.
[56] http://www.3sharecorp.com Último acceso: 10/02/2017.
[57] http://www.xumak.com Último acceso: 10/02/2017.
[58] Ian Somervile. *Software engineering, 9th edition.* Pearson, 2011.
[59] Watts S. Humphrey. *The Personal Software Process^{SM} (PSP^{SM})*. Carnegie Mellon University, 2000.

Anexo 1 Contexto

Índice

1 Datos y contenido

Las computadoras fueron construidas para procesar datos. Los datos consisten en pequeños fragmentos de información con parte de su significado humano extraído, muchas veces carecen del mismo. Como los datos, el contenido también es información, pero es muy importante su significado humano y contexto.
En términos resumidos el contenido es información producida por un proceso editorial y tiene como único cometido ser consumido por personas vía publicación.
Los WCMS tienen el gran desafío de manejar ese contenido.
En la actualidad el principal uso de las computadoras es como un procesador de datos. En contraparte, la mayoría de los usuarios buscan que las computadoras analicen y manejen enormes cantidades de información -no fracciones- y sean capaces de enviar la información que ellos seleccionen en el menor tiempo posible [1].

2 Marco histórico de la evolución del contenido

En los últimos 30 años se ha dado una revolución en materia de información. Lo que impulsó esta revolución fue la posibilidad de crear medios digitales como imágenes, sonidos y videos en las computadoras, así como también la posibilidad de acceder a ellos desde diferentes dispositivos. A lo que se adicionó la disposición para el consumo masivo de distintos artefactos de almacenamiento de alta capacidad y baratos [1].
A partir de estos avances se potenció el rápido desarrollo de la industria de la multimedia y más aún el uso de la multimedia en diferentes industrias tradicionales. Por primera vez se podía generar contenido y distribuirlo de forma masiva y barata con los CD-ROMs. Pronto la industria se despegó y los CD-ROMs proliferaron, repartiendo contenido que variaba desde grandes enciclopedias, catálogo o juegos. Las computadoras comenzaron a emerger como un reemplazo a los canales tradicionales para hacer llegar información a los usuarios como libros, televisión y radio. Estos canales transmitían contenido y no datos. La revolución había comenzado [1].
La web terminó de consolidar lo que las industrias de contenido multimedia habían comenzado. En la actualidad, buscar el contenido de forma *online* no solo es posible, sino que es lo que la mayoría de los usuarios eligen por encima de la utilización de otros canales. Aunque las necesidades y expectativas de los usuarios cambiaron, la esencia de las computadoras no. Hace 30 años las personas llegaban a las computadoras con datos de entrada, procesos y esperaban datos de salida. Hoy los usuarios utilizan las computadoras para **consumir contenido**. A pesar de esto la base tecnológica de la computación sigue siendo la misma. Se basa en la idea de que se puede reducir cualquier problema a un conjunto de simples instrucciones que trabajen con fragmentos de datos estructurados y discretos [1].
El contenido y los datos son diferentes, pero eso no significa que no exista interacción entre ellos. Innumerables transiciones de uno a otro ocurren todos los días. Sin embargo desde el punto de vista de los sistemas computacionales el contenido no existe, solo existen los datos [1].

3 Clasificación de tipos de manejo de contenido

Los cuatro grandes tipos de WCMS pueden ser separados como [2]:

3.1 WCM (WEB Content Management)

Brinda la gestión de contenido, para en primera instancia ser distribuido masivamente a través de un sitio web. Se destaca en separar el contenido entre la presentación y la publicación en múltiples canales. Es el tipo de gestor de contenido más utilizado hoy en el mercado. Por citar algunos ejemplos: WordPress, Drupal [3], AEM, Sitecore.

3.2 ECM (Enterprise Content Management)

El término contenido empresarial es a menudo usado para hacer referencia a contenido interno que usualmente no es publicado fuera de la organización.
Consiste en el manejo de contenido de negocio general, no necesariamente dirigido a un público masivo o para consumo interno de una empresa, como por ejemplo hojas de vida de empleados, reportes de incidentes, memorándums, etc. Este tipo de WCMS ha sido tradicionalmente mejor conocido como un administrador de documentos. Algunos ejemplos de ECMs son: IBM [4], OpenText [5], EMC [6].

3.3 DAM (Digital Asset Management)

Realizan la administración y manipulación de archivos digitales como imágenes, audio, y video para que estos sean utilizados en otros medios.
Mientras casi cualquier sistema administrador de contenido puede almacenar archivos de imagen y video, los sistemas DAM va un paso más allá proveyendo herramientas para crear y transformar archivos digitales. Las imágenes pueden ser redimensionadas y los videos pueden ser ajustados y editado directamente en el sistema. Por lo tanto, las características principales de un sistema DAM son estrechamente similares a las de un sistema ECM. De hecho, muchos sistemas DAM son vendidos simplemente como complementos a sistemas ECM. Asimismo muchas plataformas WCMS tienen incorporado un sistema DAM.

3.4 Manejo de archivos (Records Management)

Realiza la administración de información transaccional creada como un producto de operaciones de negocio (por ejemplo registros de ventas, registro de accesos, etc.).

En conclusión, la línea entre los tipos de CMS citados es difusa. Un sistema DAM es a menudo usado para proveer contenido a un sitio web a través de una integración con un WCMS. Además, algunos sistemas ECM tienen sistemas por los cuales pueden publicar algo de su información en la web.
Por lo tanto, las definiciones brindadas no son estrictas y muchos sistemas son reconocidos sólo a través de su intención de uso y de su percepción en la industria. Por ejemplo, Drupal es un WCMS muy conocido, pero hay sin duda organizaciones que lo usan para administrar contenido interno, empresarial. A la inversa, Documentum es un sistema ECM, pero algunas organizaciones lo utilizan para entregar todo o parte de su información a sus sitios web.
Adicionalmente a los tipos citados previamente, hay otros tipos de CMS para los cuales los límites entre ellos es aún más difuso que en los anteriores [2]:

3.5 CCM (Component Content Management)

Administración de contenido de muy poca granularidad (oraciones o párrafos) usado para ensamblar documentación o contenido altamente técnico.

3.6 LMS (Learning Management System)

Administración de recursos de aprendizaje y de interacción estudiantil. La mayoría de colegios y universidades administra la interacción estudiantil y la participación en clase a través de un LMS.

3.7 Portales

Administración, agregación y presentación de información de múltiples orígenes en un sistema unificado.

4 Definición WCMS

Un WCMS es una aplicación que permite a los usuarios obtener control sobre la creación y distribución de contenido y funcionalidades [1]. Adicionalmente provee cierto nivel de automatización para las tareas requeridas para la administración de contenido [2].

Un WCMS es usualmente un *software* basado en el servidor web y multiusuario, que interactúa con el contenido almacenándolo en un repositorio. El repositorio puede estar localizado en el mismo servidor –como parte del mismo *software*-, o estar ubicado en forma completa en una instalación de almacenamiento externa.

Los WCMS son plataformas extensas y complejas con múltiples actores involucrados. Están compuestos por varias partes, como los son la interfaz de edición, el repositorio, los mecanismos de publicación, entre otros. Para quien use este tipo de sistemas y no posea conocimientos técnicos, las partes mencionadas previamente son vistas como un todo, de forma monolítica: “el WCMS”. Sin embargo, podrían estar separadas e inclusive son partes autónomas de por sí en el sistema.

A pesar que la utilización primaria de los WCMS es la creación de sitios web de gran porte, el potencial de los WCMS para ayudar a mejorar a las organizaciones va más allá de la web. En el repositorio se persisten y organizan enormes volúmenes de información. Partiendo de ese repositorio, las organizaciones serían capaces de producir publicaciones de dicho contenido a un sitio web o a cualquier otro canal posible [1].

En conclusión, un WCMS permite la creación, edición, ejecución de procesos editoriales de contenido, y en última instancia, hacer que el mismo esté disponible para el consumo de los usuarios.

A continuación se puede visualizar –Figura 1- un entorno web para administrar contenido que presenta un WCMS o ambiente de autor –se verá más adelante-, en este caso corresponde a WordPress [7] –uno de los WCMS más utilizados en la actualidad-.

La gran mayoría de los WCMS presentan un formato similar de cómo está presentada la aplicación. En la parte central de la pantalla se provee de un panel administrador, el cual puede contener acceso a distintas funcionalidades y/o mostrar distintas estadísticas o métricas -dependiendo del WCMS-. Sobre la izquierda y barra superior se brindan menús con todas las herramientas que provee el WCMS para la administración del contenido -en todas sus etapas-.

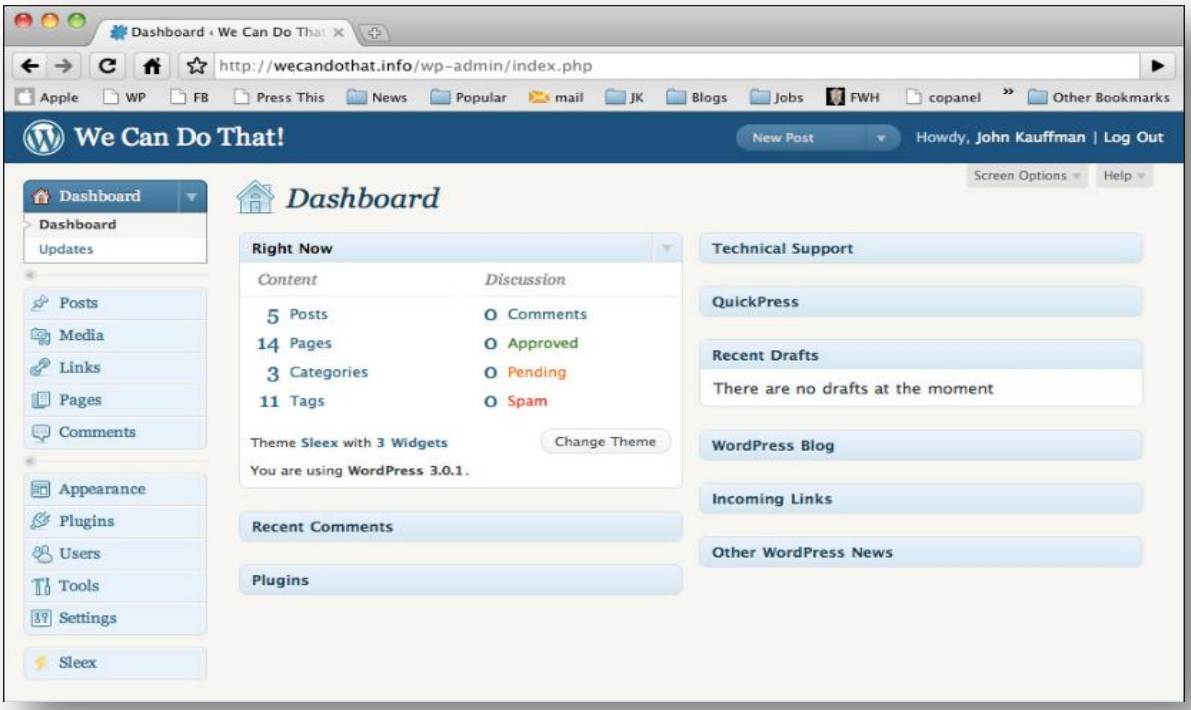


Figura 1 - Toma de pantalla de WCMS WordPress. Fuente: [8]

5 Stack tecnológico de un WCMS

Como toda aplicación, un WCMS es ejecutado sobre un conjunto de software de soporte, bases de datos y lenguajes de programación. Los mismos están implementados en un lenguaje de programación y utilizan un *framework* de almacenamiento específico [2].
El lenguaje y *framework* utilizados influencian sensiblemente el ambiente de alojamiento necesario para que el WCMS pueda ejecutar.
El *stack* tecnológico está compuesto por:

- El WCMS propiamente dicho.
- Un *framework* de programación.
- Un lenguaje de programación.
- Un servidor de base de datos.
- Un servidor web.
- Un sistema operativo.

6 Ciclo de vida del contenido

Uno de los términos más importantes –sino el más importante- de este informe es el del contenido. El mismo tiene un ciclo de vida -en un WCMS- compuesto por cinco etapas -Figura 2-, que van desde su

creación a su eliminación. Se muestra a continuación una figura representando el citado ciclo de vida y la descripción de las etapas que componen el mismo [2].

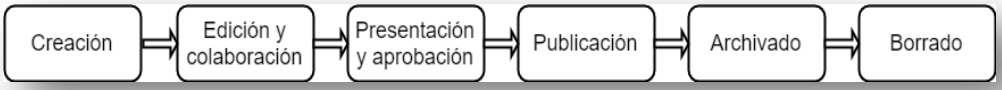


Figura 2 – Ciclo de vida del contenido en un WCMS. Fuente: Autoría Propia, en base a [2].

- **Creación**
  El contenido es iniciado en el WCMS. No está completo, pero existe como un objeto de contenido gestionable.

- **Edición y colaboración**
  El contenido es activamente editado, tanto por un solo editor –autor- como por parte de un equipo. El contenido todavía no es visible en esta etapa.

- **Presentación y aprobación**
  El contenido creado y editado es presentado a una o más aprobaciones. Continúa sin ser visible por los usuarios.

- **Publicación**
  Se publica en el sitio web. En este estado el contenido es visible para el consumo de los usuarios.

- **Archivado**
  El contenido es removido del acceso al público pero no es borrado. Usualmente no va a ser visible para los usuarios de aquí en adelante.

- **Borrado**
  El contenido es borrado de forma definitiva del WCMS

Es importante mencionar que las etapas descritas son iterativas y pueden ser aplicadas en simultáneo a distintas versiones del mismo contenido.

7 Gestión y entrega de contenido

La gestión del contenido se refiere a todo lo que le sucede al mismo desde el momento que es creado hasta que es borrado (ciclo de vida). Trata los temas de seguridad, control y eficiencia. Está compuesto de funcionalidades como el modelado de contenido, permisos, versionado y *workflow*, que facilitan la creación del contenido, permiten la colaboración editorial y mantienen el contenido seguro.
La entrega de contenido se refiere a todo lo que le sucede a la versión publicada del mismo. Trata sobre la optimización y el rendimiento, que dependen en gran medida de las capacidades del WCMS.

8 Sistemas acoplados y desacoplados

En un sistema acoplado, la gestión y la entrega de contenido ocurren en el mismo servidor (o conjunto de ellos). Los autores gestionan contenido en el mismo sistema en el que los usuarios lo consumen.
Por el contrario, en un sistema desacoplado, la gestión y la entrega se encuentran en ambientes (servidores) distintos. El contenido se gestiona en un ambiente y luego es publicado en otro, independiente. En estos casos, las funciones concernientes a la gestión son a veces referidas al servidor repositorio mientras que la entrega del contenido se realiza en un servidor de publicación o servidor de entrega.
En un principio, la web fue hecha a través de WCMSs desacoplados. En la actualidad se están usando sistemas acoplados. Este cambio surgió por varios motivos, como lo son:

- Necesidad de interactividad en tiempo real.
- Los sitios web se vuelven cada vez más variables.
- Se requiere acceso inmediato al repositorio.
- Pequeños cambios en el entorno de entrega (por ejemplo un comentario o un puntaje), forzaban a una re publicación completa, lo que resultaba problemático.

A través del tiempo se fueron agregando cada vez más funcionalidades dentro del entorno de entrega, hasta que se volvió obvio para los nuevos desarrolladores que el contenido debía ser manejado en el mismo WCMS. Esto terminó por inclinar la utilización de arquitecturas acopladas.

9 Tareas realizadas por un WCMS

Un WCMS permite controlar y mantener trazabilidad del contenido.
Específicamente, un WCMS provee funciones de control básicas, que ayudan a mejoran el nivel de control sobre el contenido y reducen riesgos. Algunas de las funciones provistas son [1]:

9.1 Control de contenido

9.1.1 Permisos

Un WCMS debe administrar los permisos de los diferentes usuarios que acceden al contenido, teniendo en cuenta quién puede ver, cambiar, o eliminar el mismo.

9.1.2 Manejo de estado y workflow

Permite tener control sobre el estado del contenido y sus posibles estados, como lo es saber si el contenido ha sido publicado, si está en proyecto o si ha sido archivado o removido.

9.1.3 Versionado

Se mantiene un control sobre las versiones que se van generando del contenido, con lo cual se puede rápidamente verificar si el contenido ha sido cambiado, cómo era el mismo tiempo atrás, qué diferencia presenta la versión actual con una anterior o si es posible restaurar o republicar una versión anterior.

9.1.4 Manejo de dependencias

Se puede verificar, por ejemplo, qué contenido está siendo usado por otro contenido, como afecta a los demás contenidos si se borra cierto contenido, o saber qué contenido está “huérfano” o en desuso.

9.1.5 Búsqueda y organización

Se puede realizar una búsqueda de una parte específica de un contenido, encontrar todo el contenido que refiere a un término o agrupar y relacionar contenido para que sea más sencillo de manejar.

9.2 Permitir el reuso del contenido

Usar contenido en más de un lugar y en más de una forma aumenta su valor.
La habilidad de reusar contenido es altamente dependiente de la estructura del contenido. La habilidad para estructurar el contenido adecuadamente para un reuso óptimo depende en gran medida en las prestaciones del WCMS usado.

9.3 Permitir automatización del contenido y agregación

Teniendo todo el contenido en una única ubicación hace más fácil su consulta y manipulación.
Si el contenido es estructurado correctamente, se puede manipular para ser mostrado en diferentes formatos, publicarlo en diferentes lugares, y reorganizarlo sobre la marcha para satisfacer las necesidades de los visitantes más eficazmente:

- Se puede permitir a usuarios que consuman el contenido en otros formatos, como PDF o formatos de libro electrónico.
- Se pueden crear listas automáticamente y navegación para el sitio web.
- Se pueden crear múltiples traducciones de contenido para asegurar que se entregue en el lenguaje más apropiado al usuario actual.
- Se puede alterar el contenido que se publica en tiempo real basado en los comportamientos específicos exhibidos por los visitantes.

Un WCMS permite estas cosas estructurando, almacenando, examinando, y proveyendo facilidades de consulta sobre el contenido.

9.4 Mejorar la eficiencia editorial

La capacidad de los editores para crear y editar el contenido de forma rápida y precisa está enormemente afectada por la plataforma utilizada.
La eficiencia del editor se incrementa en un sistema que controla qué tipo de contenidos pueden y no pueden añadir, que herramientas de formato están disponibles para ellos, cómo su contenido es estructurado en la interfaz de edición, cómo es el *workflow* editorial y cómo se gestiona la colaboración. Adicionalmente a lo que le sucede al contenido después de que se publique.
Un buen WCMS permite a los editores publicar más contenido en un marco de tiempo más corto -que aumenta el "rendimiento editorial"-, y para controlar y gestionar el contenido publicado con una menor cantidad de fricción o arrastre sobre su proceso.

10 Tareas no realizadas por un WCMS

A continuación se describen actividades que un WCMS no realiza y que en caso de ser realizadas podría generar problemas o expectativas no satisfechas [2]:

10.1 Crear contenido

Un WCMS simplemente gestiona contenido, no lo crea. Todavía se debe proveer el poder editorial para generar el contenido que se supone va a ser gestionado.

Un WCMS no asegura la calidad del contenido. Aunque puede ofrecer muchas herramientas para minimizar la pobre calidad del contenido desde un punto de vista técnico, un WCMS no puede editar el contenido para asegurarse que tenga sentido y satisfaga las necesidades del público.

10.2 Crear planes de marketing

Aun asumiendo que el contenido está creado de forma consistente y bien gestionado, eso no significa que provea a la organización algún valor.
Un WCMS desconoce sobre *marketing*. Mientras otros sistemas tienen herramientas de *marketing* incluidas, las mismas aun dependen de los humanos para su dirección. El *marketing* efectivo es una práctica humana que involucra una combinación de estética, sociología, psicología, experiencia e intuición. Un WCMS puede hacer que se ejecuten los planes de *marketing* de manera más sencilla y más eficiente, pero esos planes todavía necesitan ser concebidos, creados, y analizados por un humano competente.
Un WCMS no toma el lugar de un equipo creativo que entiende el mercado, los clientes, los competidores, y lo que es necesario para diferenciarse. No hay software que pueda tomar el lugar de una buena estrategia o buen equipo de *marketing* digital.

10.3 Efectivo formato de contenido

Mientras un WCMS puede estructurar contenido y darle formato automáticamente durante la publicación, todavía hay una amplia cantidad de espacio para el editor humano –llamados autores en el ambiente WCMS- para que cometa errores. La mayoría de los WCMS tienen un editor de texto o alguna otra interfaz que permite a los editores dar formato a texto e imágenes. Esto puede llevar a errores como:

- Uso excesivo de letra tipo negrita e itálica.
- Alineamiento inconsistente del contenido.
- Relaciones inconsistente.
- Pobre emplazamiento de imágenes.

10.4 Proveer definiciones de gobernanza

La gobernanza describe el acceso a procesos alrededor del contenido:

- Determinar quién tiene acceso a qué parte.
- Definir procesos o pasos a seguir ante un cierto evento.

Cada WCMS tiene un conjunto de métodos para limitar las acciones que un usuario puede tomar, pero estos límites tienen que ser definidos por adelantado. El WCMS simplemente llevará a cabo lo que la organización dicte que se haga. Estos planes tienen que ser creados a través de la interacción y juicio humano, entonces convertido en permisos y límites de acceso que el WCMS puede hacer cumplir.
La gobernanza es ante todo una disciplina humana. Se determinan los procesos y políticas que los seres humanos acatarán cuando se trabaja con el contenido. El WCMS es sólo un marco de aplicación.

11 Descomposición de los principales sistemas de un WCMS

Las plataformas WCMS se pueden descomponer en 3 sistemas principales que nuclean su funcionamiento, estos tres sistemas son [1]:

- El sistema recolector.
- El sistema de gestión.

- El sistema de publicación.

11.1 Sistema Recolector

El sistema recolector es responsable por todos los procesos que suceden antes de que el contenido esté listo para su publicación. Es el encargado de la transformación de la información cruda en un conjunto organizado y estructurado de componentes. El proceso incluye:

11.1.1 Autoría

La autoría se refiere al proceso de creación de contenido desde cero. Dentro de este proceso se define al autor como alguien que específicamente está encargado de crear contenido para el WCMS, si el propósito de creación de contenido es otro que no sea para el WCMS, se define su contenido como adquirido y no autorizado.
El WCMS debe ayudar al autor a trabajar de forma eficiente y efectiva brindándole las siguientes prestaciones:

- **Ambiente de autor**, el cual brinda un entorno para la creación y administración de contenido (ya sea una aplicación completa, un ambiente web o una extensión al ambiente nativo del autor).
- Una audiencia y un propósito claro para los esfuerzos del autor.
- Funcionalidades de asistencia para incluir información estándar. Por ejemplo los WCMS fácilmente pueden incluir en el contenido la fecha de la creación y el nombre del autor para persistir los esfuerzos del mismo.
- *Workflows*, *status* y control de versión para el contenido que se encuentra en proceso.

Sin importar cuantas herramientas y procesos pueda brindar el WCMS, la autoría es esencialmente una tarea manual. Ningún WCMS le puede decir a un autor que escribir o que resaltar como importante o cuál es la mejor forma de comunicar una idea o un objetivo. Esto lleva a que la autoría sea lenta y costosa.
Lo que termina ocurriendo es que las herramientas que el WCMS brinda en la práctica sirven para mantener el control y la gobernabilidad sobre cómo los autores realizan su trabajo. Muchas de las características brindadas tienen como objetivo hacer que los autores realicen las labores requeridas, en lugar de ayudar a los autores a realizar un trabajo eficiente.

11.1.2 Adquisición

La adquisición es el proceso de recolectar información que originalmente no fue creada para el WCMS. El proceso puede ser parcialmente manual o completamente automático. Generalmente el mayor esfuerzo en este proceso se encuentra en transformar o adaptar dicha información a los estándares del WCMS.
Mientras la información generada a partir de los autores tiene poco volumen pero una gran calidad, la información adquirida se encuentra en gran volumen y con una calidad baja.

11.1.3 Conversión

La información que es creada o adquirida generalmente no se encuentra en el formato o estructura que el sistema de contenido requiere, esta información debe ser adaptada para que cumpla con los estándares definidos por el sistema.
El proceso de conversión generalmente consiste en los siguientes pasos:

- **Stripping**
  Remover y descartar toda la información innecesaria que rodea al contenido como por ejemplo el *footer* y *header* de la página, contenido innecesario o navegación no requerida.

- **Format mapping**
  Cambiar el formato binario del contenido a un formato estándar que soporte el WCMS.

- **Structure mapping**
  Crear la estructura de la información de forma explícita o modificarla según sea necesario.

11.1.4 Agregación

La agregación es el proceso de acercamiento de las diferentes fuentes de información dispares en una única estructura global. La agregación consiste en tres pasos:

11.1.4.1 Control Editorial

Se le agregan estilos, se chequea consistencia y usos.
Básicamente el control editorial es un *framework* el cual limita y guía el trabajo de los autores de contenido. De esta forma la organización establece ciertas normas y prácticas en la edición de contenido para que la información sea consistente.
Las reglas que define el *framework* son:

- **Reglas de calidad**
  Estas reglas aseguran que los autores generen contenido bajo los estándares de calidad que define la organización. Las reglas como la puntuación, el uso de las palabras y la gramática son definidas y controladas en este punto.

- **Reglas de comunicación**
  A partir de establecer cuál es la imagen y audiencia objetivo del contenido, las reglas como la voz del contenido sea pasiva, activa, en primera persona o en tercera así como otras reglas de estilo son definidas dentro de este conjunto.

- **Reglas de consistencia**
  Aseguran que el autor aplique todas las demás reglas a través de toda la base del contenido.

11.1.4.2 Segmentación

La segmentación consiste en el proceso de descomponer el contenido en partes pequeñas y reutilizables. Estas partes pequeñas de contenido que son reutilizables se llamará ***componente.***
Antes de comenzar la segmentación de contenido en componentes, los autores deben tener claro qué tipo de componentes están tratando de crear.

11.1.4.3 Servicios de recolección

La principal tarea que realizan los servicios de recolección es el apoyo en la función de persistir el contenido en el repositorio. Entre las actividades que realizan se encuentran:

- La creación de componentes directamente en el repositorio.
- Cargar en el repositorio componentes previamente creados.
- Exportación o importación de contenido.

11.2 Sistema de Gestión

El sistema de Gestión del WCMS es responsable de la persistencia de los componentes y otro tipo de recursos. Debe permitir mantener la información de todos los recursos que se han recolectado y cuál es su ubicación. Entre otras cosas el sistema administrador debería ser capaz de indicar:

- El contenido persistido en detalle, incluyendo qué tipo de componentes contiene y en qué etapa del ciclo de vida se encuentra cada uno.
- Cómo se están usando los componentes en las publicaciones y cuales no están siendo utilizados.
- Quien ha tenido acceso al contenido y lo ha modificado.

Para poder proveer esta funcionalidad, el sistema de gestión incluye:

- **Repositorio**
  Un lugar donde guardar el contenido.

- **Administración**
  Un sistema administrador para configurar el WCMS

- **Workflow**
  Define entre otros procesos, los paso necesarios que deben realizarse en el contenido para que sea publicado.

- **Conexiones**
  Un conjunto de conexiones (hardware y software) a otros sistemas.

11.2.1 Repositorio

El repositorio es la pieza más importante del sistema de gestión. Básicamente es un conjunto de bases de datos, directorios u otro tipo de estructuras que guardan el contenido y cualquier otro dato asociado con el WCMS. Los componentes y otros recursos del WCMS son persistidos en el repositorio mediante los servicios de recolección –como ilustra la Figura 3-, y los servicios de publicación los extraen. El repositorio puede contener los siguientes tipos de archivos:

- Contenido en bases de datos y archivo.
- Archivos de configuración y control.

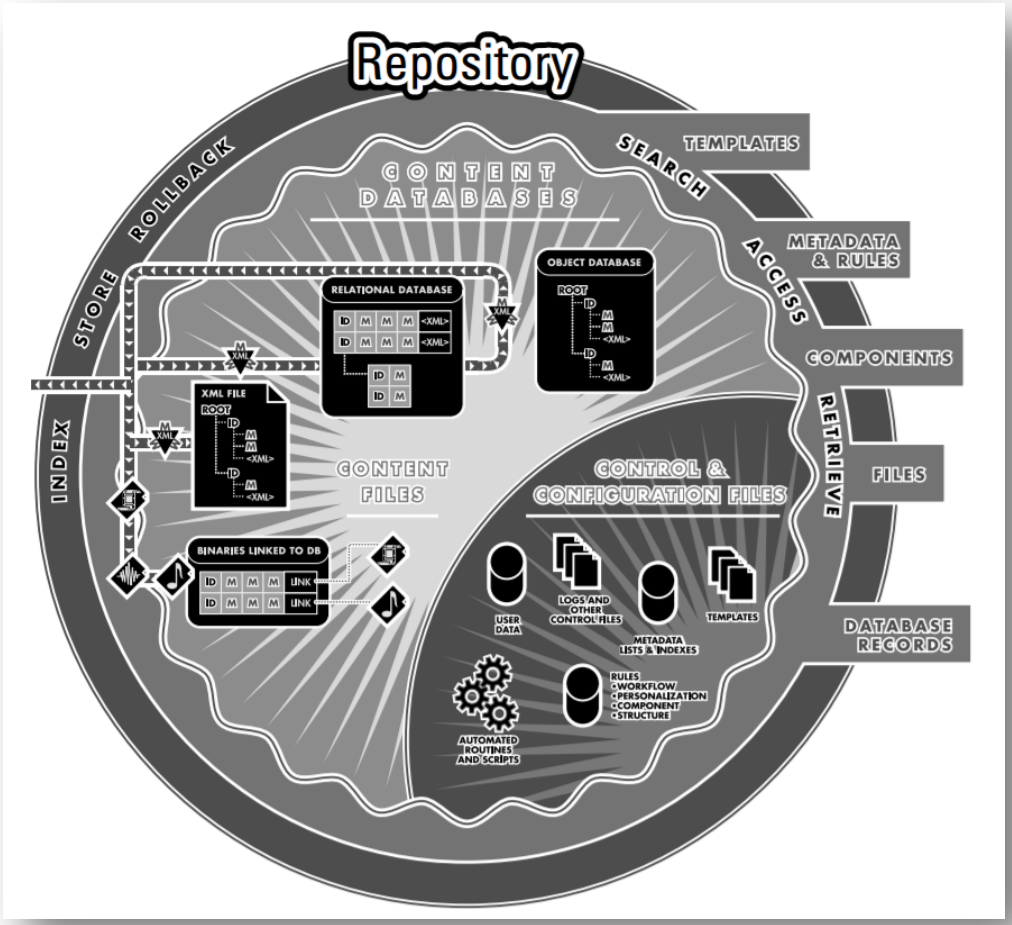


Figura 3 – Esquema del repositorio para un WCMS. Fuente: [1].

11.2.1.1 *Archivos y bases de datos de contenido*

Los archivos y bases de datos de contenido almacenan el contenido de los componentes del sistema. Las bases de datos de contenido pueden consistir en una base estándar relacional, una base de datos de objetos XML o un híbrido de ambas.

Las bases de datos relacionales utilizan tablas, columnas y filas para representar los componentes, en cambio en las bases de datos orientadas a objetos XML los componentes quedan completamente representados como un XML y son persistidos en una enorme jerarquía. En este caso cada tipo de contenido, componentes y los elementos que contienen son representados mediante una etiqueta.

11.2.1.2 *Archivos de configuración y control*

Los archivos de configuración y control no son archivos de contenido, pero de todas formas serán manejados por el sistema administrador y persistidos en el repositorio.

Los archivos de control y configuración incluyen entre otros:

- Archivos de configuración de acceso al personal de la organización y archivos y bases de datos del usuario final.
- Archivos de reglas y bases de datos.
- *Log* y archivos de control y estructura.
- *Scripts* y rutinas de mantenimiento automático.

11.2.2 Administración

El sistema administrador es responsable de configurar los parámetros y estructuras del WCMS. Dichas configuraciones afectan a todas las piezas del WCMS de la siguiente forma:

- **En el sistema de recolección**
  El sistema administrador define la configuración del personal, donde los roles y los derechos de accesos son definidos.
- **En el sistema de gestión**
  Los administradores se encargan de diferentes tareas como el mantenimiento, definición de permisos y respaldo del contenido persistido en el repositorio. Adicionalmente se definen tipos de contenido, se realizan revisiones de metadata y se crean *workflows*.
- **En el sistema de publicación**
  Los administradores se aseguran que el software y hardware para mostrar y transportar el contenido esté funcionando de forma correcta.

11.2.3 Workflow

El sistema de *workflow* es el responsable de la coordinación, programación y controlar de rutinas, estás tienen objetivos variados e impactan en los distintos sistemas del WCMS de forma diversa:

- **En el sistema de recolección**
  En el sistema de recolección vamos a encontrar *workflows* definidos para tareas de creación y agregación de contenido. En la mayoría de los casos se definen *workflows* para realizar un seguimiento a diferentes tipos de contenidos desde su creación hasta su publicación, por ejemplo se incluyen tareas de creación, chequeo y aprobación de contenido.
- **En el sistema de gestión**
  Existen *workflow*s para manejar o programar diferentes tareas de respaldo y archivado de contenido, también por ejemplo existen *workflows* creados para chequear la utilización de contenido y alertar cuando ya no es utilizado.
- **En el sistema de publicación**
  Aquí el objetivo de los *workflows* es chequear la calidad cada vez que se crea una publicación de contenido asegurando que el resultado sea el mejor esperado.

11.3 Sistema de publicación

El sistema de publicación es responsable por extraer los componentes de contenido y otros recursos del repositorio creando publicaciones automáticas con dichos elementos.
El sistema de publicación incluye:

11.3.1 Plantillas de publicación

Las plantillas de publicación son archivos que brindan una guía para la publicación del contenido persistido en el repositorio. Las plantillas de publicación contienen los siguientes elementos:

- **Elementos estáticos**
  Estos elementos son texto, scripts y medios que pasan directo a la publicación sin demasiado procesamiento.

- **Llamadas a servicios de publicación**
  Estos llamados recuperan y le dan forma a los componentes utilizados del repositorio y su metadata, adicionalmente se realizan tareas de personalización de reglas, conversión de contenido y construcción de navegación.

- **Llamadas a servicios fuera del WCMS**
  Se integra la publicación a una infraestructura más grande en la organización teniendo la posibilidad de obtener datos de sistemas fuera del WCMS mediante, por ejemplo un *web service*.

11.3.2 Servicios de publicación

Los servicios de publicación son la capa lógica y de negocio que le brinda soporte al WCMS para la creación de publicación del contenido y metadata almacenada en el repositorio.
Dentro de las funciones que deben brindar los servicios de publicación se encuentran:

- **Carga y ejecución de plantillas**
  Estos servicios se encargan de procesar, convertir, extraer y darle formato a las distintas plantillas para crear una publicación.

- **Proveer servicios específicos de publicación**
  Estos servicios pueden incluir desde salidas en PDF para imprimir hasta liberaciones incrementales para actualizar un sitio web.

- **Proveer servicios puentes a sistemas no-WCMS**
  Estos servicios son llamados para proveer información de otros sistemas que se puede incluir en las publicaciones.

11.3.3 Publicaciones web

El uso más común hoy en día de los WCMS es para crear y administrar sitios web de gran porte.
Si las publicaciones web son dinámicas, básicamente cada vez que una página web del sitio es cargada en el navegador los servicios de publicado web realizan las siguientes tareas:

- Cargan las plantillas asociadas.
- Transfieren los parámetros que hayan llegado con el pedido que creó la interacción del usuario con la web.
- Ejecutan el código que contiene las plantillas para producir la página web que va a ver el usuario.
- Envían la página web creada a través del servicio web para que sea mostrado en el navegador del usuario.

En la actualidad la mayoría de los WCMS son dinámicos, la página web son creadas en el momento de ejecución a partir de las plantillas disponibles. Generalmente los WCMS crean distintas estrategias de caché para evitar generar nuevamente una página si la plantilla no sufrió ningún cambio.

12 Equipo para la gestión de contenido

Desde el inicio hasta la etapa el lanzamiento y posterior uso, un proyecto de gestión de contenido puede impactar en muchas personas a través de la organización, todos con diferentes roles y responsabilidades.
En primer lugar, los miembros de un equipo de gestión de contenido pueden ser divididos en:

- Editores.
- Planificadores de sitios.
- Desarrolladores.
- Administradores.
- Stakeholders.

Es importante notar que lo enumerado anteriormente son roles, no personas. También ocurre que los roles no son absolutos, y sería extraño ver un proyecto donde cada rol descrito fuera ocupado por una persona, los miembros de un equipo usualmente tienen varios roles.

12.1 Editores/Autores

Los editores son los responsables de la creación, edición, y gestión del contenido en el WCMS.
Los editores tienden a ser agrupados en un solo grupo, pero el rol de "editor" es una generalización, ya que todos los editores no son iguales, y podrían tener una amplia variedad de capacidades.
Lo que caracteriza a un editor de la "corriente principal" o "normal" es específica para el proyecto. Por lo tanto se analiza cómo los editores pueden estar limitados en su capacidad para refinar sus subroles:

12.1.1 Por sección/rama/locación

Los editores pueden solamente editar un subconjunto específico de contenido en el sitio web, el cual puede ser una sección, una rama en un árbol de contenido, o algún otro método de localización. Podrían tener control completo sobre el contenido en esa área, pero no control completo sobre el contenido de otras áreas.

12.1.2 Por tipo de contenido

Los editores son capaces de editar tipos específicos de contenido. Ellos pueden manejar los perfiles de los empleados, los cuales aparecen en sitios de varios departamentos, o manejar artículos de noticias de una compañía, independientemente de la locación. De hecho, algunos editores están mejor definidos por los tipos de contenido que no les está permitido crear.

12.1.3 Por interfaz de edición

Los editores pueden estar limitados por la interfaz que tienen permitido utilizar. En instalaciones más grandes, no es raro canalizar ciertos editores a través de interfaces especializadas, hechas a medida, diseñadas para permitirles manejar sólo el contenido bajo su control.
En contraste a las limitaciones está el llamado “editor de poder”, quién puede realizar en todos los contenidos operaciones a través del sitio web. Esta persona a veces desarrolla múltiples tareas dentro de la organización como la del administrador del sitio, entrenador, experto en la materia, entre otras.
Existen otros roles editoriales comunes, dentro de los cuales se encuentran:

12.2 Aprobadores

Este rol es responsable de revisar el contenido enviado, asegurarse de que sea válido, acertado, y de calidad aceptable, y entonces publicar el contenido. Los aprobadores desarrollan pasos en más de un *workflow*. Muchos editores son también aprobadores, responsables de vetar contenido enviado por editores de menor experiencia. Estos editores pueden también tener el derecho de aprobar su propio contenido.

Algunos aprobadores pueden tener la habilidad de editar contenido enviado anterior a la publicación (por ejemplo un editor en jefe) mientras que otros aprobadores pueden tener solamente la habilidad de aprobar o rechazar. Este rol sólo se puede necesitar para entender las características de la aprobación de contenido del WCMS.

12.3 Comercializador

Este rol es responsable de revisar el contenido para su impacto de *marketing*, y gestionar el valor de *marketing* en el sitio web en su totalidad. Requiere entendimiento de las características de *marketing* y analítica del WCMS. Para algunos sitios, este es el rol dominante porque el nuevo contenido no se crea con tanta frecuencia como el contenido existente necesita ser optimizado, promovido, y analizado.

12.4 UGC/Administradores de la comunidad

Este rol es responsable de verificar la oportunidad del contenido enviado por los usuarios (contenido generado por usuarios, UGC en inglés), como información de perfil del usuario y comentarios en un blog. Estos administradores son similares a los aprobadores, pero tienen control únicamente sobre el contenido generado por los usuarios y no sobre el contenido editorial. Además, dado que el volumen de contenido generado por los usuarios es a menudo alto, es común que sea manejado luego de su publicación (o después de que es recibida una queja) y no dejarlo a la espera de publicarse hasta que no sea revisado. Este rol sólo tendrá que entender al WCMS en la medida en que les permite moderar el contenido generado por los usuarios. En algunos casos, el WCMS proporciona herramientas separadas para esto, mientras que en otros se maneja como contenido normal.

12.5 Traductores

Esta función es responsable de la traducción de los contenidos de un idioma a otro.

Los traductores sólo necesitan comprender la funcionalidad editorial del WCMS en la medida necesaria para añadir las traducciones de los objetos de contenido específicos (tal vez incluso de tan sólo atributos de contenido específico, en el caso de que los objetos de contenido son sólo parcialmente traducidos). La traducción de contenidos es a menudo llevado a cabo por organizaciones externas. En estos casos el traductor estará en forma remota y podría no trabajar con el WCMS en ningún momento, en lugar de mover contenido en ambos sentidos a través de un flujo de trabajo específico de traducción y de un formato de intercambio, como por ejemplo XLIFF.

12.6 Planificadores de sitios

Son los responsables de diseñar el sitio web que el WCMS gestionará. La mayoría de su participación será antes del lanzamiento, con participación más esporádica cuando el sitio se desarrolla y cambia con el tiempo.

Existen varios subroles, los cuales son:

12.6.1 Estrategas de contenido

Este es el rol responsable de diseñar contenido, holística y tácticamente. Como un subproducto del proceso de planificación de contenido, los estrategas definen los tipos de contenido e interacciones que el sitio web debe soportar. Este rol requiere conocimiento de cómo un WCMS modela y agrega contenido para entender cualquier limitación en el diseño. El conocimiento adicional de las características del *marketing* será necesario si el estratega es responsable de optimizar el valor de *marketing* del sitio previo a su lanzamiento.

12.6.2 Diseñadores de experiencia de usuario y arquitectos de información

Estos roles son responsables de organizar contenido y diseñar la interacción de los usuarios con el sitio web. Ellos necesitarán entender como el WCMS organiza el contenido, y que facilidades están disponibles para agregar y presentar contenido a los usuarios finales.

12.6.3 Diseñadores visuales

Este rol es responsable del diseño final, de alta fidelidad del sitio web (en oposición a la baja fidelidad de los prototipos y flujos de usuario provistos por los roles anteriores). Los diseñadores visuales no necesitan un gran conocimiento del WCMS, ya que las limitaciones relacionadas al mismo guiarán el proceso hasta su participación. En algunos casos, este rol puede solaparse con el modelado de plantillas.

12.7 Desarrolladores

Los desarrolladores son responsable de la instalación, configuración, integración, y modelado de plantillas del WCMS para alcanzar los requisitos del proyecto.
Cuánto esfuerzo lleve el desarrollo es específico de la complejidad de los requisitos y lo bien que se corresponda el WCMS con los requisitos *out of the box*. El despliegue de un simple *blog* con WordPress tendrá muy poco desarrollo (quizás ninguno en absoluto), mientras que una intranet empresarial construida desde cero es una tarea enorme.
Como los editores, no todos los desarrolladores son iguales, existen múltiples categorías de tareas que definen diferentes roles:

12.7.1 Configuración WCMS

Este es el rol responsable de la instalación y configuración del WCMS en sí mismo, incluyendo el establecimiento del modelo del contenido, creación de flujos de trabajo y otras herramientas editoriales, creación de grupos de usuarios, roles, y permisos, etc. Este trabajo es hecho a un nivel muy alto, a través de instalaciones e interfaces provistas por el WCMS

12.7.2 Desarrollo de backend (servidor)

Este es rol responsable por el desarrollo a más bajo nivel realizado en un lenguaje de programación tradicional (PHP, C#, Java, etc.) para llevar a cabo tareas de gestión de contenido más complejas o para integrar el WCMS con otros sistemas. Este desarrollador tiene que tener experiencia en el lenguaje de programación requerido y en la API del WCMS.

12.7.3 Desarrollo de frontend o modelado de plantillas

Este rol es responsable de la creación de código HTML, CSS, JavaScript, lógica de las plantillas, etc. necesario para presentar contenido gestionado en un navegador. Este desarrollador solo necesita saber sobre el lenguaje de modelado de plantillas y arquitectura provistas por el WCMS, y como se integra con HTML, CSS y JavaScript.

12.8 Administradores

Los administradores son los responsables de la operación continua del WCMS y la infraestructura asociada.
Dentro de este grupo hay varios subroles:

12.8.1 Administrador WCMS

Este rol es responsable de gestionar el WCMS en sí mismo, lo cual incluye el manejo de usuarios y permisos, flujo de trabajo de creación y gestión, gestión de licencias, y todas las tareas no relacionadas a la creación de contenido.

12.8.2 Administrador del servidor

Este es el rol responsable del mantenimiento y soporte del servidor (o servidores) en el cual corre el WCMS y/o se despliega contenido. Este es un rol tradicional en la industria informática, y el administrador del servidor a menudo no tiene conocimiento sobre el WCMS más allá de la arquitectura básica requerida para que corra sin errores (sistema operativo, entorno en tiempo de ejecución, servidor web, etc.). Este rol provee soporte cuando hay un problema en el servidor que evite que el WCMS funcione correctamente.

12.8.3 Administrador de base de datos/almacenamiento

Este rol es responsable por la gestión del servidor de base de datos y redes de almacenamiento que mantienen el contenido del WCMS. Este administrador necesita muy poco conocimiento sobre el WCMS, más allá de los tipos de archivos, tamaños, y volúmenes que necesitan ser almacenados y respaldados.

12.9 Stakeholders

Los *stakeholders* de un proyecto WCMS son un grupo que representa a las personas responsables por los resultados que el WCMS pretende brindar. Los *stakeholders* por lo normal son personas de negocios o *marketing* que ven al WCMS simplemente como un medio para un fin. Por lo general, los *stakeholders* ven en un WCMS que haga una de dos cosas:

- Aumentar ingresos.
- Reducir los costos y/o riesgos.

Los *stakeholders* a menudo no tienen contacto directo con el WCMS, y no se preocupan por las características específicas que presenta el WCMS, el único objetivo es que el resultado del WCMS sea visible.

13 Adquirir un WCMS

Existen cuatro paradigmas de adquisición de un sistema WCMS, los mismos son:

- Código abierto: Se descarga e instala.
- Comercial: Se compra una licencia y se instala.
- Software como un servicio (SaaS en inglés): Se “renta” y se usa.
- Construcción propia: Se desarrolla desde cero, dentro de la organización.

13.1 WCMSs de código abierto

A pesar que al ser de código abierto no se deba pagar para obtener una licencia, puede ocurrir que de todas formas se deba incurrir en gastos, por ejemplo, de alojamiento e integración del sistema.
Algunos WCMS son fáciles de alojar por una módica cantidad. Otros no tanto, y podrían necesitar más librerías, poder computacional, y permisos adicionales que el servicio de alojamiento promedio ofrece, por lo tanto, requiere de un ambiente de auto alojamiento con control total.
En cuanto a la integración, el software de código abierto también varía mucho. Algunos proyectos ya cuentan con abundante documentación e instaladores en el arranque para poder ejecutar rápidamente. Pero estas prestaciones a menudo son desarrolladas tardíamente en el ciclo de vida del software de código abierto, por lo que las empresas jóvenes muchas veces fallan en esta área. Además, a menudo hay una tendencia clara en el desarrollador de software de código abierto ("escrito por desarrolladores, para los desarrolladores"), y una sensación general de que ya que nadie está pagando por ello, los usuarios pueden entenderlo por sus propios medios.
Dada la falta de una licencia, los sistemas de código abierto son usados a menudo en proyectos pequeños que no tienen el presupuesto para pagar una licencia. Esto significa que la aplicabilidad a proyectos mucho más grandes podría ser cuestionable.
Su uso global presenta enormes ventajas en el soporte de las comunidades. Muchos sistemas de código abierto tienen prósperas comunidades de usuarios que están disponibles para contestar preguntas rápidamente y de forma acertada. Sin embargo, juega en contra el hecho de la falta de soporte profesional y de habilidad o voluntad por resolver problemas más intrincados.

13.1.1 Modelos de negocios de compañías de código abierto

Para pagar las cuentas, las compañías detrás del software de código abierto operan en uno o más de los siguientes modelos:

13.1.1.1 Consultoría e integración

Nadie conoce mejor el WCMS que la compañía que lo desarrolló, y es bastante común para las empresas integrar su propio software desde el inicio hasta el final, o por lo menos proveer servicios de consultoría de alto nivel con los cuales se puede asistir a los clientes para que lo integren por ellos mismos.

13.1.1.2 Freemium

El software básico es gratis, pero existe la opción paga que permite el acceso a más funcionalidades, un volumen más grande de contenido gestionado, u opciones de escalado, como la habilidad del balance de carga. A veces el producto gratuito es bastante capaz, y a veces es solamente una versión de prueba que intenta dirigir a los usuarios a pagar por la versión completa.

13.1.1.3 Alojamiento

Muchos vendedores ofrecen plataformas de “alojamiento gestionado” para los sistemas de código abierto que ellos desarrollan. El supuesto beneficio es un ambiente de alojamiento diseñado específicamente para ese sistema, y/o expertos en el sistema pendientes por eventos de problemas en el alojamiento. Es de notar que el valor actual es un poco cuestionable, ya que raramente hay información secreta conocida sólo por el vendedor que permita ajustar una plataforma de alojamiento para un WCMS en lugar de otro.
Cualquier mejora disponible de rendimiento o retoques de configuración podrían ser implementados fácilmente por un cliente capaz. El valor es a menudo solo la tranquilidad de que un “experto” esté a cargo.

13.1.1.4 Entrenamiento y documentación

El software de código abierto a menudo carece de documentación, y las tendencias del desarrollador pueden llevar a sistemas idiosincrásicos, de APIs pesadas. Por estas razones, el entrenamiento profesional puede ser útil. Muchos vendedores ofrecen opciones de entrenamiento pagas, ya sea de forma remota como en persona.
De forma menos común, algunos vendedores ofrecen acceso pago a documentación de alta calidad.

13.1.1.5 Licenciamiento comercial

Dependiendo en la licencia exacta, los cambios al software libre podrían tener que ser lanzados públicamente a la comunidad. Algunos proveedores ofrecen licencias comerciales pagas para que sus sistemas de código abierto permiten a las organizaciones ignorar este requisito, cerrar la fuente, y mantener sus cambios para sí mismos.

13.1.1.6 Soporte

Cuando el soporte de la comunidad se queda corto, el soporte profesional puede resultar de utilidad, y algunos vendedores proveen una opción de soporte paga, ya sea una subscripción anual o soporte básico por incidente.

13.1.1.7 Testeo adicional y aseguramiento de calidad

Algunos vendedores ofrecen una versión paga del software que está sujeta a un nivel "empresarial" más alto de testeo y aseguramiento de calidad. En estos casos, la versión gratuita o "comunitaria" es presentada como testeada livianamente y sin soporte, mientras que la versión empresarial (paga) está comercializada como la única adecuada para implementaciones más demandantes.

13.2 WCMSs comerciales

Como cualquier otro género de software, numerosos vendedores WCMS comerciales están disponibles para vender sus licencias y usar sus sistemas.
Es importante diferenciar el objetivo de una compra de un sistema comercial en lugar de utilizar uno de código abierto. En primer lugar, una compañía comercial se presenta a sí misma como una entidad de negocios más formal que una comunidad de código abierto, lo cual es importante para algunas organizaciones. En segundo lugar, los vendedores comerciales, generalmente adhieren a un estándar más alto y tienen ingresos de las licencias para financiar el desarrollo profesional.
Como cualquier generalización, esto no siempre es verdad, ya que algunos sistemas de código abierto son suficientemente maduros para competir con cualquier oferta comercial. Por el contrario, algunos vendedores comerciales son muy pobres en cuanto al aseguramiento de la calidad y venden productos plagados de errores. Pero como regla general, se mantiene la generalización.
Adicionalmente, en los últimos cinco años, ha habido una separación distintiva entre WCMS de código abierto y comercial en cuanto a las líneas de prestaciones de *marketing*. Mientras que la comunidad de desarrollo de código abierto está obsesionada con resolver problemas de gestión de contenido, el mundo comercial se ha movido hacia el *marketing* del contenido, el cual consta de las herramientas y prestaciones que ayudan a mejorar el contenido una vez que es publicado.
Se dice que los WCMS de código abierto están hechos por el oficial en jefe de información (CIO en inglés) mientras que los WCMS comerciales están hechos por el oficial jefe de *marketing* (CMO en inglés). Esta afirmación es verdadera en cómo los sistemas son comercializados, con el sector comercial concentrándose solamente sus ventas en los departamentos de *marketing* de sus clientes, mientras que los proveedores de código abierto están más interesados en tratar de capturar miembros del plantel tecnológico.

Como con las ofertas de código abierto, las distinciones de plataformas son claras. Muy pocos sistemas LAMP son comercializados, mientras muchos de los sistemas hechos en .NET o Java sí lo son.
Es importante recordar que los costos manejados corresponden solamente a los costos de licencia. Comprar un WCMS comercial no libera de los costos de implementarlo, se necesitará encontrar (y pagar) alguien que instale, configure, y de forma al sistema. En algunos casos, los vendedores comerciales proveen una opción para esto llamada “servicios profesionales”, y en otros casos ellos tienen una “red de socios” de firmas de integración quienes son expertos en sus sistemas y dispuestos a integrar el sistema mediante un pago.

13.2.1 Modelos de licenciamiento

Los sistemas comerciales raramente tienen un solo precio. Los vendedores usualmente tienen un sistema de fórmulas y tablas para determinar el precio final, para adaptarse a los clientes con presupuestos más altos como a los más bajos.
El rango de precios comerciales es muy vasto, puede ir de precios ínfimos hasta precios superiores a U$S 250.000 (como en el caso de Adobe AEM) y para grandes instalaciones pasará fácilmente el millón de dólares.
A continuación se detallan las maneras más comunes de determinar el precio:

13.2.1.1 Por editor/usuario

El sistema es cotizado por el número de usuarios editores. Menos común es que el sistema sea cotizado por el número de usuarios públicos registrados, pero esto usualmente solo aplica a sistemas comunitarios o de intranet/extranet donde se espera que los usuarios se registren.

13.2.1.2 Por servidor

El sistema es cotizado por el número de servidores en los cuales el sistema funciona (o menos comúnmente sobre el número de núcleos del procesador). Esto es bastante común ya que grandes instalaciones necesitan de muchos servidores, evitando una cotización más alta. Con sistemas desacoplados donde el entorno de entrega está separado del entorno de repositorio, esto puede volverse confuso, ya que no queda claro si se paga por los servidores de entrega, los de repositorio o ambos.

13.2.1.3 Por sitio

El sistema es cotizado por el número de sitios web distintos que están corriendo en él. Esto puede resultar un tanto difuso dado lo vago de lo que constituye un sitio web.

13.2.1.4 Por prestación

El sistema es cotizado por cada paquete instalado en adición al núcleo del sistema. Casi todos los vendedores comerciales tienen múltiples prestaciones o paquetes que pueden ser agregados al sistema de base para aumentar su valor o precio. Estas van desde subsistemas de comercio electrónico a herramientas de *marketing*. Es de notar que cada una de estas prestaciones podría también ser licenciada por usuario, servidor, o sitio, haciendo los precios variables.

13.2.1.5 Por volumen de contenido

El sistema es cotizado por la cantidad de contenido bajo gestión. Este es menos común que otros modelos, ya que la cantidad de objetos de contenido gestionado depende en gran medida de la implementación.

La mayoría de los sistemas son cotizados en múltiples aristas, por ejemplo, por una combinación de editores, servidores, y sitios. La cotización final puede a menudo ser imposible de determinar sin una consulta a fondo con el departamento de ventas del vendedor.

13.2.1.6 Subscripción de software

Algo a lo que hay que tener prácticamente siempre en cuenta con los vendedores comerciales es la necesidad de pagar por la suscripción de software, la cual es una cuota anual continua basada en el precio de la compra. Esto no es único para los WCMSs, casi todas los *softwares* empresariales se cotizan de manera similar.
La suscripción es generalmente un porcentaje del precio de compra, típicamente del 18% al 22%. El primer año a menudo es incluido en la compra, pero el cliente será facturado por la fecha de aniversario y cada año siguiente.
Si se toma como promedio el 20% anual, se estima que se “recompra” el WCMS en un período de tiempo que va de 4 a 6 años.
Pagar esta cuota varía por vendedor. En la mayoría se puede simplemente dejar de pagar la suscripción en cualquier momento y seguir utilizando el producto, pero se perderán todos los beneficios de valor agregado que garantizaba la cuota de subscripción. En la mayoría de los vendedores, los beneficios son una combinación de los siguientes:

- Soporte por demanda.
- Actualizaciones y parches a medida que son liberados.
- Licencias gratuitas para servidores de desarrollo o prueba.
- Gestión de licencias, en el caso de necesitar licenciar nuevos servidores o sitios.

13.3 Software como un servicio

En lugar de comprar e instalar un WCMS, simplemente se paga una cuota mensual y se corre en el sitio web del cliente dentro de un sistema más grande administrado por el vendedor. Cada cliente corre sus sitios web dentro del mismo sistema.
Este tipo de software se conoce como de "tenencia múltiple", ya que una sola instancia instalada en el servidor del vendedor sirve a múltiples clientes. Por el contrario el software adquirido e instalado es de "tenencia simple”.
Los beneficios esperados son la ganancia de tiempo y la evitación de problemas de alojamiento. En efecto, el sistema ya está corriendo, a la espera del cliente, y ya que el mismo corre en el servidor del vendedor, se evitan las preocupaciones concernientes al manejo de la infraestructura. SaaS estaba en la “nube” antes de que el término se hiciera conocido.
Otra gran ventaja es que siempre se va a estar a la vanguardia en cuanto a las versiones. Ya que el vendedor corre el entorno de alojamiento, cuando libera una nueva versión, el cliente ya dispone de la misma.
Sin embargo la ventaja de uso inmediato que presentan los SaaS fue marginada aún más con la llegada de la virtualización de servidores y computación en malla como EC2 de *Amazon* y *Microsoft Azure*. Se puede tener una instalación de casi cualquier WCMS en menos de una hora.
Estos sistemas no son de tenencia múltiple, pero ofrecen los mismos beneficios de ahorro de tiempo y alojamiento en terceros.

13.4 Construir un WCMS propio

Como cualquier otro software, un WCMS puede ser hecho es una organización por el equipo de desarrollo. En algunos sentidos, un WCMS se asemeja a cualquier otra aplicación basada en datos, y no es difícil construir un WCMS sencillo con bastante rapidez.
Para este modelo hay algunas razones que lo justifican, como:

- No requiere una cuota de licenciamiento.
- El que lo realice se convertirá en un experto en el uso del sistema resultante y no sufrirá la curva de aprendizaje de un sistema existente.
- Se construirán solamente las funcionalidades necesarias, evitando software inflado y compilación innecesaria.

En un análisis más profundo, algunas de estas razones resisten el análisis. A menudo, el proyecto se justifica basado en un conocimiento muy superficial de las necesidades de la organización o en general de la disciplina de gestión de contenidos. Mientras que es posible generar ganancias rápidas, la primera emoción de progreso desaparece demasiado rápido, y finalmente, la organización se encuentra reconstruyendo grandes piezas de funcionalidades básicas de los WCMS que otros sistemas tienen desde hace mucho tiempo resuelto.
Desde el exterior, un WCMS se ve como un simple ejercicio de contenido de edición y publicación. Pero yendo con más profundidad en el proyecto, los editores comienzan a pedir características como control de versiones, múltiples idiomas, flujo de trabajo, gestión de múltiples sitios, etc. Estas funciones pueden ser engañosamente complejas para desarrollar, más aún cuando tienen que ser instaladas en un sistema en ejecución.
Una sutileza es que muchos de los problemas que implica la construcción de un WCMS son lógicos y conductuales, más que técnicos. Los vendedores que trabajan en este espacio tienen el beneficio de años de experiencia con los editores y cómo trabajan con el contenido. Hay a menudo una "cuota de entrada" de implementación de una función equivocada dos o tres veces antes de hacer las cosas bien. Si se construye algo desde cero, a menudo se va a pagar esta cuota cada vez que se expande el sistema. Típicamente, la mayoría de las organizaciones cruzan una “línea de arrepentimiento” donde les gustaría volver atrás y elegir una opción ya hecha. No es común ver un resultado positivo de un esfuerzo hecho en una organización en el largo plazo. Sin embargo, hay situaciones donde podría ser la elección correcta, como por ejemplo:

- Cuando el modelo de contenido es muy específico a la organización. Por ejemplo si todo lo que se va a publicar son videos de autos, construir una plataforma de gestión podría ser sencillo.
- Cuando las necesidades de la organización son muy simples y los planes de desarrollo futuro son absolutamente conocidos que serán limitados.
- Cuando el WCMS se basa en gran medida en el aprovechamiento de los *frameworks* existentes para evitar el mayor re trabajo posible. Por ejemplo *Symphony* para PHP, *Django* para *Python*, *Entity Framework* y MCC para ASP.NET.

14 Prestaciones de un WCMS

Las plataformas WCMS son productos potentes -que van más allá de la simple publicación de contenido para un sitio web- que cuentan con varias prestaciones que fundamentan dicha afirmación y que les brindan un valor agregado. Algunas de las prestaciones que brindan son las siguientes:

- **Escalabilidad**

- En particular al contar con un WCMS desacoplado –gestión y consumo de contenido en distintos servidores-, la publicación de contenido se vuelve escalable, estable y segura [2].
- **Integración de módulos**
  Permiten la integración con módulos externos, por ejemplo, con los relacionados con su lado comercial y de *marketing* –en particular *Analytics*-.
- **Experiencia de usuario**
  Cada combinación de un visitante más contenido es una nueva experiencia de usuario. Un WCMS es capaz de administrar, optimizando la interacción con el visitante a través de una variedad de herramientas, como por ejemplo, personalización, *testing*, *analytics* [2]. El sitio web podría cambiar en función del perfil del usuario, proveyendo contenido y funcionalidades que ese visitante puede requerir en ese momento [2].
- **Independencia de personal de IT**
  Es posible la utilización de un WCMS sin conocimiento técnico en el área de tecnologías de la información, por ejemplo para los autores, solo bastaría con la práctica suficiente para la utilización del WCMS.
- **Multicanal**
- Se pueden brindar muy buenas experiencias de usuario para una gran variedad de dispositivos, en particular para los dispositivos móviles. Puede ser logrado a través de diseño adaptativo, responsivo (web) o a través de diseño a través de APIs que permitan el desarrollo de aplicaciones móviles. Adicionalmente, atendiendo al avance y auge de los últimos años de las redes sociales, éstas se han convertido en un canal más que importante.
- **Soporte de varios idiomas**
  Permite la creación de traducciones para asegurar la distribución del mismo en el idioma adecuando según el visitante del sitio web.
- **Administración multisitio**
  Es posible administrar una cantidad arbitraria de sitios web a través de una única instancia.
- **Procesamiento de imágenes**
  Se brinda una biblioteca de imágenes, permitiendo subir las que se deseen para su posterior uso, además de imágenes que ya trae precargadas.

15 Presente del WCMS

Actualmente, la utilización de WCMS ha ido en expansión, principalmente por los grandes volúmenes de información manejados por los sitios web y por las ventajas que presenta la utilización de una plataforma de este tipo. Los usuarios de estas plataformas van desde usuarios particulares, hasta grandes empresas a nivel mundial que necesiten de una gestión eficiente y eficaz de su sitio web.

En la actualidad, de acuerdo al sitio https://builtwith.com [9], el cual realiza un análisis de todas las plataformas WCMS utilizadas para construir sitios web que componen Internet, se llega a la conclusión que los WCMS de código abierto lideran ampliamente en el mercado, en particular WordPress que es utilizado por el 51% de los sitios web que utilizan alguna plataforma WCMS. Los WCMS que secundan a WordPress también son de "código abierto", entre los cuales se encuentran Joomla (7%), Blogger (2%), y Drupal (2%). Se puede apreciar lo mencionado en la Figura 4 .

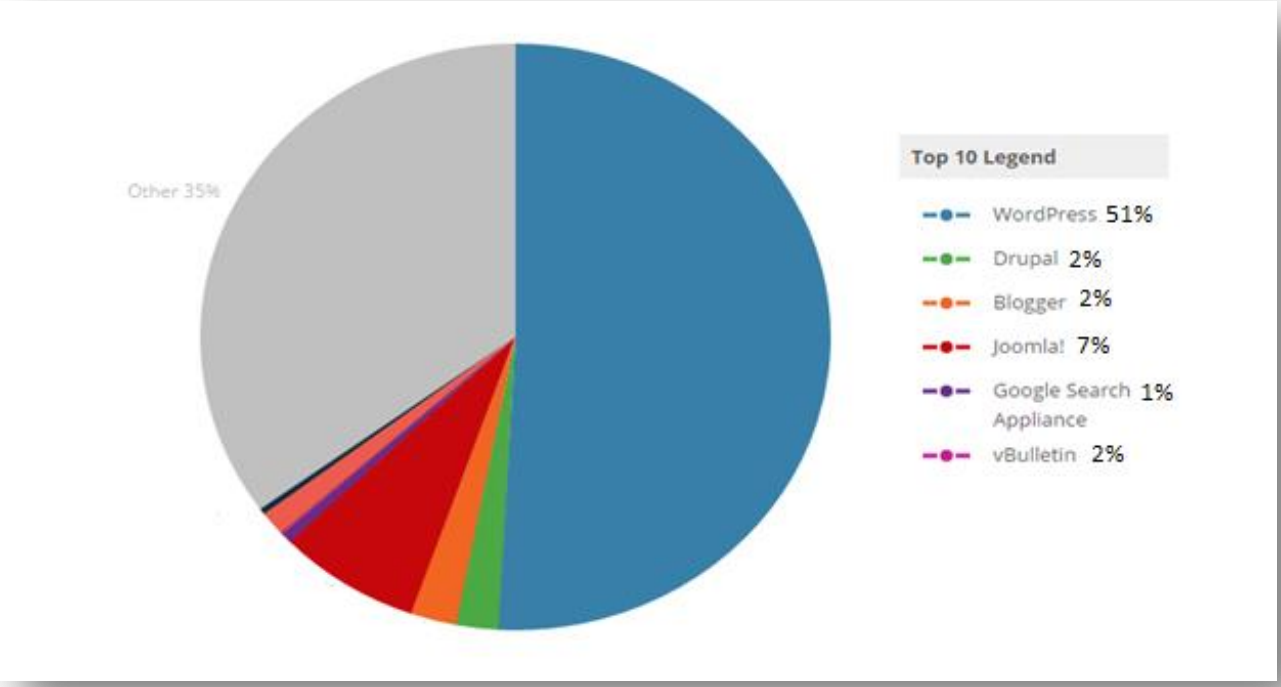


Figura 4 - Gráfico de uso de plataformas WCMS en sitios web. Fuente: www.builtwith.com [9]

En cuanto a los productos propietarios, las consultoras Gartner [10] y Forrester [11] analizan los principales productos WCMS del mercado de acuerdo a determinados criterios tomados por cada una de ellas. Es importante mencionar que no se puede brindar una comparativa de las plataformas de código abierto –al menos las más importantes dentro de ese segmento-, ya que los informes evalúan prácticamente en su totalidad WCMS comerciales. No se determina explícitamente la razón de la no inclusión de las plataformas de "código abierto" en los informes estudiados, pero es bastante factible que una de las razones principales haya sido el no cumplimiento de uno a más criterios de inclusión, en particular el de los ingresos generados por estas plataforma o la cantidad de empresas que lo utilizan –al ser en mayor parte de uso gratuito-.

Tanto para Forrester como para Gartner, se van a tener en cuenta los dos últimos informes.

Para el caso de Forrester, presenta los resultados en su llamada Ola de Forrester. Esta grafica está compuesta por cuatro categorías, y cada producto WCMS evaluado es ubicado de acuerdo a la oferta actual que ofrecen y la fortaleza de su estrategia. También se visualiza en una escala de cinco posibles valores, la presencia en el mercado de cada plataforma. En primer lugar, en la Figura 5, se aprecia la ola de Forrester correspondiente a su informe del año 2015 [12]:

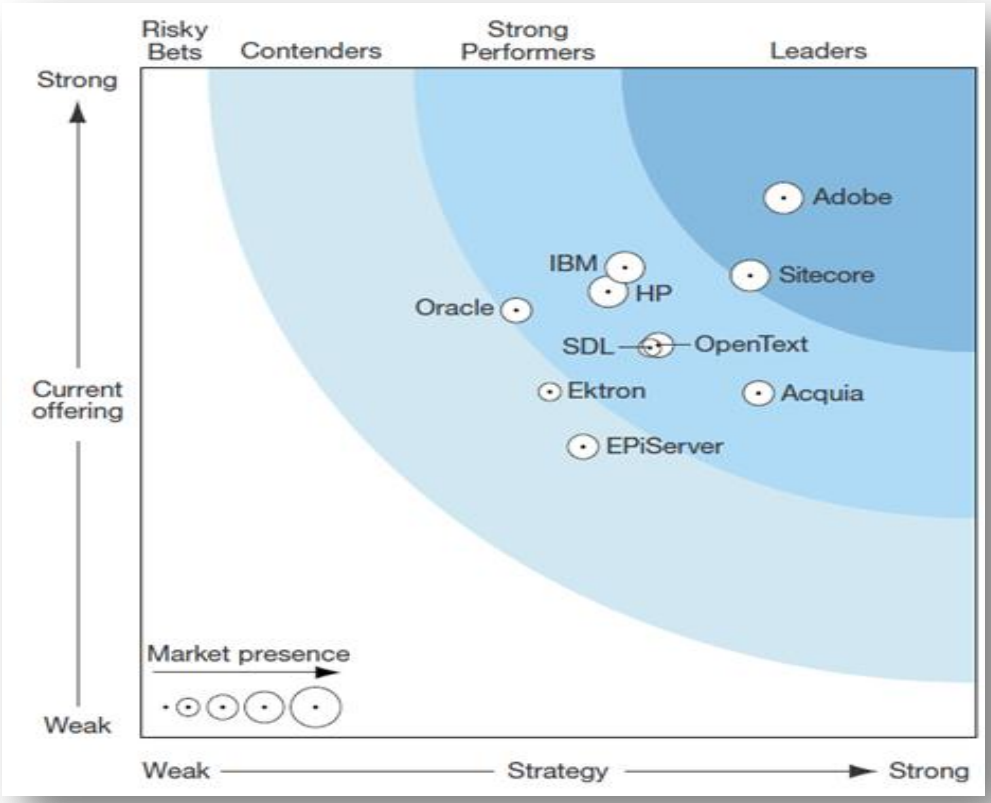


Figura 5 - Ola de Forrester. Fuente: Informe Forrester febrero 2015 [4].

Se desprende de la figura anterior que para Forrester, Adobe AEM y Sitecore son líderes del mercado en cuanto al producto WCMS ofrecido, además de poseer una buena presencia en el mercado. Forrester saca la siguiente conclusión en su informe del año 2015:
"*Adobe y Sitecore son líderes, con Adobe dominando el campo. Adobe sobresale por su portafolio integrado. Sitecore tiene un producto fuerte y con gran crecimiento*".
A continuación se presenta otra sección del informe de Forrester, el cual califica –para cada plataforma WCMS evaluada- una gran cantidad de puntos, agrupados dentro de las tres conceptos mostrados en la ola –oferta actual, estrategia y presencia en el mercado-. En otras palabras, es la gráfica mostrada previamente, pero presentada en categorías –que componen los conceptos mostrados en la ola- y sus puntajes correspondientes. A su vez cada punto evaluado tiene una ponderación otorgada por Forrester. Las calificaciones oscilan en la escala de 0 a 5.

	Forrester's Weighting	Acquia	Adobe	Ektron	EPiServer	HP	IBM	OpenText	Oracle	SDL	Sitecore
CURRENT OFFERING	50%	2.57	4.03	2.58	2.17	3.33	3.51	2.93	3.19	2.91	3.45
Content authoring and workflow	5%	3.00	4.00	4.00	3.00	3.00	3.00	4.00	3.00	4.00	4.00
Content management	5%	3.00	3.00	3.00	3.00	4.00	4.00	4.00	4.00	3.00	3.00
Content tagging and taxonomies	5%	3.00	4.00	3.00	2.00	4.00	3.00	4.00	3.00	4.00	4.00
Digital asset management	5%	1.00	3.00	1.00	1.00	4.00	3.00	5.00	3.00	2.00	2.00
Globalization and localization	5%	4.00	3.00	3.00	3.00	3.00	4.00	3.00	3.00	5.00	4.00
Profiling and segmentation	2%	4.00	5.00	3.00	2.00	4.00	4.00	3.00	4.00	3.00	4.00
Site design	3%	3.00	5.00	3.00	3.00	4.00	4.00	3.00	4.00	4.00	4.00
Content targeting and preview	3%	3.00	5.00	2.00	3.00	3.00	3.00	2.00	3.00	3.00	4.00
Multichannel delivery	2%	3.00	4.00	3.00	4.00	4.00	4.00	3.00	3.00	4.00	4.00
Multisite support	2%	4.00	4.00	4.00	3.00	4.00	4.00	4.00	4.00	4.00	4.00
Polling, feedback, and user-generated content	3%	4.00	4.00	4.00	3.00	4.00	4.00	3.00	3.00	4.00	4.00
Mobile apps	5%	2.00	4.00	3.00	1.00	3.00	2.00	2.00	2.00	1.00	3.00
Mobile Web	5%	4.00	5.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00
Performance optimization	5%	3.00	3.00	2.00	2.00	3.00	3.00	3.00	2.00	3.00	3.00
Testing	10%	1.00	5.00	1.00	1.00	4.00	1.00	1.00	3.00	1.00	3.00
Analytics	10%	3.00	5.00	3.00	2.00	3.00	5.00	2.00	3.00	3.00	4.00
Marketing suite integration	10%	2.00	5.00	2.00	2.00	2.00	5.00	3.00	4.00	3.00	4.00
Commerce suite integration	5%	2.00	4.00	2.00	2.00	2.00	5.00	2.00	5.00	2.00	3.00
Customer communication management integration	2%	0.00	0.00	0.00	0.00	5.00	0.00	4.00	0.00	1.00	0.00
CRM integration	2%	3.00	3.00	3.00	2.00	3.00	2.00	2.00	3.00	3.00	3.00
Privacy and preferences	2%	4.00	3.00	3.00	2.00	4.00	4.00	3.00	1.00	3.00	3.00
Extranets	2%	1.00	2.00	4.00	3.00	4.00	4.00	4.00	3.00	4.00	4.00
Intranets	1%	2.00	2.00	3.00	2.00	2.00	4.00	3.00	3.00	2.00	2.00
Enterprise portal integration	1%	2.00	2.00	3.00	1.00	2.00	5.00	5.00	5.00	2.00	3.00
Product information	0%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Figura 6 - Calificaciones por cada punto evaluado. Fuente: Informe Forrester febrero 2015 [4]

	Forrester's Weighting	Acquia	Adobe	Ektron	EPiServer	HP	IBM	OpenText	Oracle	SDL	Sitecore
STRATEGY	50%	3.70	3.85	2.45	2.65	2.80	2.90	3.10	2.25	3.05	3.65
Product architecture	10%	3.00	4.00	2.00	3.00	3.00	3.00	3.00	3.00	4.00	3.00
Mobile strategy	25%	2.00	5.00	2.00	1.00	4.00	4.00	4.00	3.00	2.00	4.00
Cloud strategy	25%	5.00	2.00	2.00	3.00	1.00	1.00	3.00	0.00	3.00	3.00
API strategy	5%	5.00	3.00	4.00	3.00	3.00	3.00	3.00	3.00	3.00	4.00
Component ecosystem	10%	5.00	3.00	2.00	3.00	2.00	3.00	2.00	3.00	3.00	4.00
Partner strategy	5%	2.00	5.00	1.00	2.00	2.00	2.00	2.00	3.00	3.00	4.00
Reference customer assessment	20%	4.00	5.00	4.00	4.00	4.00	4.00	3.00	3.00	4.00	4.00
MARKET PRESENCE	0%	2.75	3.95	1.75	2.60	3.15	3.25	2.70	2.70	1.70	3.15
Product customer count	15%	2.00	2.00	2.00	3.00	2.00	2.00	1.00	1.00	1.00	2.00
Product revenue	40%	2.00	5.00	1.00	2.00	3.00	4.00	3.00	3.00	2.00	3.00
Product revenue growth	30%	5.00	4.00	3.00	3.00	3.00	2.00	3.00	2.00	1.00	4.00
Global presence	15%	1.00	3.00	1.00	3.00	5.00	5.00	3.00	5.00	3.00	3.00

Figura 7 - Calificaciones por cada punto evaluado. Fuente: Informe Forrester febrero 2015 [4]

En su informe más reciente sobre WCMS que data de enero de 2017 [13], la ola de Forrester resultó ser la que se muestra en la Figura 9.

Se desprende que AEM sigue como líder de mercado, además de contar con la mayor presencia de mercado, según la escala proporcionada.

Sin embargo para el caso de Sitecore, la situación ha tenido un pequeño retroceso en comparación con el informe anterior. Sigue manteniendo una muy buena presencia en el mercado, pero ha pasado a perder la categoría de líder que ostentaba previamente, además de ser superado por algunos competidores -que pasaron a ser líderes-. Sin embargo, la posición que ostenta es muy buena de todas formas, siendo todavía de las mejores opciones en el mercado.

Como se presentó previamente para el informe del 2015, se presentan las calificaciones hechas por Forrester para cada plataforma evaluada -Figura 9-.

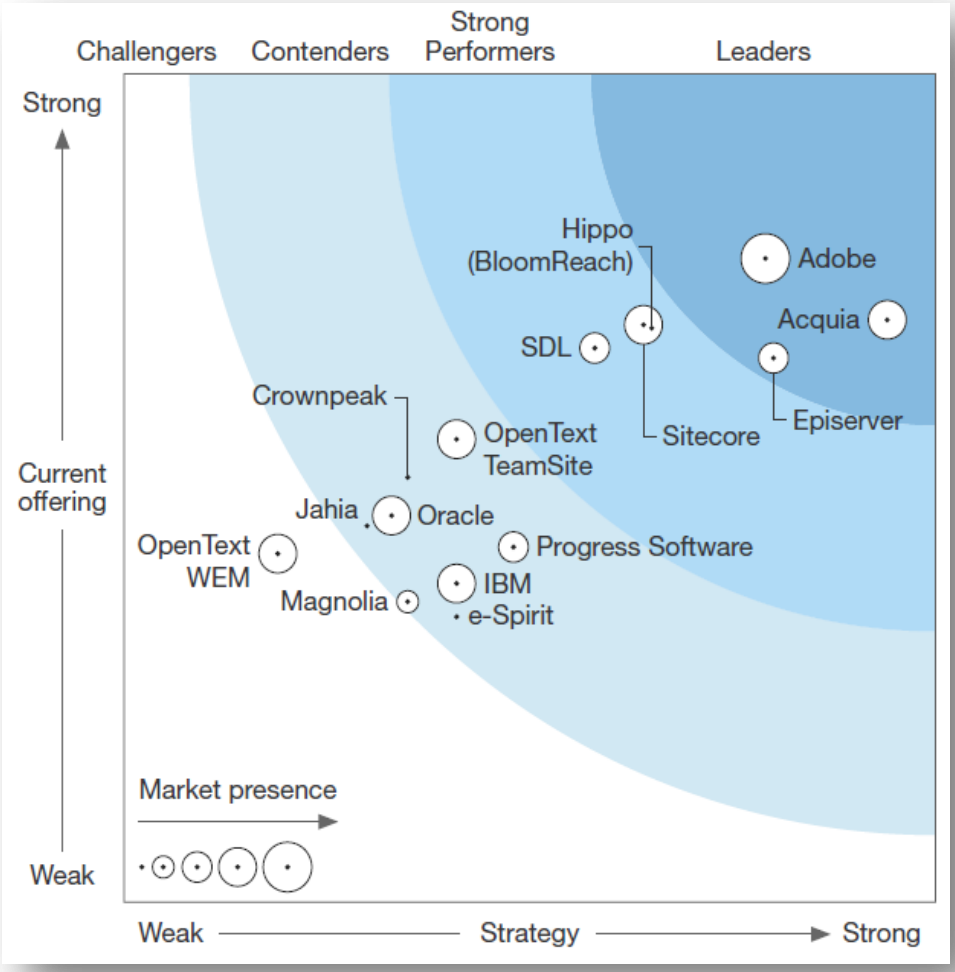


Figura 8 - Ola de Forrester. Fuente: Informe Forrester enero 2017 [94].

Por otra parte Gartner también ha publicado informes sobre productos WCMS, presentando su cuadrante mágico, el cual posee cuatro categorías en donde los productos evaluados son ubicados según la habilidad de ejecución y completitud. El último informe publicado fue en setiembre de 2016 [15] -Figura 11-, adicionalmente se cuenta con el informe correspondiente a julio del 2015 [14] -Figura 10-.

	Forrester's weighting	Acquia	Adobe	Crownpeak	Episerver	e-Spirit	Hippo (BloomReach)	IBM	Jahia	Magnolia	OpenText TeamSite	OpenText Web Experience Mgmt.	Oracle	Progress	SDL	Sitecore
Current offering	50%	3.50	3.87	2.55	3.27	1.71	3.45	1.91	2.26	1.80	2.78	2.09	2.32	2.13	3.33	3.47
Content	30%	3.90	4.50	3.00	2.60	2.30	4.00	2.00	2.70	2.20	3.10	2.30	2.30	2.60	3.60	3.70
Operations	25%	3.60	4.40	1.60	4.00	1.60	3.40	2.40	2.20	1.00	3.00	2.40	2.80	1.60	3.60	4.20
Architecture	30%	3.70	2.70	3.70	3.40	1.60	3.60	1.60	2.30	2.50	2.30	2.00	2.40	2.20	3.80	2.80
Extensions	15%	2.10	4.05	0.95	3.10	0.90	2.10	1.50	1.40	0.95	2.70	1.35	1.40	1.90	1.40	3.10
Strategy	50%	4.70	3.95	1.75	4.00	2.05	3.25	2.05	1.50	1.75	2.05	0.95	1.65	2.40	2.90	3.20
Vision	35%	5.00	5.00	2.00	5.00	3.00	3.00	3.00	3.00	2.00	3.00	1.00	3.00	3.00	3.00	4.00
Cloud	20%	4.00	1.00	3.00	3.00	1.00	5.00	1.00	0.00	1.00	1.00	0.00	2.00	2.00	3.00	1.00
Service partner program	20%	5.00	5.00	1.00	3.00	1.00	1.00	3.00	1.00	3.00	3.00	3.00	1.00	1.00	3.00	3.00
Pricing transparency	5%	3.00	0.00	1.00	1.00	0.00	0.00	0.00	1.00	1.00	0.00	0.00	0.00	3.00	1.00	0.00
Developer program	20%	5.00	5.00	1.00	5.00	3.00	5.00	1.00	1.00	1.00	1.00	0.00	0.00	3.00	3.00	5.00
Market presence	0%	4.00	5.00	1.00	3.00	1.00	1.00	4.00	1.00	2.00	4.00	4.00	4.00	3.00	3.00	4.00
Product customer count	50%	5.00	5.00	1.00	3.00	1.00	1.00	3.00	1.00	1.00	3.00	3.00	3.00	3.00	3.00	5.00
Global presence	50%	3.00	5.00	1.00	3.00	1.00	1.00	5.00	1.00	3.00	5.00	5.00	5.00	3.00	3.00	3.00

Figura 9 – Calificaciones por cada punto evaluado. Fuente: Informe Forrester enero 2017. [94]

En ambas figuras -Figura 10 y Figura 11 - se observa que, pese a pequeñas variaciones, Sitecore y Adobe AEM se mantienen como líderes según Gartner.
Las cuatro evaluaciones muestran con bastante certeza que en la actualidad Adobe AEM y Sitecore son los productos WCMS líderes en el mercado –en particular AEM está avalado en los cuatro informes

estudiados-, validando las hipótesis planteadas sobre las plataformas a usar en este proyecto –como fue mencionado al comienzo de este informe.

Figura 10 - Cuadrante mágico de Gartner. Fuente: Informe Gartner, julio 2015

Figura 11 - Cuadrante mágico de Gartner. Fuente: Informe Gartner, setiembre 2016

16 Futuro del WCMS

El manejo de contenido está en estado de cambio constante, por lo tanto tampoco se puede precisar con certeza pero si con cierta seguridad algunas características sobre el tema a futuro. Algunas de ellas son las siguientes:

16.1 Los WCMS de código abierto tendrán menos auge

El Mercado de los WCMS de código abierto ya está sobrepoblado. Muy pocas plataformas de código abierto han conseguido envión en los últimos cinco años, especialmente en comparación a los anteriores cinco años.

El motivo que sostiene y hace crecer un proyecto de código abierto es la habilidad para atraer desarrolladores, y el entusiasmo de un nuevo sistema es rápidamente templado por la falta de una comunidad y base instaladas en las cuales el sistema podría crecer y ser probada.

Lo que lleva al nacimiento de nuevos sistemas es la adopción de nuevos *frameworks* para la web. Cada vez que un nuevo lenguaje o paradigma de programación consigue empuje, un puñado de proyectos de código abierto resultará de los primeros proyectos escritos para esas plataformas.

16.2 Volverá el desacoplamiento

Aunque en el presente predomine el uso de una arquitectura acoplada, la tendencia es a volver a la arquitectura desacoplada. La interacción necesaria en el ambiente de entrega se ha separado del WCMS –es servido a través de sistemas que pueden no necesitar interacción con el WCMS-. Por ejemplo, agregar comentarios a un sitio web es tan simple como agregar un sistema como Disqus, IntenseDebate o incluso Facebook. Tecnologías del lado del cliente han avanzado al punto de que la integración de plataformas de redes sociales depende solo de una inclusión de JavaScript. Los vendedores de *marketing* automatizado se pueden integrar de manera independiente del lado del cliente, y aún el testeo A/B está disponible virtualmente sin cambios en las plantillas.

Lo descrito previamente significa que las experiencias de usuario pueden ser llevadas a cabo sobre archivos HTML estáticos. Los archivos HTML con el contenido simplemente juegan el rol de alojar una gran cantidad de interacción provista por otros servicios.

Inclusive la relación entre las tecnologías generadas por el servidor y el cliente pueden revertirse por completo. Los *frameworks full stack* JavaScript como Angular JS, React y Dojo están dando surgimiento a las "aplicaciones de una sola página", donde lo que es liberado al cliente no es contenido, sino una aplicación que corre en un navegador y puede obtener contenido basado en el comportamiento del usuario. En estos casos, en vez de que el contenido brinde las funcionalidad al cliente, se tiene lo apuesto (la funcionalidad del cliente está brindando el contenido).

Todos los cambios mencionados están reduciendo significativamente las ventajas que proveían los WCMS acoplados. La publicación desacoplada trae ventajas únicas en términos de escalabilidad, estabilidad y seguridad. A su vez, los entornos de liberación se están convirtiendo cada vez más simples para crear y mantener.

16.3 Crecimiento en el foco en las herramientas de marketing e integración

Cada combinación de un visitante y contenido es una nueva experiencia que necesita ser gestionada. Aunque el contenido quizás no cambie, se necesita optimizar la interacción del visitante a través de un conjunto de herramientas de *marketing* y optimización.

Cómo reaccionará la industria al respecto aún es una interrogante, hay dos opciones posibles. Una es que se desarrolle una suite de automatización de *marketing* y experiencia de usuario dentro del mismo WCMS. Otra alternativa es la integrar un sistema externo.

16.4 SaaS de nivel de entrada conquistará la parte más baja del mercado

La expresión nivel entrada se refiere a los productos SaaS que pueden ser establecidos mediante un pago y algunos datos personales.

Estos sistemas ganarán la parte baja del mercado del WCMS, en la medida que las organizaciones decidan que el nivel de gestión que ofrecen estos sistemas es suficientemente bueno y se concentren en la optimización del *marketing*.

Aunque muchos de estos sistemas son limitados en cuanto a sus prestaciones, excederán la necesidad de varias organizaciones. En caso de no ser así, también les favorece el hecho de bajo costo, haciendo que las empresas que tengan mayores demandas, las recorten o posterguen.

Como se mencionó, habrá organizaciones que requieran simplemente un sitio web, mientras otras requerirán cada vez más personalización. Por lo tanto el éxito de este tipo de plataformas radicará en la habilitad de mantener un balance en las necesidades de una gran cantidad de usuarios con distintos rangos de necesidades.

16.4.1.1 Crecimiento de distribución multicanal

El surgimiento y crecimiento de las redes sociales ha forzado el tema de los canales múltiples. Grandes volúmenes de contenido que están siendo creados nunca verán el interior del sitio web de la organización. El ciclo de vida del contenido transcurrirá en plataformas externas y distribuidas.
Lo que está surgiendo es el ascenso de los "WCMS de redes sociales", término asignado a las plataformas que gestionan actualizaciones de multimedia social.
En el futuro se avizora a más empresas alentando la gestión de multimedia social desde sus productos WCMS. Algunas empresas han llevado a cabo a través de *add-ons* y subsistemas, y otros lo han hecho tratando a una actualización de multimedia social como un objeto de contenido como cualquier otro, sujeto a *workflow*, permisos, auditorías, etc. Cuando el contenido es publicado, no aparece en el sitio web, sino que es agregado a las plataformas de red social a través de varias APIs.

16.5 El consumo de contenido distribuido comenzará a crecer

Los servicios están empezando a ser dirigidos directamente al proceso de creación editorial y de contenido. Algunas plataformas se centran en la gestión de calendarios editoriales, creación de tareas y colaboración. Han separado efectivamente ese conjunto de funcionalidades del WCMS y se concentran en él específicamente.
Algunas organizaciones recurrirán esos servicios como herramientas de proceso editorial dentro de sus WCMSs en breve. Los equipos comenzarán a trabajar en el laborioso proceso de creación de contenido, con una llamada a una API que crea o actualice el contenido en el WCMS en el último momento. Adicionalmente, las organizaciones demandarán cada vez más el consume de contenido de forma distribuida.
En contrapartida, las organizaciones de igual manera es probable que comiencen a distribuir su manejo de contenido, teniendo múltiples sistemas internos pero un solo WCMS externo que distribuye todo el contenido. Por ejemplo, los autores internos de blogs, pueden escribir sus mensajes en una instalación privada de WordPress y luego envía el contenido a un WCMS de *marketing* más robusto para su distribución.
A la fecha, la asunción por defecto de un vendedor de WCMS es que todo el contenido empezará su ciclo en WCMS. Esto comenzará a cambiar a medida que los vendedores se den cuenta y se adapten al hecho que otras plataformas están manejando el proceso editorial con más foco y funcionalidad.

17 Referencias

[1] Bob Boiko. *Content Management Bible, second edition.* Wiley Publishing, Inc, 2005.
[2] Deane Baker. *Web Content Management. Systems, features, and best practices.* O´Reilly, 2016.
[3] https://www.drupal.org/. Último acceso: 29/1/2017.
[4] http://www-03.ibm.com/software/products/es/category/enterprise-content-management. Último acceso: 29/1/2017.
[5] http://www.opentext.com/what-we-do/products/enterprise-content-management. Último acceso: 29/1/2017.
[6] http://documentum.opentext.com/. Último acceso: 29/1/2017.
[7] https://wordpress.com/. Último acceso: 27/1/2017.
[8] https://techversyssolutions.wordpress.com/tag/wordpress-cms-2/. Último acceso: 31/1/2017.
[9] https://builtwith.com Último acceso: 6/12/2016.
[10] http://www.gartner.com/. Último acceso: 29/1/2017.
[11] https://go.forrester.com/. Último acceso: 29/1/2017.
[12] Ted Schadler. The Forrester Wave™ : Web Content Management Systems, Q1 2015. Forrester Research, Inc., 2015.
[13] Mark Grannan. The Forrester Wave™ : Web Content Management Systems, Q1 2017. Forrester Research, Inc., 2017.
[14] Mick MacComascaigh, Jim Murphy. *Magic Quadrant for Web Content Management*. Gartner, Inc., 2015.
[15] Mick MacComascaigh, Jim Murphy. *Magic Quadrant for Web Content Management*. Gartner, Inc., 2016.

Anexo 2 Documentación técnica de AEM

Índice

1 Marco Historico

Day Software fue una compañía suiza de software con especialización en gestión de contenidos fundada en el año 1993. Una de sus creaciones más relevantes constituye la infraestructura de gestión de contenidos llamada CRX, la cual está basada en JAVA. Asimismo, estuvo involucrada en el desarrollo de la API de JAVA para administrar repositorios de contenidos (JCR API). Mediante este producto realizó una gran contribución para facilitar los procesos de estandarización y proyectos *open source,* tales como: Apache Jackrabbit y Apache Sling -los cuales se convertirían en componentes base para su plataforma WCMS, Day CQ-.

La empresa fue adquirida por Adobe Systems el día 28 de julio del año 2010 por una suma total de 240 millones de dólares, tomando control de la plataforma WCMS y lanzando un año después la primera versión Adobe CQ 5.4. A partir del año 2013 Adobe decide modificar el nombre del producto, denominándolo Adobe Experience Manager (a partir de ahora AEM), en su primera versión 5.6. En la actualidad se posicionan como líderes en el mercado de los WCMS.

AEM se integra así a la familia Adobe Marketing Cloud: una colección de productos y servicios ofrecidos por dicha empresa para brindar una solución integral de marketing digital –la cual no sólo ofrece la gestión de contenido web sino que involucra además a distintos productos para el análisis de datos, creación de campañas y análisis de redes sociales, entre otros-.

Es importante señalar que para acceder a estos productos se deben adquirir licencias individuales. La licencia del producto AEM tiene un costo por encima de los 200.000 dólares anuales para un único servidor (cualquier proyecto desarrollado en AEM necesita un mínimo de dos servidores para ejecutar su plataforma). Fue con fines académicos y con el propósito de implementar el caso de estudio que nuestro cliente, Conexio, nos facilitó una licencia.

Actualmente el producto AEM se encuentra en la versión 6.2, la cual fue lanzada el día 21 de abril del presente año. Ésta será la utilizada en nuestro caso de estudio.

2 Arquitectura

AEM es un WCMS que se autodefine [3] como un sistema cliente-servidor ofreciendo una plataforma web cuya función oscila entre la construcción, gestión y el despliegue de sitios web comerciales, así como servicios relacionados. Combina una serie de funciones a nivel de infraestructura y de aplicación agrupadas en un único paquete integrado.

Es una plataforma que permite entregar contenido digital a través de distintos canales y proveer un entorno propicio para los autores cuenten con soporte para realizar edición en línea de contenido.

Está integrado por los siguientes tres macro componentes:

- **Web Application Server**

AEM puede ser desplegado en modo *standalone* con un servidor web *Jetty* integrado, el cual utilizaremos para implementar el caso de estudio o como una aplicación con cualquier servidor web –utilizado para ambientes de producción-.

- **Web Application Framework**

  AEM incorporó el *framework* Sling que simplifica la escritura de servicios *RESTful* y las aplicaciones web orientadas al contenido.

- **Repositorio de Contenido**

  AEM incluye un JAVA Content Repository (a partir de ahora JCR), una base de datos especializada para persistir contenido.

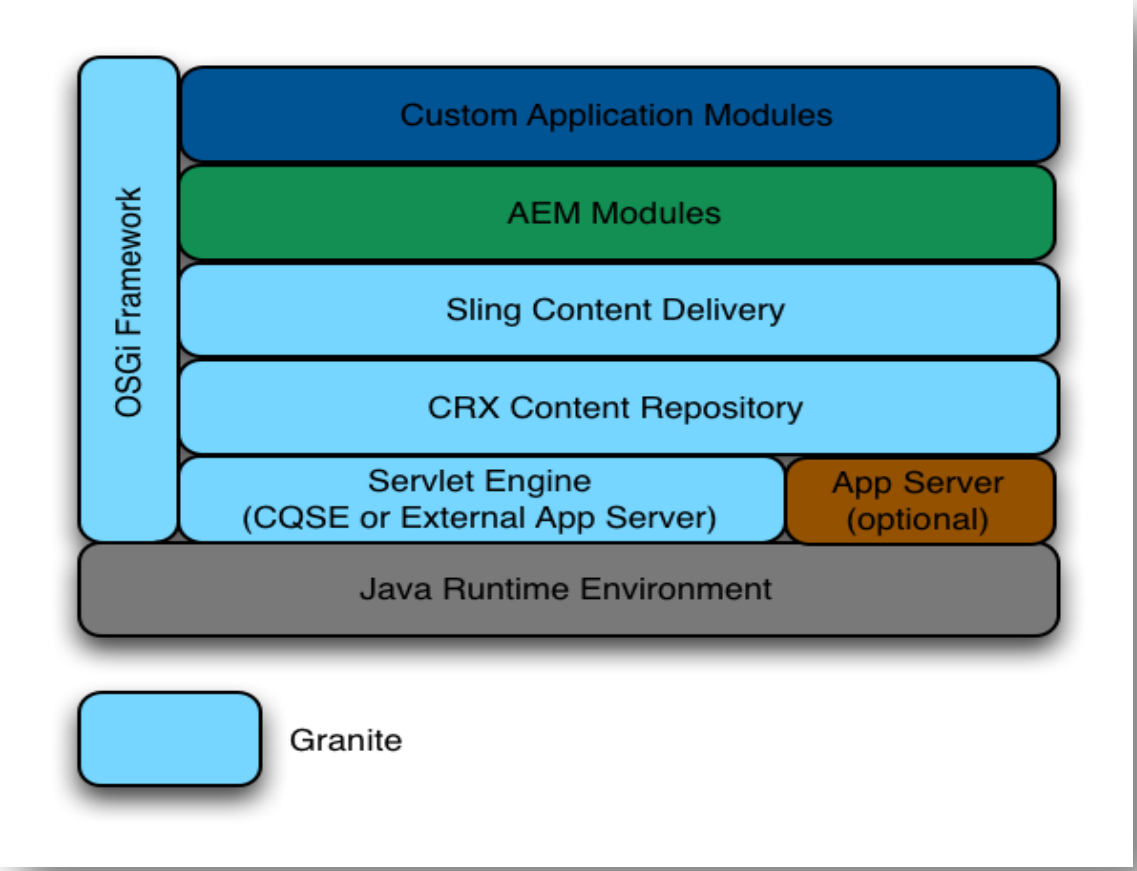


Figura 1– Esquema de arquitectura de AEM. Fuente: [13]

Si detallamos el *stack* de tecnologías [2] que utiliza AEM, partiendo de la base de la Figura 1 encontramos la plataforma JAVA, cuya versión recomendada para la versión de AEM 6.2 es JAVA 1.8

2.1 Granite

Constituye la base técnica para lograr el funcionamiento de AEM, sincronizando e integrando las diferentes tecnologías que componen el *stack* del mismo.

2.2 OSGi framework

Define la arquitectura para desarrollar y desplegar aplicaciones modulares y bibliotecas. Los contenedores OSGi permiten descomponer la aplicación en módulos individuales (son archivos *jar* que cuentan con meta información y son llamados paquetes en terminología de OSGi). También ofrece la capacidad de poder cargar o descargar dichos componentes (que llamaremos *bundles*) sin la necesidad de reiniciar el servidor.

2.3 Repositorio de contenido CRX

Absolutamente todos los datos dentro de AEM se almacenan en el repositorio de contenido CRX, el cual constituye una implementación creada por Adobe de la API JCR 2.0 (JCR). A su vez, parte del código base de CRX está basado en el proyecto *open source* de Apache Jackrabbit.

Un repositorio basado en la API JCR es un tipo específico de base de datos diseñado para el acceso de datos jerárquicos -siendo éstos últimos no estructurados o semi-estructurados-. De esta manera, combina las características de una base de datos relacional tradicional con los de un sistema de archivos convencional.

Dentro de las funciones que incluyen este tipo de repositorios se encuentran:

- **Jerarquía**
  El contenido en un repositorio JCR puede ser direccionado mediante la ruta. Esto resulta sumamente útil cuando se trata de enviar el contenido a la web, ya que la mayoría de los sitios también están organizados de forma jerárquica.
- **Contenido semi-estructurado**
  Los repositorios JCR pueden persistir documentos como XML, HTML, JSP o cualquier otro tipo, ya sea como archivos opacos (como un sistema de archivos haría) o como estructuras integradas directamente en la jerarquía JCR.
- **Control de acceso y bloque**
  Los JCR pueden restringir el acceso a las diferentes partes del contenido basándose en políticas jerárquicas llamadas Access Control List (ACLs), además de soportar bloqueos de contenidos para prevenir conflictos.

Las funciones que brindan los repositorios JCR similares a una base de datos relacional son:

- **Acceso mediante *querys***
  Los JCR soportan el acceso mediante querys como SQL.
- **Contenido Estructurado**
  Pueden definir restricciones sobre las estructuras de datos de acuerdo a un esquema.
- **Integridad Referencial**
  JCR puede exigir la integridad referencial entre los elementos de contenido.
- **Transacciones**

Las interacciones con un repositorio JCR pueden encapsularse en transacciones y realizar un *roll-back* cuando sea necesario.

Por otra parte, los JCR ofrecen los siguientes servicios que las aplicaciones orientadas al contenido a menudo necesitan, pero que ni los sistemas de archivos ni las bases de datos relacionales ofrecen:

- **Contenido no estructurado**
  JCR también puede soportar estructuras dinámicas de datos arbitrarios sin restricciones de esquema.
- **Búsqueda por texto**
  JCR soporta búsqueda por texto de contenido.
- **Orden**
  Los elementos dentro de la jerarquía pueden mantener su orden, si se desea.
- **Observación**
  Los clientes de la API pueden registrar *listeners* para reaccionar a los cambios realizados en el repositorio.
- **Control de versiones**
  JCR admite un sistema de control de versiones avanzado para el contenido del repositorio.

2.4 Estructura Repositorio CRX

- **/apps**
  En esta carpeta se persiste todo lo relacionado a la aplicación en particular que se está desarrollando, incluye la definición de componentes específicos del sitio web, la definición de templates, configuración, etc.

- **/content**
  En esta carpeta se persiste todo el contenido creado para el sitio web.

- **/etc**
  En esta carpeta se encuentran diferentes configuración globales de la instancia y del sitio web, puede incluir contenido global.

- **/home**
  Se persisten los usuarios y grupos del sistema.

- **/libs**
  En esta carpeta se encuentran las definiciones pertenecientes al núcleo de AEM. Las sub carpetas que se encuentran incluidas contienen muchas funcionalidades y componentes que trae por defecto la plataforma. El contenido persistido en está carpeta nunca debería ser modificado por que podrían afectar el comportamiento de AEM.

- **/tmp**
  Se guardan los archivos temporales del sistema.

- **/var**
  Se persisten los archivos que son actualizados por el sistema como log, estadísticas, manejadores de eventos.

2.5 Sling Framework

Se trata de un *framework* para aplicaciones web basada en los principios *REST* que proporcionan un fácil desarrollo de aplicaciones orientadas a contenidos. Sling utiliza un repositorio JCR, como Apache Jackrabbit; o bien, en el caso de AEM, el repositorio de contenido CRX como su base de datos para administrar, almacenar y enviar contenido.

El uso de Sling permite que se tenga como principal consideración el resultado obtenido a partir de la URL, considerando si se encuentra una secuencia de comandos o archivo para llevar a cabo la renderizacion. Esto proporciona un excelente soporte para los autores de contenido web a la hora de crear páginas que son fácilmente adaptables a sus necesidades.

Como vemos en la Figura 2, Sling descompone la URL para buscar el recurso requerido por el pedido, descomponiendo la misma en *path*, selector, extensión entre otros para poder ubicar el contenido.
Luego, busca en primer lugar en la carpeta /apps y en caso de no encontrar el recurso, lo busca como segunda opción en la carpeta /libs -ambas carpetas existen en AEM-. La primera es donde el desarrollador ubica sus componente creados específicamente para un sitio web, mientras que la ubicación /libs se utiliza para guardar contenido específico de AEM que viene pre instalado con la plataforma.

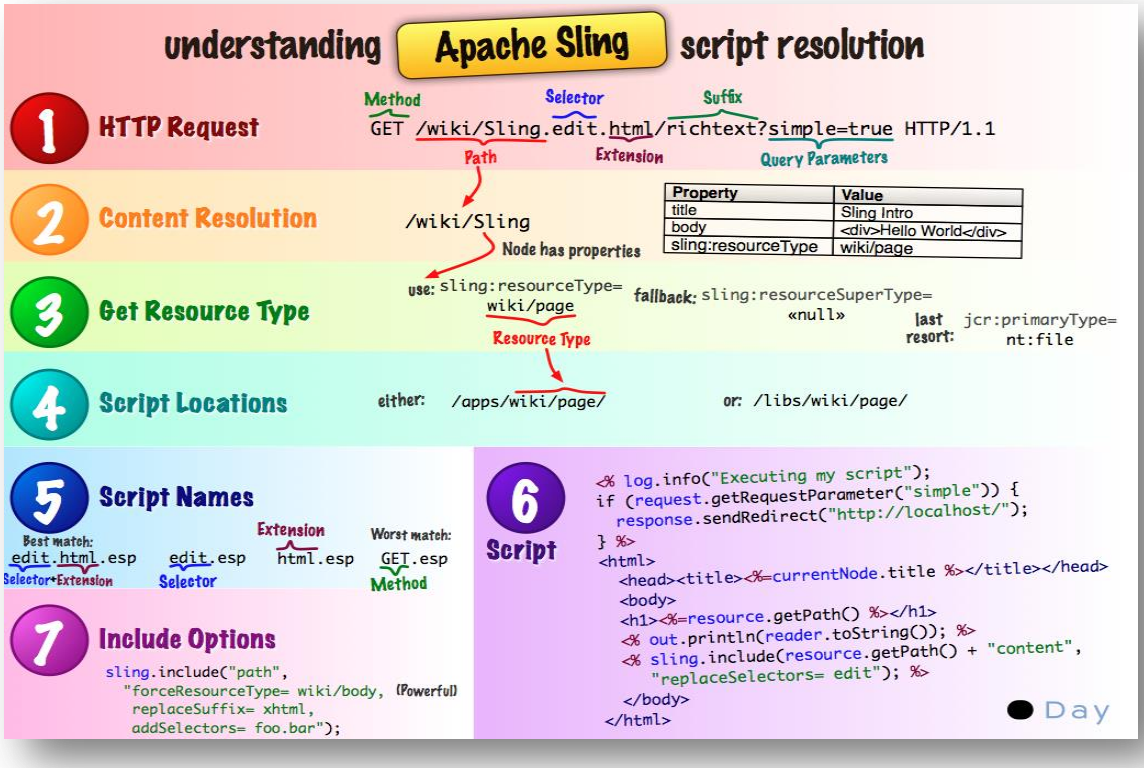


Figura 2 – Esquema de funcionamiento de Sling. Fuente: [14]

3 Conceptos básicos

Debido al gran tamaño de la plataforma AEM -y considerando que algunas de las funcionalidades que ofrece quedaron por fuera del alcance del proyecto- nos centraremos en definir los conceptos y herramientas que atañen tanto a la construcción de componentes como a la creación de contenidos que ofrece la plataforma.

Definiremos en primer lugar algunos conceptos básicos para poder luego profundizar sobre la temática.

3.1 Ítem, Nodos y Propiedades

Comenzaremos definiendo los tres conceptos básicos del repositorio de AEM.
Un ítem [4] es la unidad básica del repositorio, es decir, una interfaz que tiene definida dos implementaciones: nodos y propiedades.
Los nodos [4] definen la estructura del contenido, pueden tener cero o más ítems como hijos. La propiedad más relevante de los nodos es *jcr: primary Type* la cual define el tipo de nodo (es especificado en el momento de creación del nodo y es inmutable). AEM trae un conjunto de tipos de nodos ya definidos, mediante configuración avanzada es posible crear nuevos tipos. En los ambientes de Sling un nodo también es denominado como un recurso.

Las propiedades [4] en cambio, persisten los datos específicos y la metada del contenido, una propiedad no puede tener hijos pero puede tener cero o más valores definidos. Las propiedades tienen ciertos atributos que deben ser definidos:

- ***Name***
  El nombre de una propiedad la identifica de forma local al nodo padre.
- ***Type***
  Define el tipo de dato que va a ser persistido por la propiedad, por ejemplo *String*, *Long*, *Double* y *Date*, entre otros, dichos tipos son brindados por AEM.

En la Figura 3 vemos una sección de la interfaz gráfica que ofrece AEM del repositorio CRX, notamos el nodo *text* que contiene varios ítems hijos, a su vez podemos ver las distintas propiedades que tiene definida.

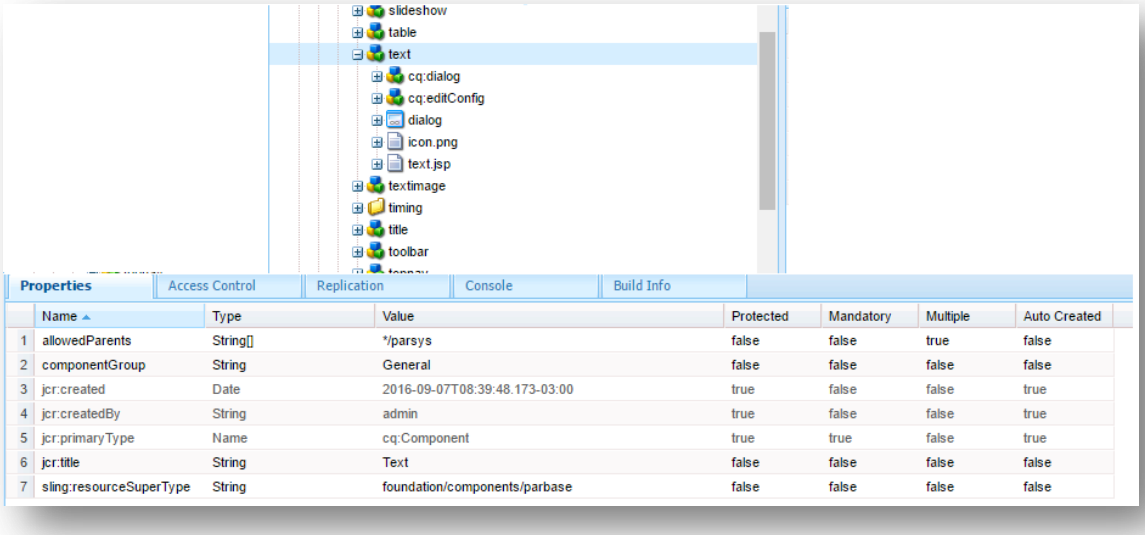


Figura 3 – Captura de pantalla recortada de la interfaz web del CRX. Fuente: Autoría propia.

3.2 Componente

Un componente de *software* es un elemento del sistema que ofrece determinado servicio o evento y es capaz de comunicarse con otros componentes. En cambio, un componente de contenido es considerado desde la plataforma AEM, es un nodo que agrupa *scripts*, archivos, datos y propiedades, que tienen como objetivo renderizar y administrar una porción de contenido. Cuando el recurso es una página, los componentes son encargados de renderizar la página completa. Este tipo de componente es denominado de página (aunque los componentes no necesariamente renderizan contenido).

La definición de un componente con renderizado incluye:

- El código utilizado para renderizar el contenido, ya sea un JSP o HTML.
- Un diálogo para las entradas de usuarios utilizadas para la configuración del contenido del componente.

En el proyecto nos centraremos en los componentes desarrollados para que sean utilizados por los autores; estos forman parte fundamental de un proyecto WCMS y pueden variar desde simples títulos hasta un navegador complejo con varios niveles de paginación. Aunque AEM ofrece la ventaja de tener un conjunto pre definido de componentes definidos y listos para su uso, en la mayoría de los casos no se ajustan a los requerimientos del sitio web. De todas formas, resultan útiles para ejemplificar el funcionamiento *per se* de la plataforma o extender su funcionalidad-como se verá más adelante-.

En la Figura 4 podemos observar como se ve un componente pre definido por AEM en el entorno de autores llamado *Text and Image,* sin ningún contenido definido. Este componente permite al autor, como sugiere su nombre, establecer una imagen y un texto para mostrar en una página. Los pequeños íconos con diferentes herramientas que vemos en la imagen se

despliegan al hacer click en el mismo, permitiéndonos copiarlo, cortarlo, eliminarlo o acceder a su configuración, entre otras funciones.

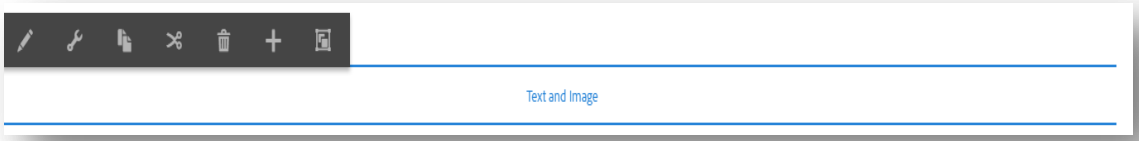


Figura 4 – Captura de pantalla del componente *Text and Image* en el ambiente de autor. Fuente: Autoría propia.

3.3 Widget

En AEM todas las entradas de usuarios son manejadas por un tipo de nodo denominado *widget*. Estos usualmente son utilizados para editar una pieza de contenido, y los diálogos son construidos mediante una combinación de los mismos. La plataforma cuenta con varios *widgets* definidos, los cuales son utilizados por el desarrollador para incluirlos en los diálogos, o bien, crearlos de forma personalizada. Podemos observar en las figuras Figura 5 y Figura 6 dos ejemplos de *widgets* de tipo imagen y texto respectivamente.

Figura 5 – Widget de tipo imagen. Fuente: Autoría propia.

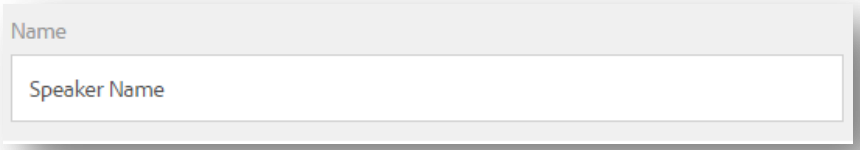


Figura 6 - Widget de tipo texto. Fuente: Autoría propia.

3.4 Diálogo

Un dialogo es un tipo especial de widget.

Para editar el contenido de un componente, AEM utiliza los diálogos definidos por los desarrolladores. Estos diálogos combinan una serie de *widgets* que les brinda a los usuarios un conjunto de campos relacionados al contenido, los cuales son editables. Los diálogos también pueden ser utilizados para editar metadata y configuración administrativa.

Continuando, en la Figura 7 podemos ver el diálogo asociado al componente *Text and Image* antes mencionado. Señalado mediante flechas rojas en la imagen podemos ver los distintos tipos de *widgets* que constituyen este diálogo en particular, permitiendo al usuario definir por ejemplo la imagen que desea mostrar en el componente, el título y la *URL* (interna o externa) asociada a la imagen.

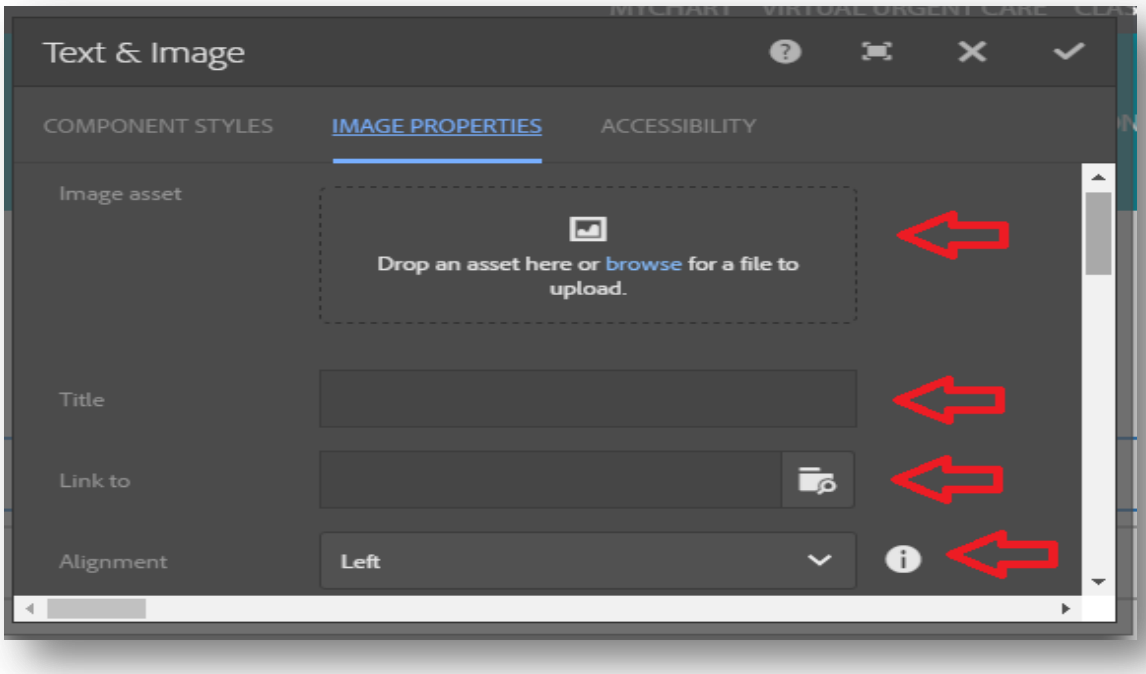


Figura 7 – Captura de pantalla del dialogo del componente Text and Image. Fuente: Autoría Propia

3.5 Template

Un *template* define un tipo de página. Al crearse una página en AEM debe seleccionarse un *template* de pertenencia. De esta forma, un *template* conforma una jerarquía de nodos que contienen la misma estructura que debe tener la página que está definiendo, pero sin ningún contenido definido. Así, siempre tienen asociados un componente de página (el cual se encarga del renderizado de la misma).

3.6 Página

Una página es una instancia de un *template*, teniendo una jerarquía de nodos cuya raíz es del tipo *cp:Page* y separadamente un nodo de contenido del tipo *cq:PageContent*. De esta forma se pueden crear plantillas HTML para que sean reutilizadas por las páginas. AEM incluye una definición de página (componente de página) dentro de la carpeta *libs,* es posible para el desarrollador construir un componente de página de cero, pero es una tarea engorrosa e innecesaria, ya que nos podemos basar en la definición que ofrece la plataforma. La misma ya viene con varias funcionalidades y propiedades útiles que, como explicaremos más adelante, pueden ser extendidas y sobre escritas, también define un dialogo para configurar dichas propiedades.

3.7 Paragraph System

El *Paragraph System*[5] (a partir de ahora *parsys*), es un componente que permite a los autores agregar otros tipos de componentes a una página. También permite mover, copiar, cortar y pegar los distintos componentes contenidos en el *parsys*. Es posible configurar los *parsys* para restringir qué componentes o qué grupo pueden ser incluidos. En la Figura 8, podemos ver un diálogo de configuración de un *parsys* y cómo es posible habilitar a los grupos o componentes individuales mediante un *checkbox*.

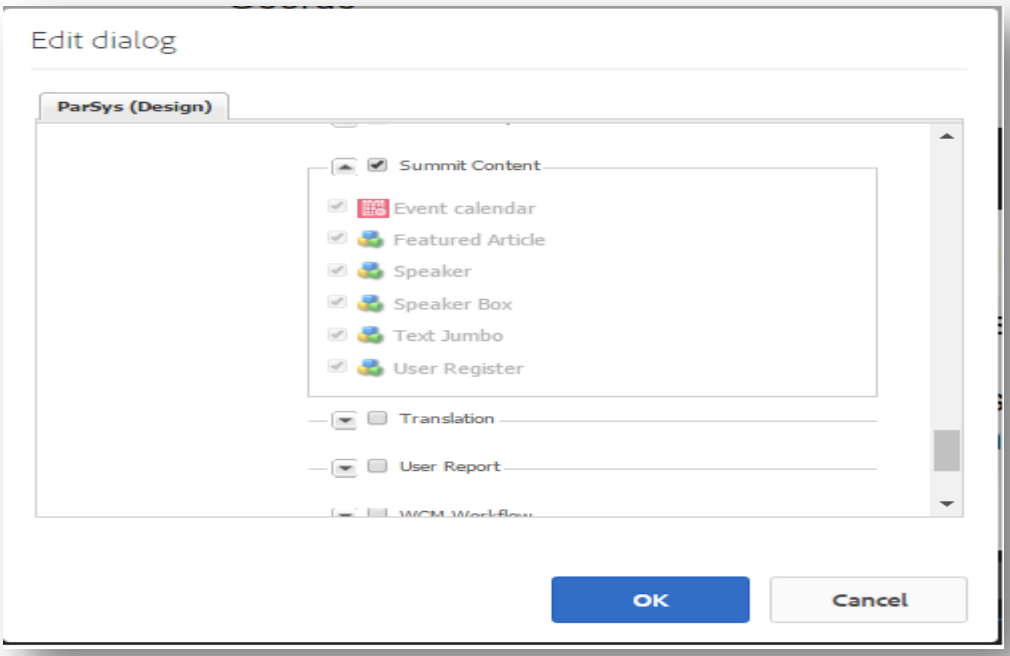


Figura 8 – Captura de pantalla del dialogo que ofrece AEM para la configuración de un parsys.

Fuente: Autoria Propia

En una página pueden existir diferentes *parsys*, cada uno de ellos es identificado con un nombre definido por el desarrollador. A su vez, cada uno de estos *parsys* puede ser configurado de forma individual. En la **¡Error! No se encuentra el origen de la referencia.** se

uede observar una página de AEM la cual tiene definida dos parsys-recuadros de bordes azules con el texto *'Drag components here'*-.

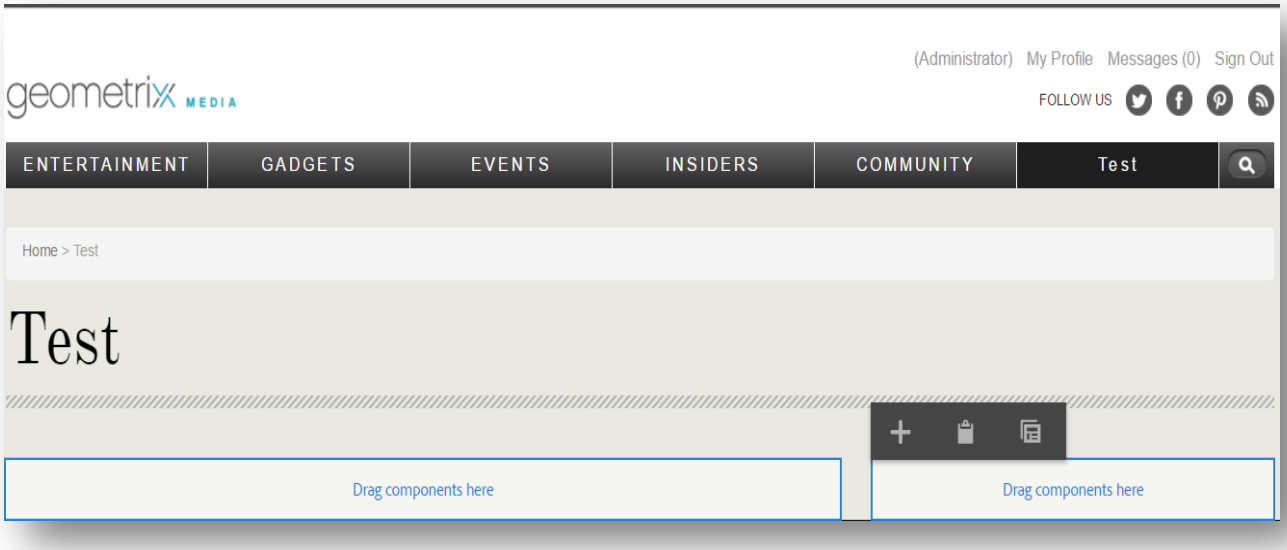


Figura 1 - Página de AEM en el ambiente de Autor con dos *parsys* definidos. Fuente: Proyecto Geometrixx de AEM [54]

3.8 Resumen

Resumiendo los conceptos mencionados, podemos observar en la **¡Error! No se encuentra el rigen de la referencia.** que los *templates* definen a las páginas, pudiendo existir varias de ellas. A su vez, las páginas pueden contener cero o más *parsys*, dentro de los cuales, los autores pueden insertar cero o más componentes. También podemos observar un diagrama que ejemplifica el modelado de conceptos -Figura 10-.

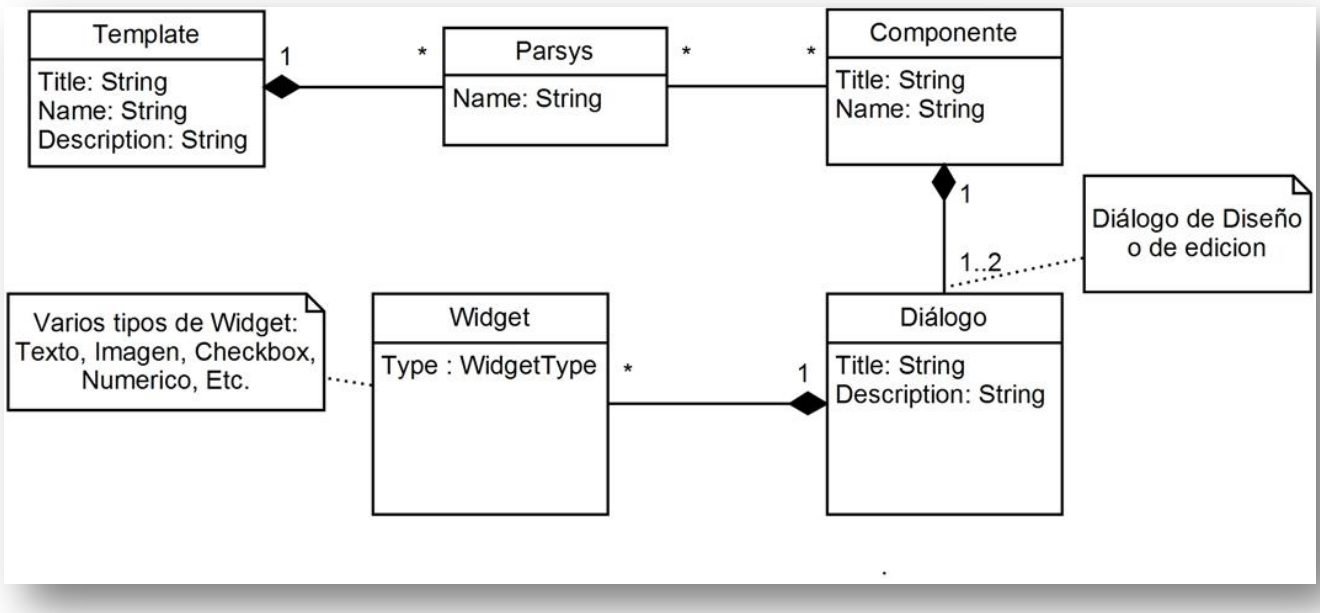


Figura 9 – Modelado de conceptos de AEM. Fuente: Autoría Propia

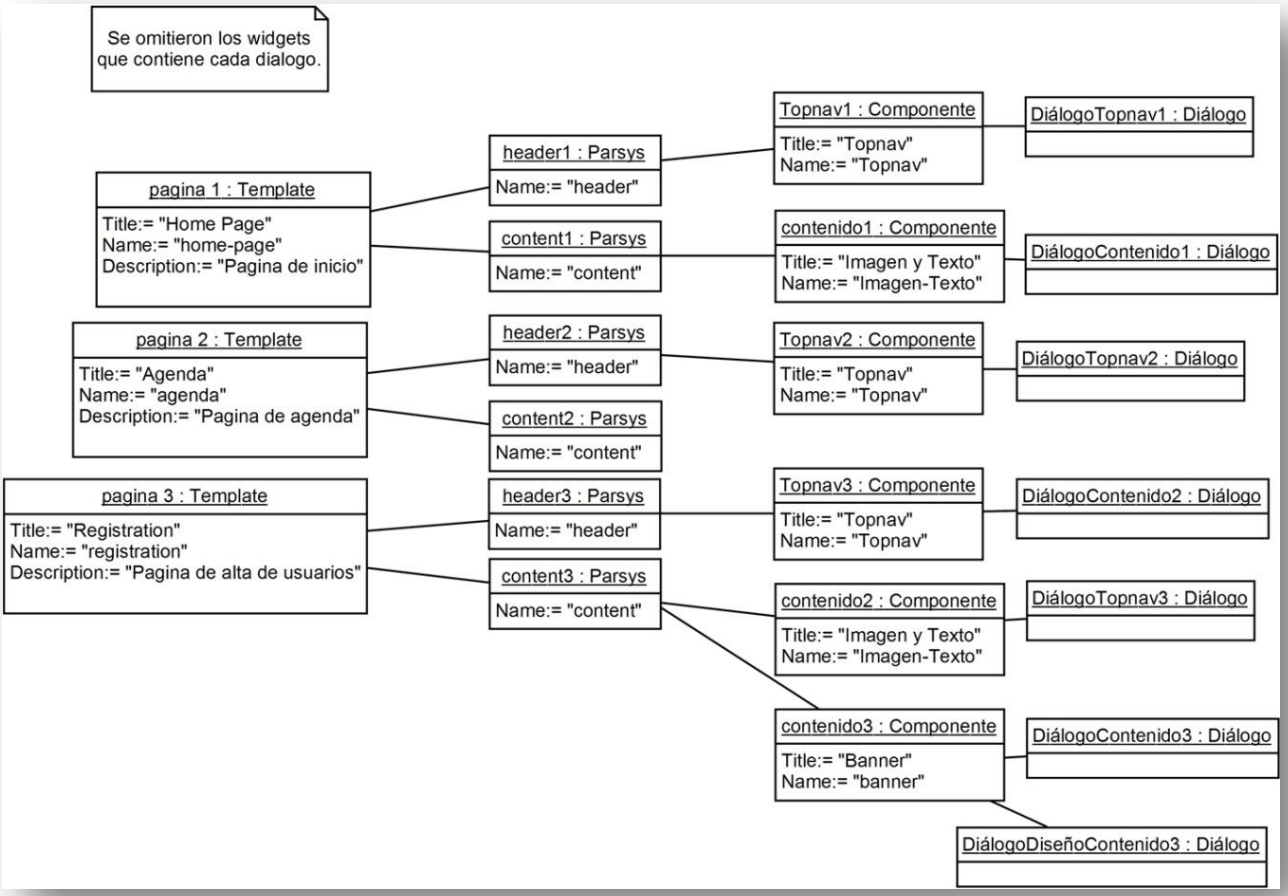


Figura 10 - Diagrama de conceptos de AEM. Fuente: Autoría Propia

4 Ambiente de Desarrollo

Las herramientas que vamos a utilizar para desarrollar el caso de estudio son:

- *IDE* Eclipse versión 4.5.1
- *Apache Maven* versión 3.3.9
- JAVA versión 1.8.0_101
- GIT versión 2.8.3.windows.1
- Brackets versión 1.7

4.1 Eclipse

La decisión de utilizar Eclipse como *ide* surge a partir de la disponibilidad del plug-in *AEM Developer Tool* [7] que facilita el desarrollo de proyectos para la plataforma en AEM. A partir del plug-in antes mencionado podemos crear un arquetipo de proyecto [8] diseñado bajo las recomendaciones de Adobe, con una estructura modula divida en carpetas. Cada carpeta es un proyecto que utiliza el *framework* Maven. Señalaremos los tres proyectos más significativos para nuestro caso de estudio:

- **core**
  El proyecto core contiene todas las clases JAVA utilizadas por el sitio que vamos a desarrollar. Cuando nosotros desplegamos nuestro proyecto a un servidor AEM el JAR generado a partir de dicho proyecto es cargado en el servidor como un *bundle* de OSGi, por eso es posible acceder a las clases definidas dentro de dicho *bundle*.
- **ui.apps**
  El proyecto ui.apps, básicamente es un conjunto de carpetas donde encontramos todos los componentes que vamos a definir en nuestro proyecto, estos componentes son representados por carpetas y dentro de las mismas encontramos los script, archivos HTML y otros diferentes archivos que pertenecen a los nodo correspondientes en AEM. Las propiedades de los nodos son representadas mediante archivos XML que son generados automáticamente por la plataforma. Nosotros vamos a utilizar otro IDE para modificar los archivos definidos en este proyecto en particular.
- **ui.content**
  Similar a la estructura proyecto ui.apps, tenemos el proyecto ui.content el cual guarda el contenido del sitio web correspondiente a nuestro proyecto.

A su vez el arquetipo divide a los componentes en dos grandes grupos:

- **Componentes estructurales**
  Son parte de la estructura del sitio web y comparten la misma configuración para todas sus instancias incluidas en el mismo.
- **Componentes de contenido**
  A diferencia de los componentes estructurales, cada instancia puede contener una configuración y contenido único.

El *framework* Maven no sólo es utilizado para administrar las distintas dependencias, sino que también es el encargado de desplegar los distintos cambios que se realicen localmente a cualquier servidor AEM. Para ello, debe configurarse en el *pom.xml* del proyecto la dirección del servidor AEM al cual queremos enviar nuestros cambios -en nuestro caso será *localhost,* junto a un usuario administrador y su contraseña-.

4.2 Brackets

Se trata de un IDE desarrollado por Adobe para la edición de archivos que atañan al desarrollo *frontend*. Lo particular de este *ide* es que cuenta con un *plug-in* para AEM que permite descargar o desplegar cambios al servidor, nuevamente debemos ingresar la dirección del servidor de AEM, en la cual queremos descargar o desplegar los cambios en los scripts junto con un usuario administrador y su contraseña. Por este motivo vamos a utilizar esta herramienta para el desarrollo de archivos del tipo HTML, scripts o estilos, que se encuentran en el proyecto *ui.apps*.

4.3 Sightly

A partir de la versión 6.0, AEM incorpora un lenguaje de modelo HTML llamado *Sightly* para sustituir los archivos JSP de la solución. El mismo brinda funcionalidades que facilitan la integración de los archivos HTML con AEM. Esto permite, por ejemplo, acceder a propiedades de nodos o del componente al cual pertenece el archivo HTML, instanciar una clase JAVA de

cualquier *bundle* cargado en el servidor a partir de su ruta completa de paquetes, recorrer listas o importar otros archivos HTML.

Dicho lenguaje se utilizará para el desarrollo del caso de estudio, esto es, básicamente un HTML con la extensión de las funcionalidades ya mencionadas.

4.4 CRX Lite

AEM ofrece una interfaz gráfica a través del servidor web para su repositorio CRX. Tal como observamos en la Figura 11, podemos acceder a todo el repositorio del servidor, recorriendo el contenido del mismo. Asimismo, permite crear o eliminar nodos, modificar sus propiedades y editar archivos, entre otras opciones.

Es precisamente desde esta interfaz es que comenzamos el desarrollo de un componente, creando el nodo correspondiente para luego descargarlo en el *IDE bracket* y continuar editando los archivos que definen el renderizado del mismo.

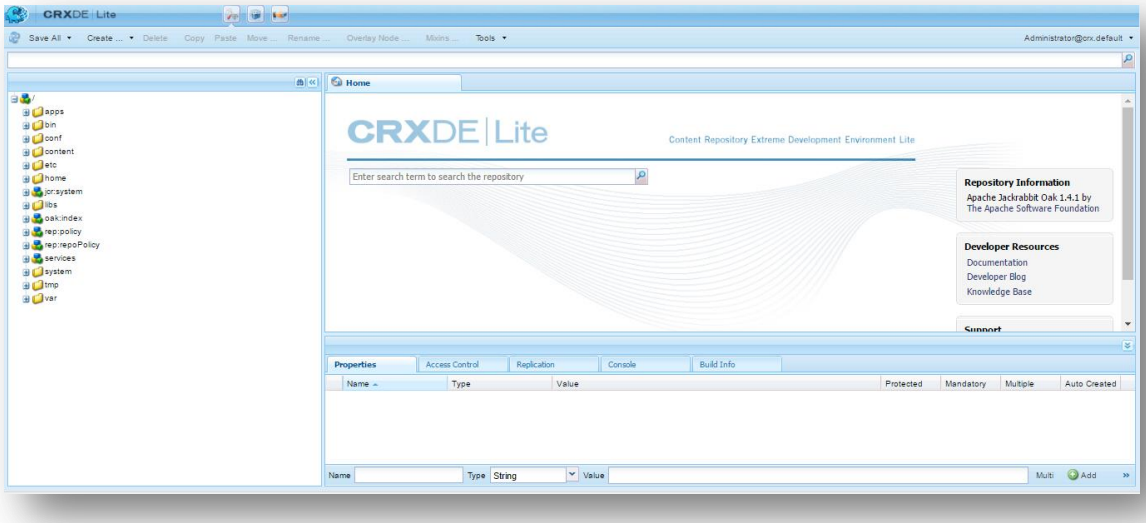


Figura 11 – Captura de pantalla de la interfaz gráfica que ofrece AEM del CRX. Fuente: Autoría Propia

4.5 Git

Con el fin de generar un ambiente colaborativo, utilizaremos *GIT* y el repositorio gratuito *Bitbucket*. El proyecto anteriormente definido resulta completamente funcional, por lo que prescinde de herramientas extra para ser desplegado y compartido por otro desarrollador. Esto potencia la colaboración entre desarrolladores.

4.6 Resumen

Nuestro ambiente de desarrollo se podría dividir en tres partes. Tal como vemos en la **¡Error! o se encuentra el origen de la referencia.**, nuestro proyecto contiene todo el código del sitio web que será editado con dos IDEs, *Brackets* y *Eclipse*. De este modo, utilizaremos Maven para desplegar los cambios que son realizados del código a nuestro servidor AEM (local). A su vez, utilizaremos la interfaz web *CRX Lite* que provee AEM para crear nodos y definir sus

propiedades -estos cambios serán descargados al código del proyecto mediante el *plug-in* que ofrece Adobe en el *IDE Brackets*-. Por último, persistiremos el proyecto en el repositorio de *Bitbucket* utilizado GIT.

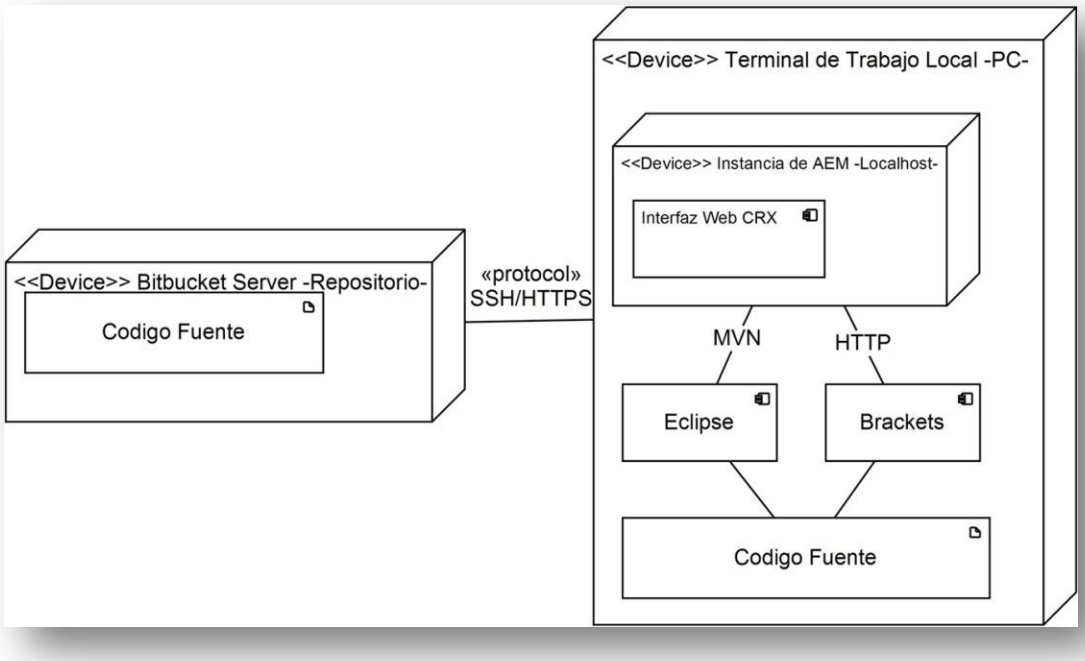


Figura 12 – Modelado de ambiente de desarrollo utilizado. Fuente: Autoría Propia

5 Ambiente de Autor

AEM ofrece un ambiente web de edición para el autor. En el mismo- y con los permisos suficientes- los autores pueden crear páginas, agregarles componentes, editar sus parámetros, editar sus contenidos, publicarlos, entre otras tareas. Tal como ilustra la Figura 13, AEM ofrece una interfaz gráfica para administrar las páginas de los sitios web. Las mismas son ordenadas de forma jerárquica y desde esta vista los autores pueden ver el estado de las páginas: si están publicadas detalla la fecha de publicación (entre otras funciones), y haciendo clic sobre

cualquiera de ellas, el autor puede acceder a la vista de una página web individual.

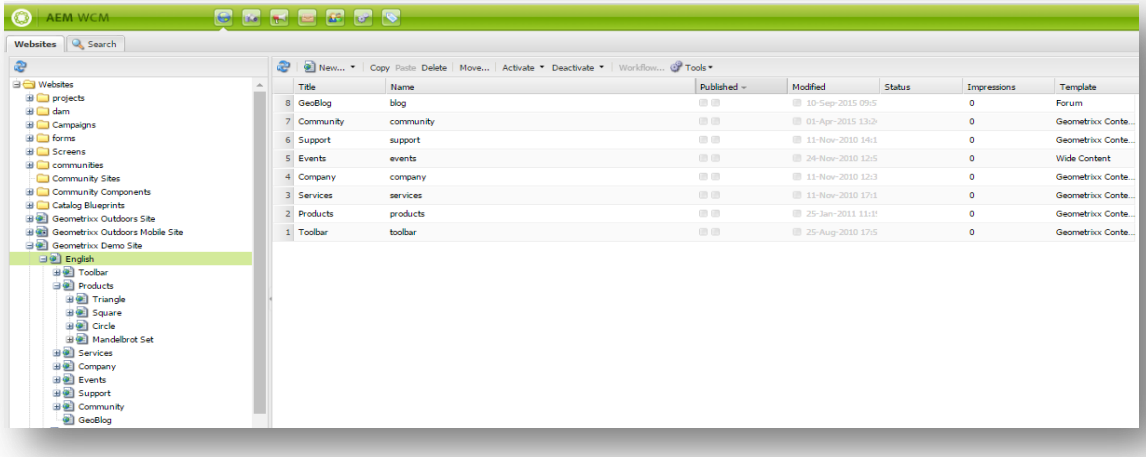


Figura 13 - Captura de pantalla de la interfaz gráfica web que ofrece AEM para administrar las páginas.

Fuente: Autoría Propia

AEM renderiza la página web que seleccionó el autor con su componente de página, en la Figura 14 podemos ver un ejemplo de cómo luce la misma para el autor. El menú que podemos observar a la izquierda de la imagen, llamado *sidekick* muestra los distintos componentes habilitados para usar en la página. El autor puede filtrarlos por nombre o por grupo, o bien, puede simplemente arrastrarlos al *parsys* para agregarlos a la página. La página que observamos en la figura de ejemplo tiene un único *parsys* (aparece dentro de un rectángulo con un texto que dice "*Drag components here").*

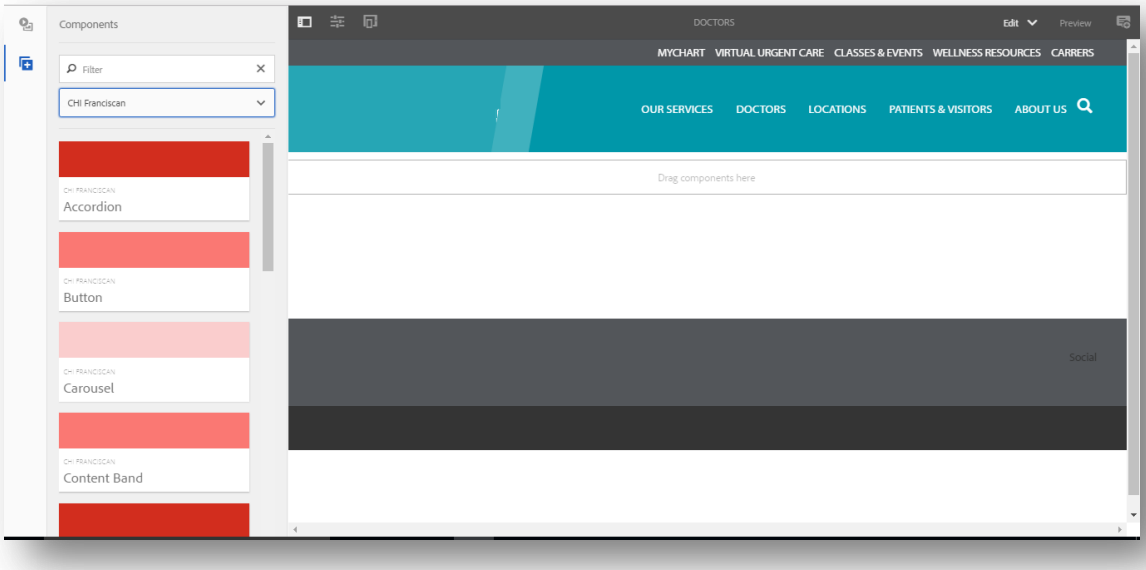


Figura 14 - Captura de pantalla de la interfaz gráfica web que ofrece AEM como ambiente de autores.

Fuente: Autoría Propia

El ambiente cuenta con diferentes modos, de los cuales vamos a nombrar tres que son los más utilizadas por los autores:

- **Modo Edición:**
  El modo de edición es donde el autor puede insertar, eliminar o mover los componentes en los *parsys* que contiene la página. También se le permite acceder a los diálogos de los componentes y editar su contenido.
- **Modo Diseño**
  El modo diseño le permite a los autores –con permisos suficientes.- configurar cuales son los componentes habilitados para cada *parsys*. También permite acceder al diálogo de diseño de los componentes ya agregados a la página. El contenido y los parámetro definidos en dichos diálogos tienen un comportamiento diferente a los definidos en el modo edición (más adelante detallaremos la diferencia).
- **Modo Pre visualización**
  El modo pre visualización muestra a la página renderizada tal como si estuviera publicada y siendo vista por el usuario final.

6 Conceptos para el desarrollo de componentes

6.1 Introducción

Un componente de contenido es una unidad que agrupa *scripts*, archivos, datos y propiedades, los cuales cumplen en conjunto una función específica. En el proyecto nos centraremos en los componentes desarrollados para que sean utilizados por los autores; estos forman parte fundamental de un proyecto WCMS y pueden variar desde simples títulos hasta un navegador complejo con varios niveles de paginación. Aunque AEM ofrece la ventaja de tener un conjunto pre cargado de componentes definidos y listos para su uso, en la mayoría de los casos no se ajustan a los requerimientos del sitio web. De todas formas, resultan útiles para ejemplificar el funcionamiento *per se* de la plataforma o para extender sus funcionalidades.

6.2 Estructura

Existe una configuración básica para que un componente pueda funcionar de forma correcta. Las principales propiedades de un componente son:

- **jcr:primaryType**
  Esta propiedad define el tipo de nodo (que en cuyo caso los componentes deberá ser *cq:Component)*. AEM viene con varios tipos de nodos establecidos, siendo esta propiedad definida por el desarrollador al crear el nodo.

- **jcr:title**
  Establece el título con el cual el autor identificará al componente.

- **componentGroup**
  Define a qué grupo pertenece el componente, básicamente su función es facilitar la tarea del autor, agrupando los componente en diferentes conjuntos según la

conveniencia del desarrollador. También se puede utilizar para restringir el acceso a cierto grupo de componentes -si un componente no tiene esta propiedad asociada, se lo asociada al grupo llamado *undefine*-.

Adicionalmente, el componente debe tener por lo menos dos nodos hijos, uno que defina el diálogo, y un segundo archivo que represente el código que será utilizado para el renderizado (generalmente este archivo es HTML o JSP).

- **cq:dialog**
  Este nodo contiene una jerarquía de nodos hijos que definen el diálogo y los diferentes *wdigets* que lo compone. Tal como muestra la Figura 15, su *jcr:primaryType* es *nt:unstructure.* Es posible acceder al diálogo desde el modo de edición del ambiente de autor.

- **Archivo de renderizado**
  Este archivo debe tener el mismo nombre que el nodo del componente, puede ser un HTML o JSP. Cuando un componente es insertado en una página por el autor, por defecto AEM buscará este archivo para renderizarlo.

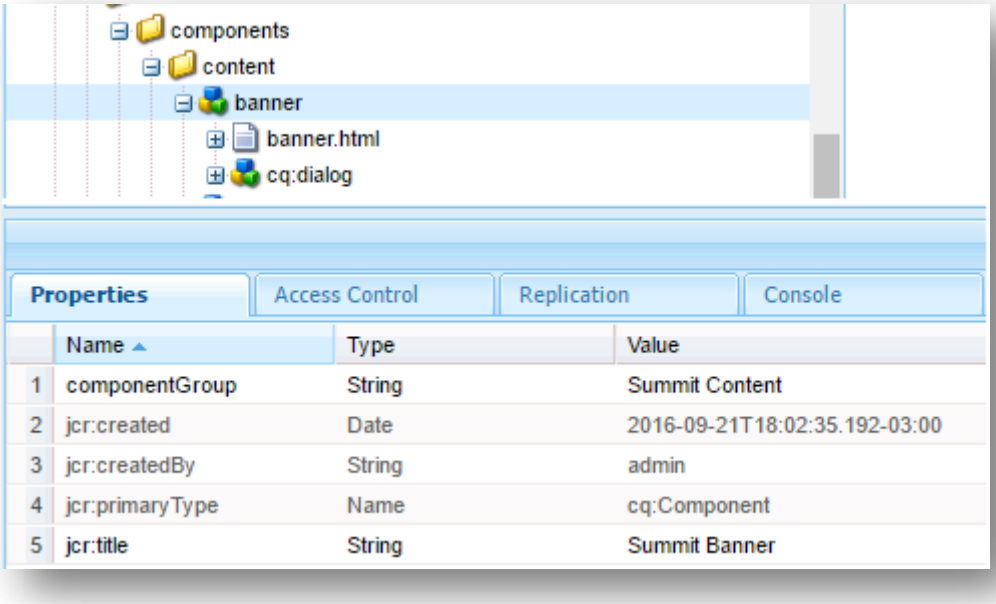


Figura 15 – Captura de pantalla del CRX mostrando las propiedades del componente Banner.

Fuente: Autoría propia.

6.3 Dialogo de diseño

También es posible definirle un diálogo de diseño al componente. La jerarquía y estructura de especificación es idéntica a un *cq:dialog*, la diferencia en el repositorio es el nombre, el cual debe ser *cq:design_dialog.* Dicho diálogo es accedido por el autor en el modo diseño -el contenido definido en este tipo de diálogo es común a todas las instancias del componente, a

diferencia del dialogo en el modo edición, el cual es específico para la instancia del componente en el que se está editando-.

6.4 Administración de archivos de estilos y scripts

AEM ofrece una funcionalidad especial para administrar los archivos de estilos CSS, LESS y JAVA SCRIPT individualmente en cada componente, para que luego puedan ser exportados de forma simple dentro del sitio web. Como vemos en la Figura 16, el componte *read-more,* contiene un nodo llamado *clientlibs* [11] (a partir de ahora librerías) cuyo tipo primario es *cq:ClientLibraryFolder.* Dentro del mismo se pueden diferenciar cuatro archivos; los cuales referente a su configuración son:

- **css.txt**
  En el archivo css.txt se especifica el nombre de los archivos de estilos que deben ser incluidos en el momento que se importe la librería. En la Figura 16, únicamente tiene escrito *read-more.less.*

- **js.txt**
  Análogo al archivo css.txt, en *js.txt* se definen los Java Scripts que deben ser incluidos cuando se importe la librería. En el ejemplo de la Figura 16, únicamente tiene escrito *read-more.js*

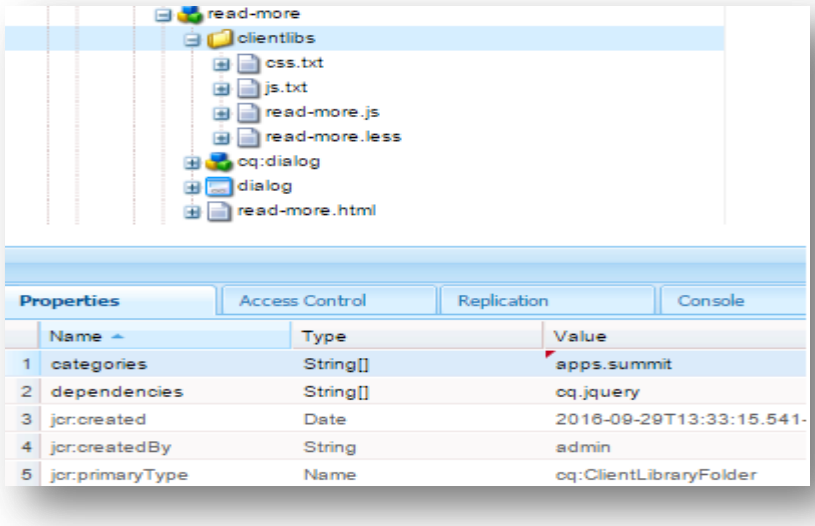


Figura 16 – Captura de pantalla mostrando una librería y sus propiedades en el CRX. Fuente: Autoría propia.

Podemos observar dos propiedades con múltiples valores:

- ***categories***,
  Dicha propiedad define una o más categorías a la cual pertenece particularmente la librería. Estas categorías pueden ser utilizadas luego de forma simple para importar todos los archivos que incluye la librería.

- **dependencies**

En esta propiedad el desarrollador puede definir cero o más librerías como dependencias, que son cargadas antes de ejecutar cualquier estilo o *script* definido en la misma.

Mediante las librerías los desarrolladores pueden administrar, editar y mantener los archivos de estilos que son utilizados por los componentes ordenadamente dentro de los mismos. Una estrategia interesante que nos brindan las librerías es la posibilidad de que todos los componentes de un proyecto compartan una misma categoría. De esta forma, y mediante una sintaxis simple, es posible especificar su importación en el HTML de un componente de página asociado a un *template*, y así todas las páginas creadas a partir del mismo, incluirán los archivos de estilos y *Java Script* de la categoría. Ver Figura 17.

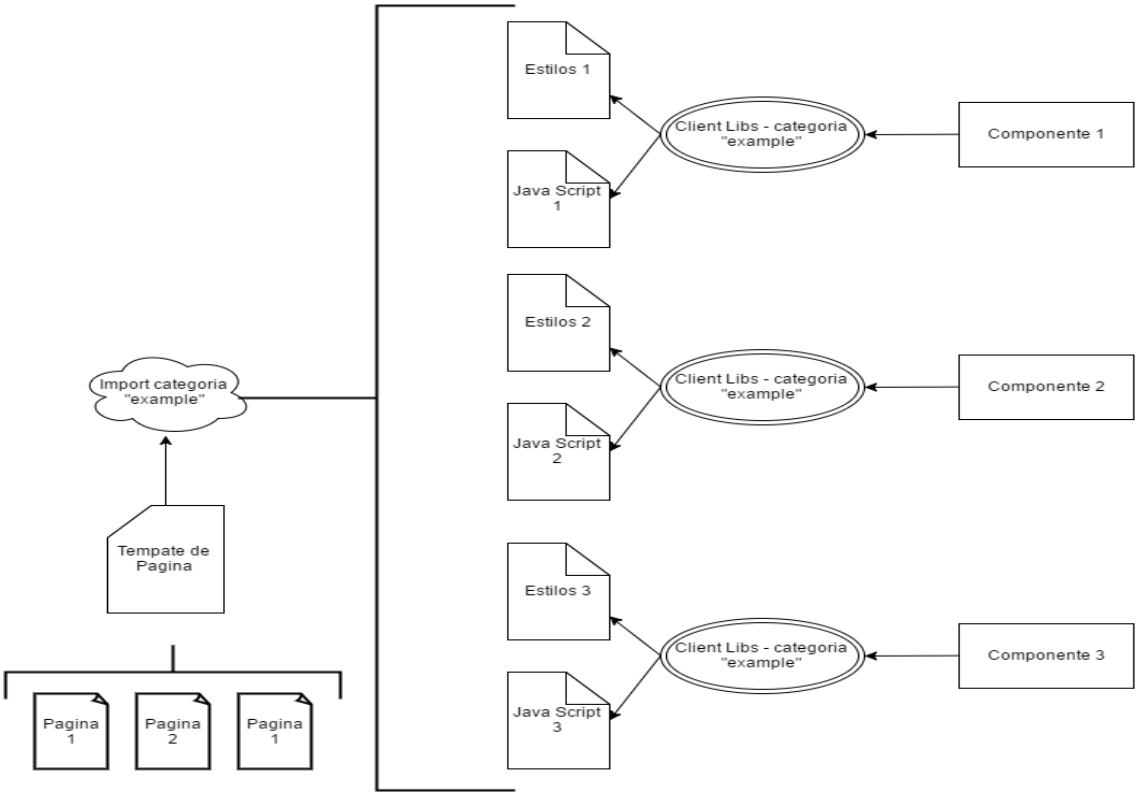


Figura 17 – Modelo conceptual de funcionamiento de las librerías de AEM. Fuente: Autoría propia

6.5 Enfoque de creación de componentes

Existen tres diferentes enfoques a la hora de crear un componente en AEM: construir el componente de cero, por herencia, extendiendo la funcionalidad de un componente ya creado o sobreescribir completamente un componente a través de nuestra implementación.

6.5.1 Creación desde cero

Este enfoque es utilizado cuando no existe ningún componente dentro del servidor AEM que tenga una funcionalidad similar al componente que debemos desarrollar. Básicamente consiste en construirlo desde el inicio al componente, en la carpeta correspondiente a nuestro sitio dentro de *apps*.

6.5.2 Herencia

AEM ofrece el concepto de herencia entre nodos, con especial énfasis en los componentes [9]. Es un concepto similar a la herencia en un paradigma orientado a objetos. Un componente puede basar su comportamiento en otro, e implementar cierta funcionalidad específica diferente, que se considere necesaria. Como ya mencionamos, AEM incluye la implementación de toda su funcionalidad pre definida (incluyendo componente listos para su uso) en la carpeta *libs*, la que se recomienda mantener intacta. Sin embargo, existe la posible de que existan componentes dentro de dicha carpeta que puedan ser útiles pero no se ajusten por completo a nuestras necesidades. Por ejemplo, AEM trae un componente *page* (componente de página) con un conjunto de funcionalidades útiles a la hora de renderizar una página. El desarrollador puede crear un componente *home page* dentro de la carpeta de su sitio y heredar el comportamiento ya definido del componente *page* de AEM, mediante la propiedad *sling:resourceSuperType (*definiendo en el valor de la misma la ruta del componente *page).*

Ilustraremos lo antedicho con un ejemplo: asumamos una jerarquía de nodos como vemos en la Figura 18, donde el componente *home page* tiene como súper tipo el componente *page*, cuando rendericemos una página con el componente *home page,* AEM resolverá todos los recursos faltantes buscándolos en la ruta definida por la propiedad del súper tipo, por defecto dentro de la carpeta *libs*.

De esta forma se podría mantener el mismo diálogo definido en el componente padre, *page,* y a la vez, implementar una versión propia del archivo *page.html* dentro del componente *home page*, con el fin de renderizar la página de forma que se adecúe a nuestras especificaciones.

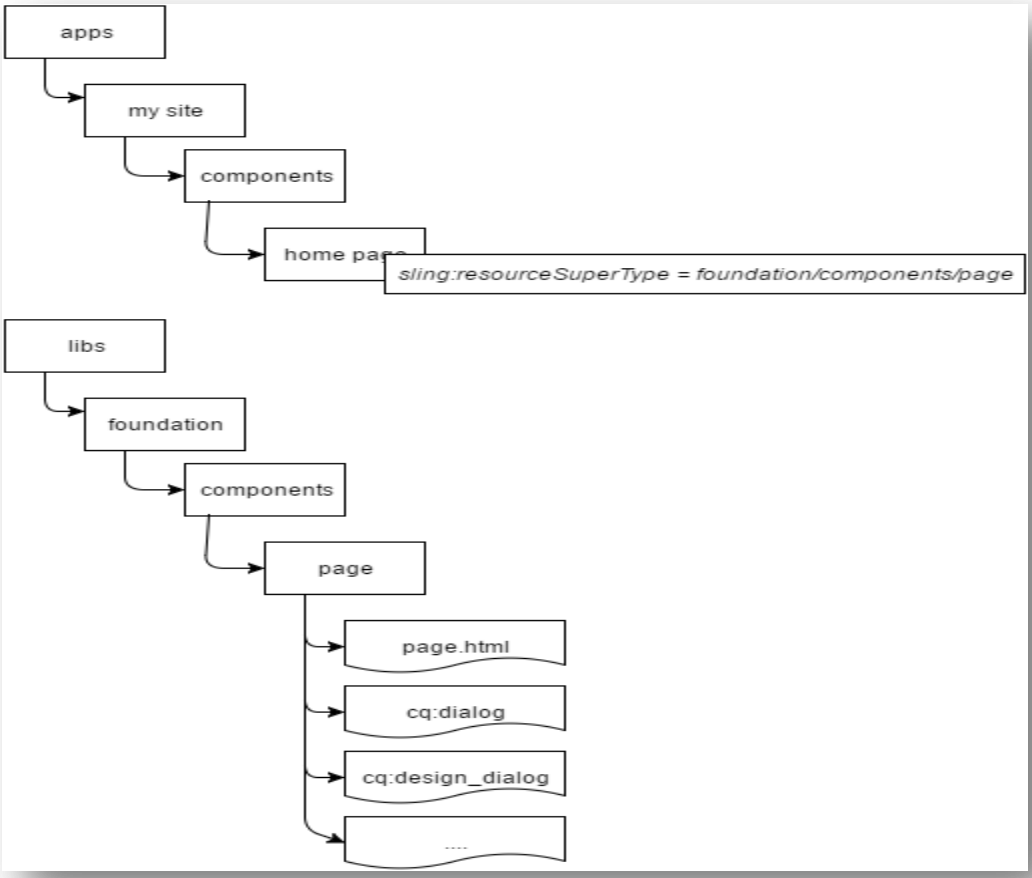


Figura 18 – Esquema de ejemplo de funcionamiento sling:resourceSuperType. Fuente: Autoría Propia

6.5.3 Sobre-escritura

El último enfoque consiste en sobreescribir completamente un componente ya existente. Esto es posible asignándole al nodo del componente definido en la carpeta *apps* el mismo nombre del cual queremos sobreescribir. De esta forma, dentro de nuestro sitio web todas las referencias hechas a dicho componente serán redireccionadas al nodo antes mencionado.

6.6 Persistencia

La mayor parte del contenido definido es guardado en la carpeta *content* de AEM con una jerarquía de nodos. En esta carpeta, cada página creada por el autor es persistida en forma de nodo -con nodos hijos que representan a los *parsys* que contiene la misma-. A su vez, dentro de dichos nodos que hacen referencia a los *parsys*, se encuentran los nodos hijos que representan al componente que definió el autor dentro de los *parsys* antes mencionados. Veamos el ejemplo de la Figura 19, donde se detalla una página con dos componentes de texto, definidos en un mismo *parsys* identificado como *content.* Cada componente de texto posee un valor diferente.

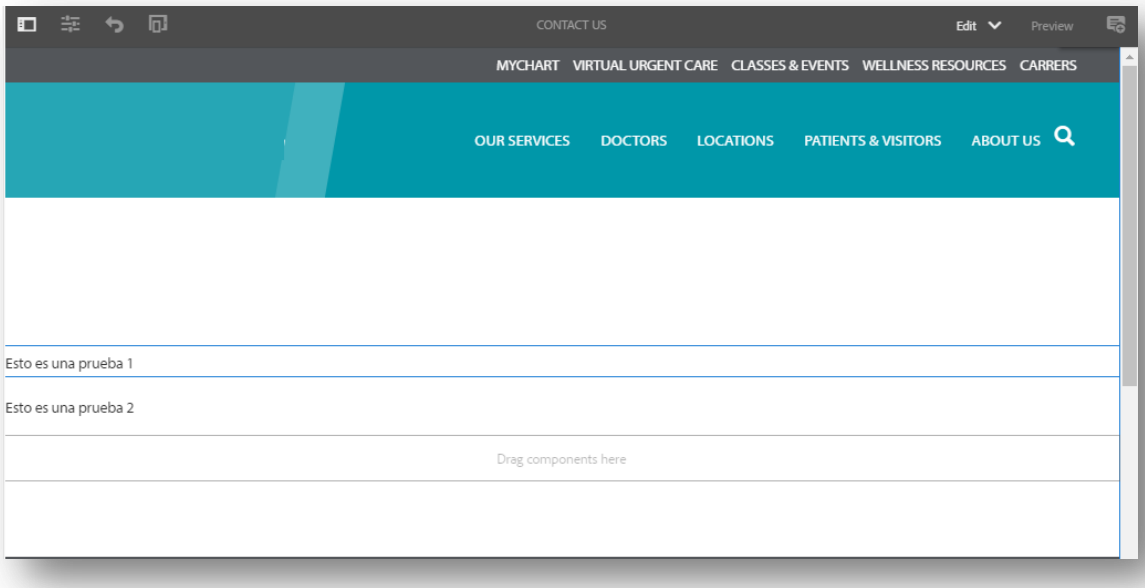


Figura 19 – Captura de pantalla del ambiente de autor de AEM donde hay incluidos dos componentes de texto. Fuente: Autoría Propia.

Si ingresando en el navegador a *CRX Lite*, podemos observar en la Figura 20 como es representada la página antes mencionada. El nodo *contact-us* representa a la página, como nodo hijo observamos un nodo especial llamado *jcr:content,* el cual es utilizado por AEM para agrupar el contenido, dentro del mismo podemos encontrar el parsys *content* el cual contiene a los nodos hijos que representan a las dos diferentes instancias de componente *text* que están incluidas en el mismo. Como podemos ver estos nodos tienen diferente nombre y AEM es el responsable de generar un nombre único local.

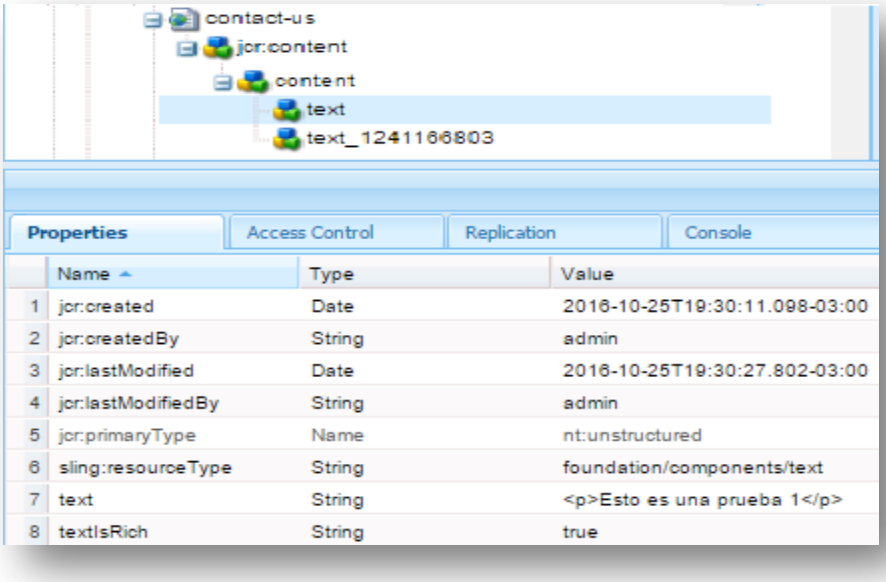


Figura 20 – Captura de pantalla con los datos persistidos correspondientes a los dos componentes de texto antes descriptos. Fuente: Autoría Propia.

Observando las propiedades, se evidencia que el súper tipo de los nodos que representa al componente *text,* es el componente propiamente dicho. También se puede observar que en estos nodos se persiste la propiedad definida mediante el diálogo por el autor que especifica el valor del texto a mostrar.

Las propiedades definidas mediante un diálogo de diseño, son persistidas en una estructura análoga pero en la carpeta *etc.*

7 AEM JAVA USE-API

La API que provee AEM permite una comunicación simple desde Sightly a una clase JAVA, de esta forma la implementación de lógica compleja de negocio queda encapsulada en código JAVA, mientras que Sightly solo se encarga de mostrar los diferentes datos.

Una de las características más utilizadas en la clase abstracta *WCMUsePojo*, la cual brinda diferentes funcionalidades a las que puede acceder el desarrollador. Entre ellas se encuentra la posibilitada de acceder al nodo en el cual se encuentra el componente o a la página, junto a las propiedades de ambos. Como vemos en la Figura 21, para poder acceder desde sighlty a la clase JAVA debe pertenecer a un *bundle* de *OSGi* y a su vez extender de la clase WCMUsePojo. Cumpliendo ambas especificación es posible acceder a otras diferentes funcionalidades de la API como el *PageManager* que permite buscar y obtener paginas dada una ruta especifica o el QueryCreator el que permite crear *querys* para realizar buscqueda de nodos o contenido dentro del repositorio JCR, todo esto sucede del lado del servidor, el resultado final de la interacción entre sightly y el bundle es el HTML, estilos y scripts que ve el cliente en su navegador.

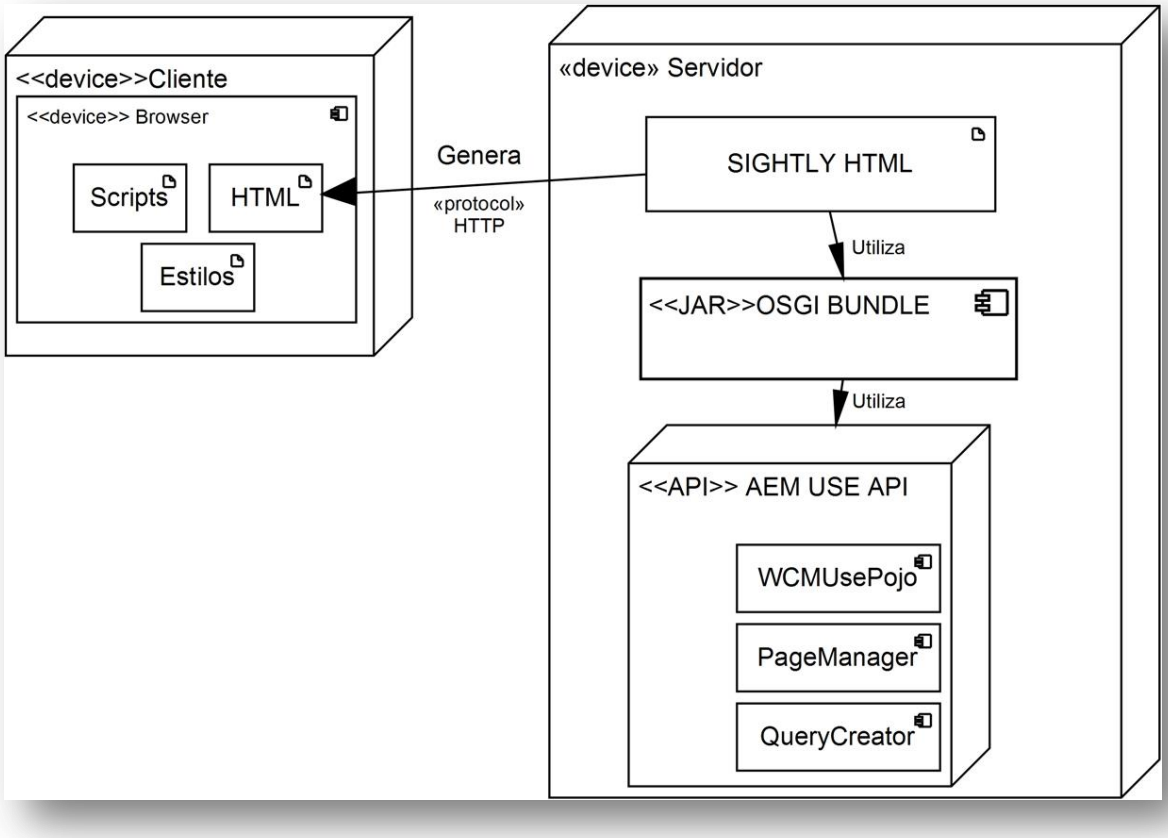


Figura 21 – Modelado de funcionamiento de AEM USE-API. Fuente: Autoría Propia

8 Referencias

[1]https://docs.adobe.com/docs/en/htl/docs/use-api/java.html
[2][https://docs.adobe.com/docs/en/aem/6-2/develop/the-basics.html]
[3][https://docs.adobe.com/docs/en/aem/6-2/deploy.html#What%20is%20AEM?]
[4] http://jackrabbit.apache.org/jcr/node-types.html Ultimos Acceso 21/10/2016
[5] https://docs.adobe.com/docs/ko/aem/6-1/develop/components.html#Paragraph System. Último Acceso 21/10/2016
[6] https://docs.adobe.com/docs/en/aem/6-2/develop/dev-tools/howto-projects-eclipse.html
[7] https://docs.adobe.com/docs/en/dev-tools/aem-eclipse.html
[8] https://github.com/Adobe-Marketing-Cloud/aem-project-archetype
[9] https://docs.adobe.com/docs/en/aem/6-2/develop/platform/sling-resource-merger.html
[10] https://docs.adobe.com/docs/en/aem/6-2/develop/platform/overlays.html
[11] https://docs.adobe.com/docs/en/aem/6-1/develop/the-basics/clientlibs.html
[12]https://docs.adobe.com/content/docs/en/spec/jcr/2.0/3_Repository_Model.html 21/10/2016
[13] https://docs.adobe.com/docs/en/cq/5-6-1/exploring/architecture-overview.html. Último acceso 21/10/2016.
[14] https://docs.adobe.com/content/docs/en/cq/5-6-1/developing/the_basics. Último acceso 21/10/2016.

Anexo 3 Solución completa del caso de estudio en AEM

Índice

1 Estructura

1.1 Estructura de proyecto

La estructura general del proyecto se basa en el arquetipo descripto en el capítulo anterior. Tal como nos señala la Figura 1, el nombre del proyecto es *Summit* -el cual viene ya inicializado con dos carpetas de componentes diferenciando los que son de estructura y de contenido-. El arquetipo crea dos *templates* pre-definidos, denominados *page-content* y *page-home*. Ambos poseen como componente de página al componente *page* que se encuentra en el proyecto bajo la ruta */apps/summit/components/structure/page*, el mismo tiene como súper tipo al componente *page* definido por AEM en la carpeta *libs*.

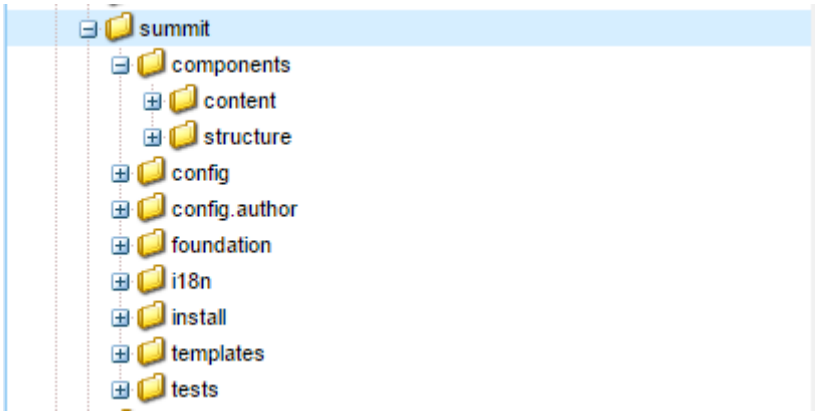


Figura 1 - – Captura de pantalla de CRX mostrando estructura del proyecto. Fuente: Autoría Propia

El arquetipo define también una librería global cuya categoría es *summit.all* y cuyo nodo se encuentra en la carpeta asociada al proyecto bajo *etc,* tal como muestra la Figura 2. También podemos observar que en dicha carpeta creamos los archivos globales de estilo y *scripts* del sitio web.

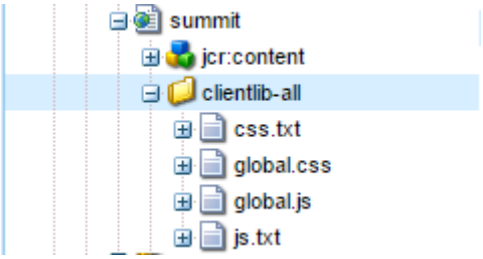


Figura 2 – Captura de pantalla mostrando la librería global del proyecto. Fuente: Autoría Propia

1.2 Estructura de Sitio

1.3 Componentes

1.3.1.1 Template de página

Si bien no es necesario modificar los nodos templates definidos por defecto en el proyecto *Summit,* sí lo será a la hora de modificar el archivo HTML del componente de página asociado a los mismos. La propuesta consiste en sobrescribir el archivo encargado del renderizado de la página - *page.html*-, ajustándolo a nuestras especificaciones pero manteniendo las funcionalidades definidas por el componente padre de AEM.

Tal como nos muestra la Figura 3 al mismo le agregaremos tres *parsys*: uno llamado *content* -el cual será utilizado por los autores para incluir componentes de contenido a la página-, otro llamado *header* -donde se incluirá estáticamente al componente *topnav -,* y por último, haremos lo análogo para el *footer*.

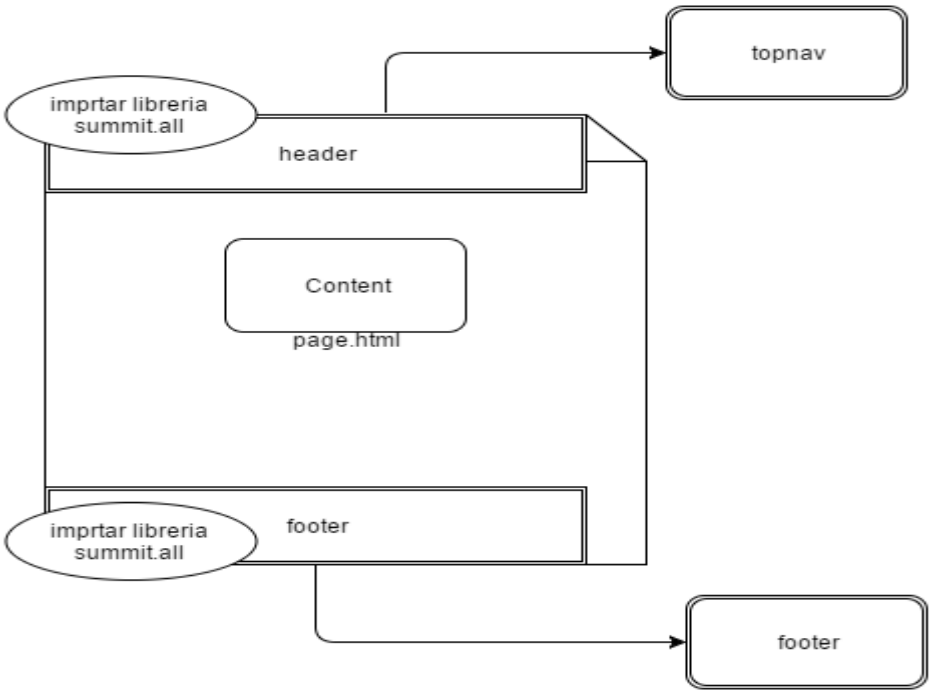


Figura 3 – Esquema de *template* de página desarrollado para la solución de caso de estudio. Fuente: Autoría Propia

Existen dos importaciones a la librería *Summit.all*: en el *header* del documento podremos importar los estilos mientras que en el *body* importaremos los *scripts*. La sintaxis de *sighlty* permite diferenciar si las importaciones son de archivos de estilo o archivos de *scripts*. Con la implementación descripta nos aseguramos que todas las páginas creadas a partir de los *templates* de nuestro proyecto tengan incorporado al componente *topnav* y *footer*, así como las importaciones de estilos y scripts necesarios.

1.3.2 Componentes Estructurales

1.3.2.1 Topnav

El componente *topnav* es el encargado de manejar la navegación del sitio web. Para facilitar la utilización del componente por parte de los autores -ilustrado en la Figura 4- se decidió otorgarle la capacidad de cargar de forma automática las páginas de primer nivel creadas por los autores bajo la página raíz del sitio web. Dos propiedades incidirán en el comportamiento del componente: el título asociado a cada página (utilizado para renderizar el texto del link en la navegación del componente) y la propiedad *hide in navigation* -la cual por defecto es falsa, pero en caso de ser verdadera la página no será mostrada en el navegador-.

El *topnav* también debe brindar la funcionalidad de *log-in* y *log-out* de los usuarios. El componente será incluido en la categoría *hidden* con el fin de mantenerlo oculto, evitando que sea utilizado por los autores, ya que el mismo es incluido de forma estática en todas las páginas del sitio web.

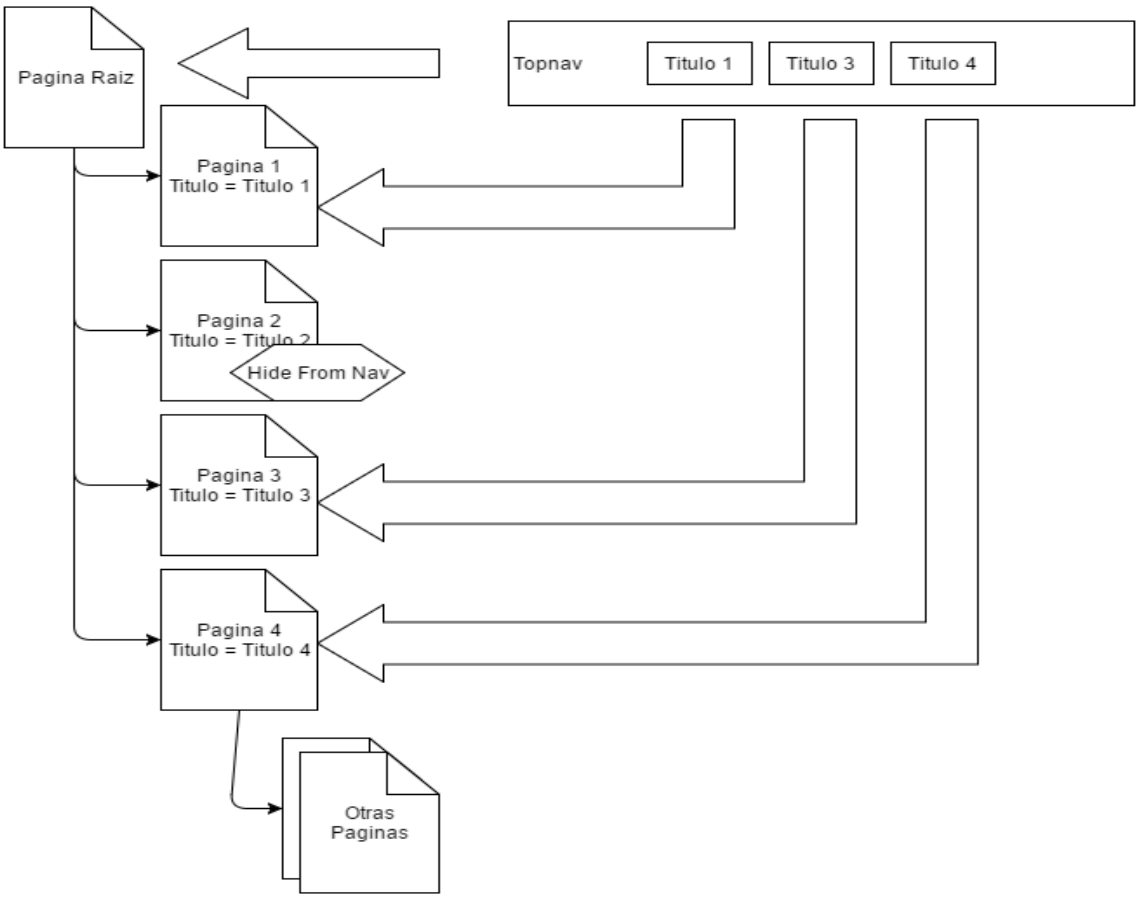


Figura 4 – Esquema del funcionamiento de los link del Topnav. Fuente: Autoría Propia

1.3.2.1.1 Diálogo del Topnav

El componente incluirá un diálogo de diseño, esto permite que la configuración definida por los autores sea compartida por todas las instancias del mismo, evitando así que se deba configurar individualmente.

El diálogo está compuesto, a su vez, por ocho *widgets* que definen las siguientes propiedades:

- ***Root Page***
  Definida como un *widget* tipo *path.* La propiedad define la ruta donde se encuentra la raíz de las páginas del sitio web y a partir de ella el componente será capaz de cargar automáticamente las páginas hijas de primer nivel.
- ***Registration Path***
  Definida como un *widget* tipo *path*. La propiedad define la ruta de la página a la cual se debe redireccionar a un usuario del sitio web que busca darse de alta.
- ***Logo Image***
  Definida como un *widget* tipo *image* La propiedad define la imagen del logo del *topnav.*
- **Textos asociados a la parte visual**
  Contiene cinco *widgets* tipo texto, los cuales son utilizados para la configuración de los diferentes textos que son visibles dentro del *log-in box.*

Para facilitar la utilización del diálogo por el autor se divide a los *widgets* en dos conjuntos diferentes. Tal como se señala en la Figura 5 cada conjunto es agrupado en un *tab* conteniendo la definición de las rutas, del logo y de los textos respectivamente.

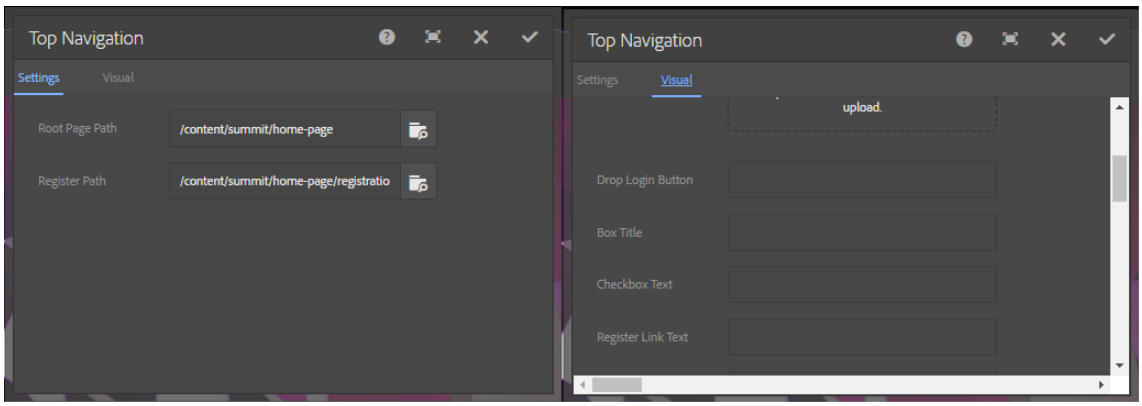


Figura 5 – Captura de pantalla con el dialogo desarrollado para el componente Topnav. Fuente: Autoría Propia

1.3.2.1.2 Backend del Topnav

El componente utilizará una clase de JAVA -creada por nosotros- llamada *MegaMenu*, la cual extiende de la clase *WCMUsePojo*. Cuando el componente es renderizado la vista *topnav.html* se comunica con la clase *MegaMenu* utilizando *sightly.* Mediante esta comunicación *MegaMenu* devuelve a *topnav.html* los datos que deben ser mostrados en el menú.

MegaMenu utiliza la propiedad *Root Page* definida por el autor y la clase *PageManager* que brinda la API de AEM para obtener todas las páginas hijas de primer nivel de la página raíz. A su vez, las páginas son representadas por la clase *Page* definida por la API de AEM y el conjunto obtenido de las mismas es filtrado por la propiedad *hide in navigation.* Dado que la clase *Page* contiene gran volumen de información que resulta irrelevante para generar el navegador, se mapean los atributos nombre y ruta de las páginas en una clase llamada

MenuItem -la cual creamos y utilizamos para enviar una lista de la misma con los datos obtenidos al archivo *topnav.html*-.

Las decisiones tomadas durante la implementación del componente *Topnav* se enmarcaron dentro del objetivo de maximizar la reutilización de componentes y propiedades ya brindadas por AEM (ver Figura 6).

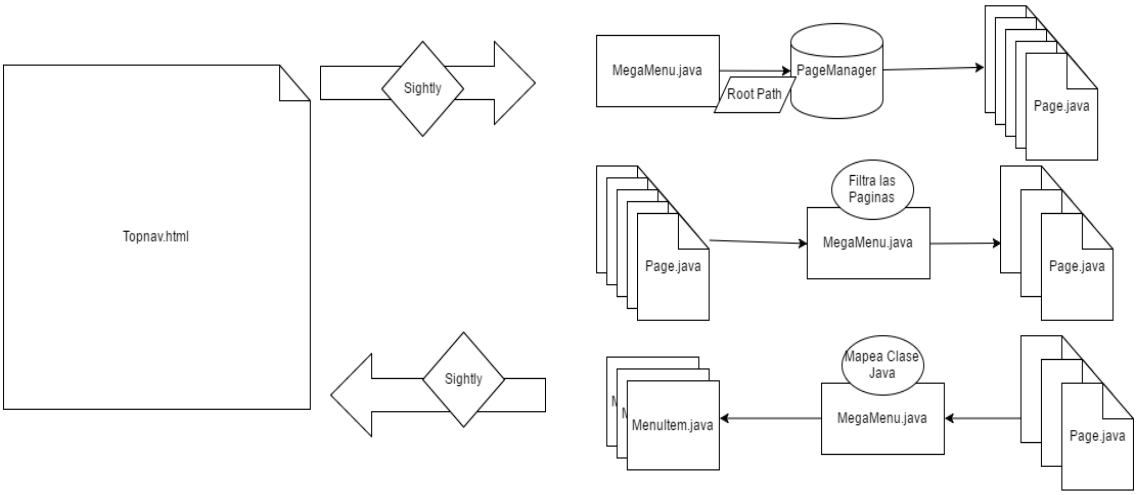


Figura 6 – Esquema que representa el funcionamiento del *Topnav* en lo que respecta a los links de navegación. Fuente: Autoría Propia

1.3.2.1.3 Log in frl Topnav

El componente brinda la posibilidad a los usuarios del sitio web de poder identificarse. Como vemos en la Figura 7 el *log-in box* permite al usuario ingresar su *username* y contraseña. Al hacer click en el botón de *Log In*, el componente realiza un pedido a través de AJAX utilizando *sightly* a la clase *UserUtils,* que creamos específicamente para el manejo de los usuarios del sitio web. Dicha clase utiliza las funcionalidades que ofrece la API de AEM a través del *UserManager* para en este caso, chequear que el usuario y la contraseña recibida coincidan. Como mencionamos anteriormente, todos los textos del *log-in box* son configurables por los autores.

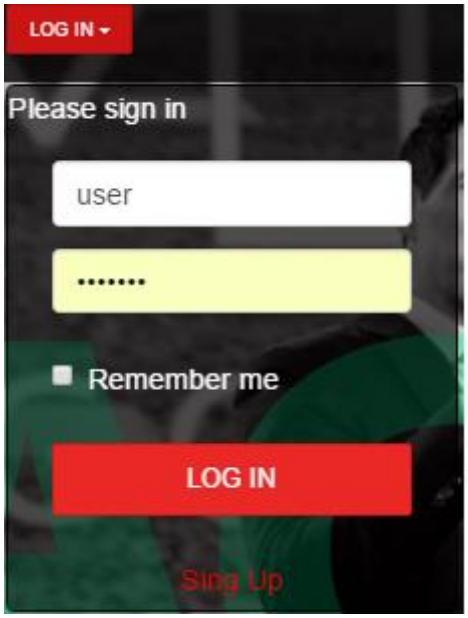


Figura 7 – Captura de pantalla que muestra el login box del Topnav. Fuente: Autoría Propia

1.3.2.2 Footer

El componente *Footer* resulta similar al *Topnav* respecto a que todas las instancias del mismo deben compartir la configuración a través del sitio web, por ello, está compuesto por un dialogo de diseño.

Con el objetivo de restringir el uso del componente, el mismo será incluido en la categoría *hidden,* de esta forma se mantiene oculto para los autores y el mismo es agregado de forma estática para cada página creada en el sitio web.

El diálogo está compuesto por tres *widgets*:

- ***Logo***
  Se trata de un *widget* de tipo imagen que permite al usuario definir el logo del footer.
- ***Text Logo***
  Es un *widget* de tipo texto que permite definir el texto junto al logo
- ***StaticLinks***
  Se define como un *widget* tipo lista que permite crear varios nodos, a los cuales llamaremos s*taticLink* y que contendrán dos propiedades: una URL y un texto. De esta forma, el autor podrá definir desde el diálogo los *links* que aparecen en el *footer* junto con un texto para mostrar.

Como podemos observar en la Figura 8, el diálogo consta de dos *tabs* que dividen en dos conjuntos a los *widgets* antes definidos. Para facilitarle la tarea al autor, una contiene la edición visual del logo y su texto, mientras la siguiente *tab* contiene la lista de *links* que el autor puede definir de forma dinámica.

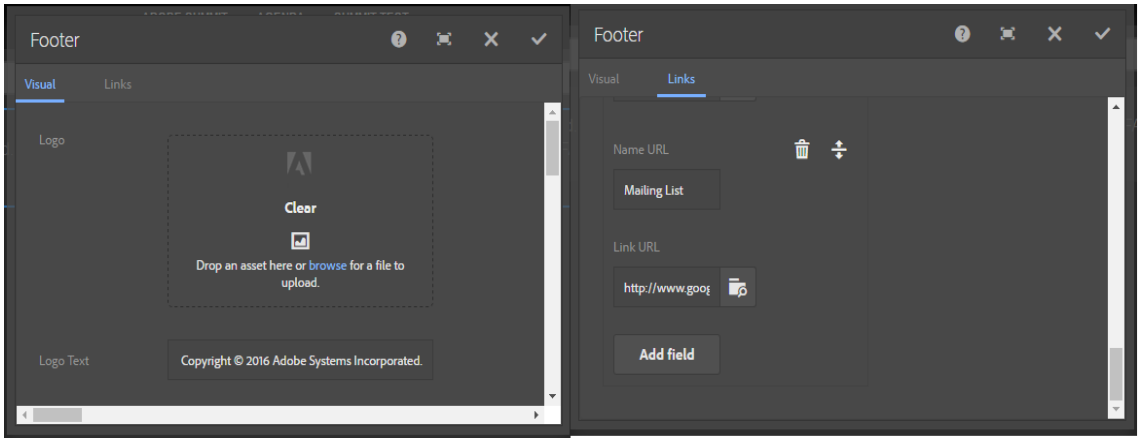


Figura 8 – Captura de pantalla del dialogo creado para el componente Footer. Fuente: Autoría Propia

1.3.3 Componente de contenido

La categoría de todos los componentes definidos a continuación es *Summit Content,* a menos que se especifique lo contrario.

1.3.3.1 Banner

El componente *banner* debe ofrecer la posibilidad de mostrar una imagen configurable por los autores. Como existen varias instancias en el sitio web de dicho componente y cada una de ellas cuenta con una imagen diferente, contará con un diálogo simple y un *widget* de tipo imagen.

1.3.3.2 Jumbo

El componente *Jumbo* es uno de los más versátiles del caso de estudio ya que su presentación es totalmente configurable. El mismo está compuesto por un diálogo simple para que cada instancia pueda ser configurada de forma independiente. Además, contiene ocho *widgets* donde cada uno modifica una propiedad específica del componente. Observando la Figura 9:

- ***Title***
  Mediante un *widget* de texto el autor define el título del componente, como se puede ver en el recuadro rojo.

- ***Sub Title***
  Mediante un *widget* de texto el autor define el sub título del componente, como se puede apreciar en el recuadro verde.

- ***Text***
  El usuario puede editar el texto dentro del recuadro violeta, mediante un *widget* de *Rich Text Editor*, el cual permite enriquecer el texto con herramientas de edición.

- ***ButtonURL y ButtonText***
  Ambos *widgets* se encargan de definir el link recuadrado en naranja, son del tipo *path* y texto respectivamente.

- ***Background Image***
  Se trata de un *widget* tipo imagen que define la imagen de fondo del componente. La misma tiene prioridad sobre el color de fondo, ya que si está definida escribirá sobre el mismo. El autor puede optar por dejar este campo vacío.

- ***Background Color***
  Es un *widget* de tipo *color picker:* permite al autor definir el color de fondo del componente.

- ***Font Color***
  También se trata de un *widget* del tipo *color picker* que permite definir al autor el color del texto mostrado en el componente.

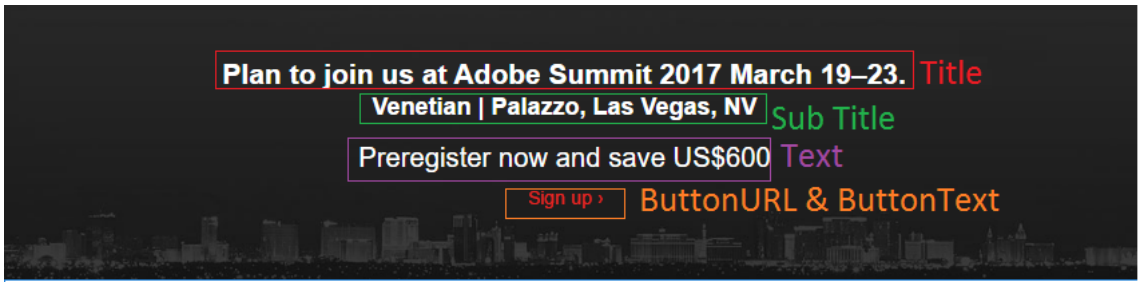


Figura 9 –Captura de pantalla indicando a que campos corresponden los textos del componente.

Fuente: Autoría Propia

Este componente utiliza estilo propio, por lo cual define una librería de categoría *summit.jumbo*- la cual nos limitaremos a nombrar por el momento ya que la misma se definirá como dependencia de la librería *summit.all,* y será ilustrada más adelante en el texto-.

1.3.3.3 Registration

El componente *Registration* es el encargado de dar de alta al usuario en el sitio web. Todos los textos del mismo deben ser editables por el autor. El diálogo será simple, pudiendo configurar de forma diferente las instancias del mismo. Básicamente todos los *widgets* del diálogo son de tipo texto y cada uno especifica una etiqueta de los campos de entrada del formulario de registro.

El componente se comunica con una clase JAVA llamada *UserUtils*, nombrada anteriormente en el componente *Topnav*. La misma utiliza la funcionalidad de creación de usuarios de la API de AEM brindada por la clase *UserManager*. La comunicación será través de AJAX*,* en el caso que se cree el usuario de forma satisfactoria se envía un mensaje de existo o de fracaso si sucede de forma contraria.

Este componente utiliza una librera de estilos y *scripts* propios cuya categoría es *summit.userRegister.*

1.3.3.4 Speaker

La creación del componente *Speaker* surge a partir de una decisión de diseño, que fue seleccionada entre dos opciones. Los speakers definidos en el caso de estudio son utilizados por dos componentes, *Speaker Box* y *Calendar*, ahora bien, surge la disyuntiva respecto a la forma en la cual los autores crearan a los *speakers*. En una primeria instancia, se podría pensar que los mismos deben ser creados utilizando los diálogos propios para cada componente, mediante una lista donde cada elemento contenga una imagen y dos widget de texto que definen su foto, nombre y profesión respectivamente–como vemos en la Figura 10-. Esta implementación es similar a la creación de los *links* en el componente *Footer*, sin embargo en este caso, se obtendría como resultado datos duplicados en los casos que ambos componentes - *Speaker Box* y *Calendar* - requieran la utilización de un mismo *speaker*. A su vez, introduce el riesgo de generar incoherencia entre los datos, en caso de que por error un mismo *speaker* sea definido de forma diferente entre los componentes.

Figura 10 – Captura de pantalla mostrando el dialog del componente Speaker. Fuente: Autoría Propia

El componente *speaker* surge como una alternativa para esta solución y con el objetivo de que sea utilizado por los componentes *Speaker Box* y *Calendar.* Mediante esta decisión de diseño nos aseguramos que los autores creen al *speaker* -representado por un componente- una única vez y de forma centralizada. Para ello, los autores deberán crear una página con la propiedad *hide in navigation* en verdadero. Dicha página va a contener las instancias del componente *Speaker* y permanecerá oculta para los usuarios del sitio web.

El componente está compuesto por un diálogo simple que incluye tres *widgets*, uno de tipo imagen para la foto del speaker, y los siguientes dos de tipo texto para su nombre y profesión respectivamente.

El componente utiliza una librería de estilos y *scripts* propios cuya categoría es *summit.speaker.*

1.3.3.5 *Speaker Box*

Speaker Box está compuesto por cuatro instancias del componente definido por AEM llamado *Paragraph Reference*, quien permite referenciar a una instancia de otro componente. Mediante los mismos, los autores podrán referenciar las instancias del componente *Speaker* antes mencionado a través de su ruta específica.

Los cuatro componentes *Paragraph Reference* incluidos en *Speaker Box* tienen un diálogo simple que contiene un widget de tipo *path.* Éste permite definir la ruta de la instancia de un componente al cual se precise referenciar. A su vez, el propio componente *Speaker Box* provee de un diálogo permitiendo definir el color de fondo con un *widget* de tipo *color picker*.

Por último y para facilitar la edición del componente *Speaker Box*, a la hora de agregar o quitar referencias de los diferentes *Pragraph Reference,* el renderizado se verá de forma diferente dependiendo si se está en modo edición (en un ambiente de autor) o está siendo visto por un usuario del sitio web. Esto es posible gracias a las funcionalidades que brinda *sightly*.

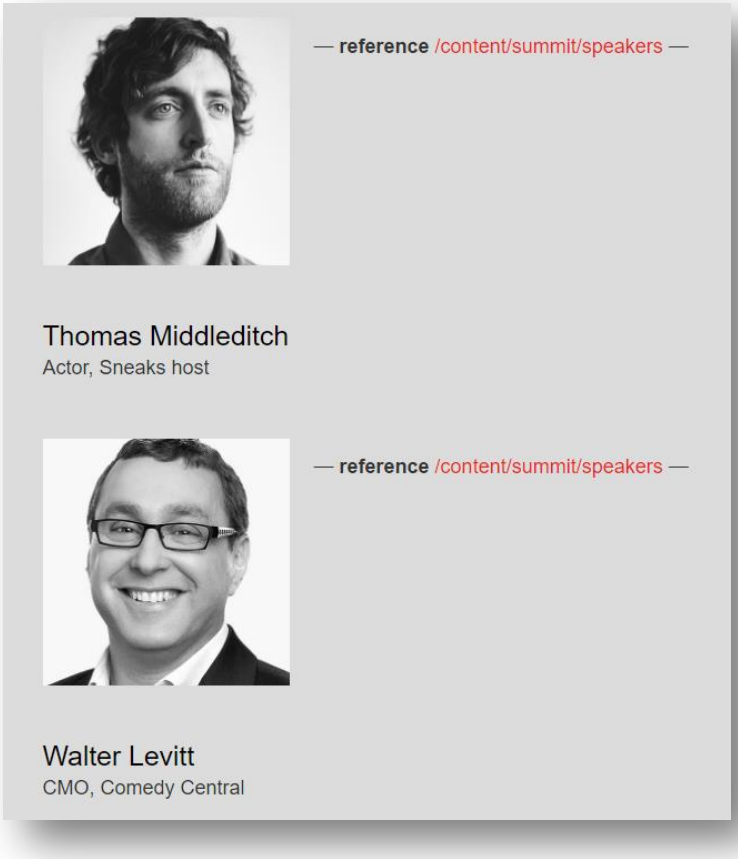


Figura 11 – Captura de pantalla que muestra como son desplegados los speakers en el ambiente de autor. Fuente: Autoría propia

1.3.3.6 *Calendar*

El componente *Calendar* es uno de los más complejos de la solución del caso de estudio, debido a que maneja varios renderizados distintos y una lógica de negocio avanzada.

1.3.3.6.1 Diálogo

El diálogo permite al autor definir la ruta de la página donde se encuentran las instancias del componente *Speaker* que serán utilizadas para la creación de eventos. Este campo es un *widget* de tipo *path.*

1.3.3.6.2 Renderizado

Con el objetivo de dividir las diferentes responsabilidades el componente consta de tres archivos *HTML* para renderizar las distintas vistas utilizadas.

El archivo *calendar.html* es el encargado del renderizado general del componente, desplegando una vista genérica: en caso de estar en un ambiente de autor se mostrará un botón para la creación de un evento, el cual se mantendrá oculto para los usuarios del sitio web. Ver Figura 12.

El objetivo de este archivo es mostrar el calendario y los diferentes eventos creados en él. Cabe destacar que utilizamos el *framework fullcalendar* [1] que utiliza *Angular* para la creación y configuración de su parte visual. La implementación *frontend* fue brindada por nuestro cliente Conexio, y genera la visualización de los eventos asociados a los días de una semana específica, donde los usuarios pueden fácilmente recorrer los distintos meses o años.

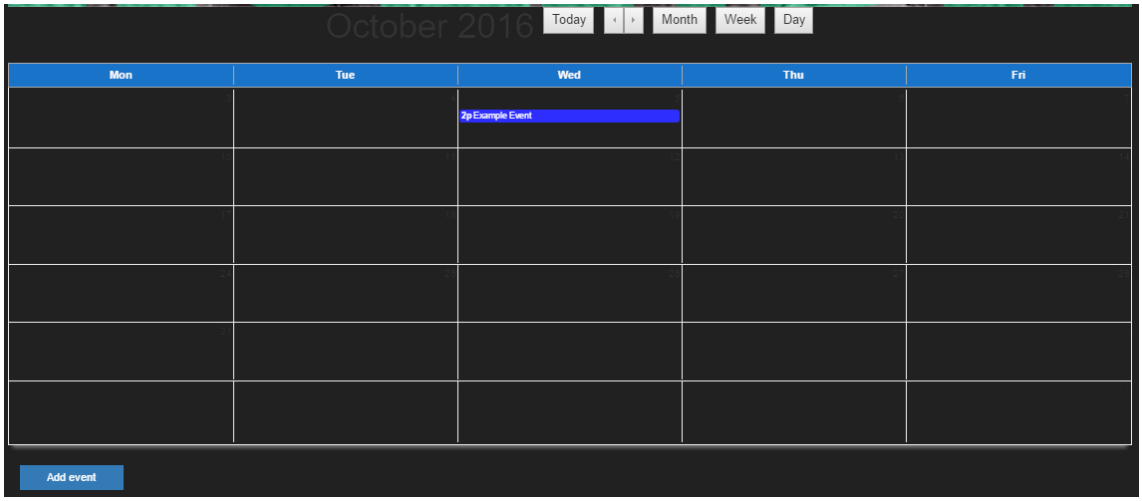


Figura 12 – Captura de pantalla del componente Calendar. Fuente: Autoría Propia

Una de las mayores dificultades que surge a partir de la creación de este componente es la de proveer a los autores una forma simple de administrar los eventos en el calendario.

Debido a la dificultad que presenta esta tarea los diálogos ofrecidos por AEM fueron descartados, ya que la posibilidad de armar una lista dinámica de eventos –similar a los links del componente *Footer*- no sería adecuada dada la gran cantidad de campos que posee cada evento, entorpeciendo su administración.

Por otro lado, el enfoque utilizado para los *speakers* también fue descartado ya que generaría que los autores deban crear una página específica –oculta para la navegación- alojando las instancias del componente que representaría los eventos por cada calendario. Esto implica un sobretrabajo para los autores.

La solución planteada surgió ante la posibilidad de utilizar el propio *framework fullcalendar* para que los autores administren los eventos del componente. Esto es posible creando diferentes vistas que pueden ser accedidas únicamente por los autores para modificar, eliminar o crear nuevos eventos. Estas vistas se comunican mediante *sightly* utilizando AJAX para acceder a la clase *EventCalendar* –encargada de la lógica de negocio, como veremos más adelante-.

La vista a cargo de la creación de un evento, *form-new-event.html,* puede ser accedida por los autores al presionar el botón *Add Event* y es desplegada en forma de *pop-up*. Esta vista (como vemos en la Figura 12) es un formulario cuyas datos viajan al servidor mediante un POST a la clase *EventCalendar,* si el evento es creado satisfactoriamente aparece en el calendario. Cabe destacar que el campo *Speaker* es cargado de forma automática con las instancias del componente *Speaker* incluido en la página -cuya ruta fue definida por los autores en el diálogo del componente-.

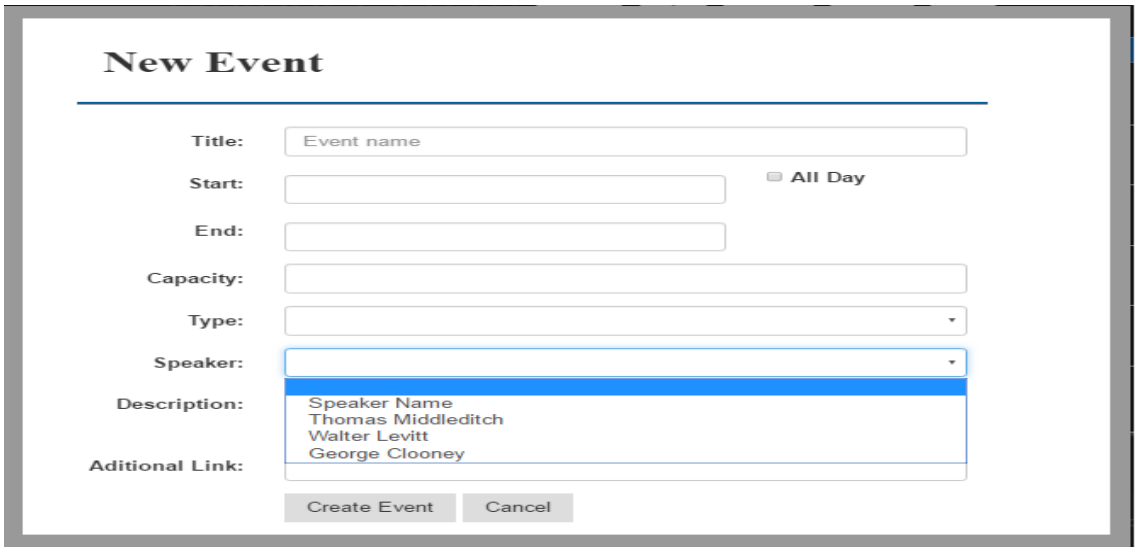


Figura 13 – Captura de pantalla del formulario construido para que los autores puedan crear un evento.

Fuente: Autoria Propia

Por otra parte, los autores al hacer click en un evento creado en el calendario, pueden acceder a una vista similar a la de creación como vemos en la Figura 12, la cual les permite editar o eliminar al evento.

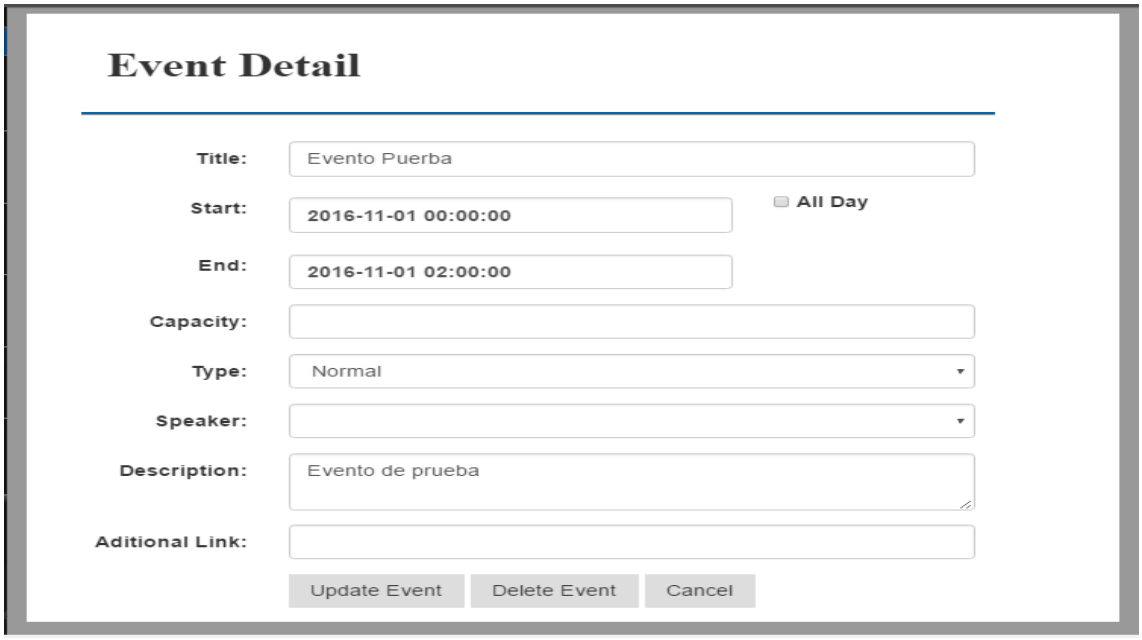


Figura 14 – Captura de pantalla para la edición o eliminación de eventos que es accedida por los autore.

Fuente: Autoria Propia

Por último, existe la vista encargada de mostrar los detalles y las acciones que se pueden desarrollar sobre los eventos del calendario. A la misma acceden únicamente los usuarios del sitio web mediante un click en un evento que despliega la vista en un *pop-up*. Ver Figura 12-

En caso de estar logueado el usuario podrá acceder a un conjunto de acciones ya definidas anteriormente en el caso de estudio, y en caso de no estarlo se ocultará cualquier acción sobre el evento.

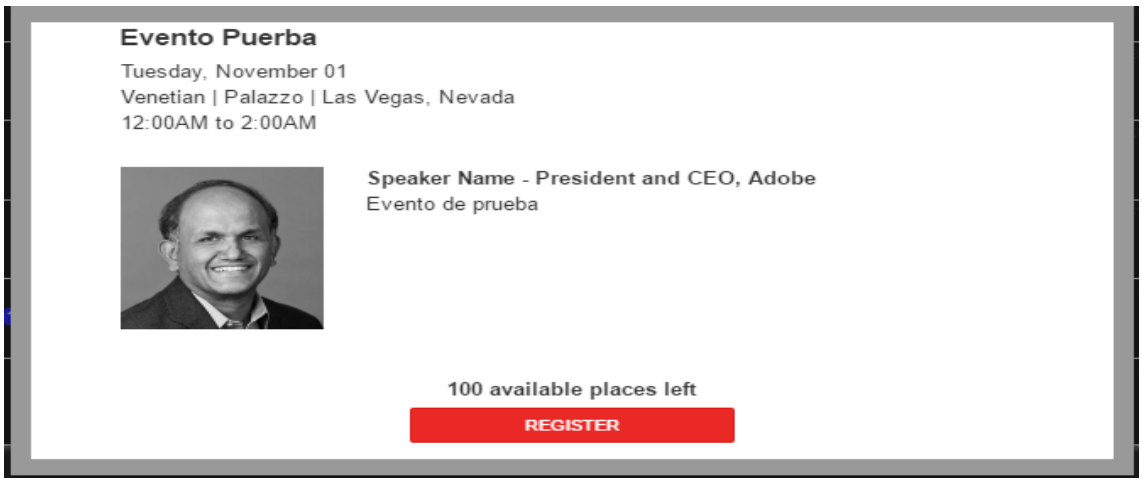


Figura 15 – Captura de pantalla que muestra un formulario con los datos de un evento al cual acceden los usuarios del sitio web.

Fuente: Autoria Propia

1.3.3.6.3 Lógica de negocio

La clase *EventCalendar* extiende de la clase WCMUsePojo lo que permite que las vistas se comuniquen con ella mediante *AJAX*. Es la encargada tanto de la creación de los eventos en forma de nodo, como de sus reservas o eliminaciones. Asimismo, la clase *QueryBuilder* (brindada por la API de AEM) es utilizada para realizar búsquedas dentro del repositorio CRX.

A su vez, la clase *EventCalendar* devuelve una lista de *SpeakerDropBox* a la vista de creación de eventos. A partir de dicha lista se crea el *dropbox* que utiliza el autor para seleccionar el *speaker* asociado al evento. La clase *SpeakerDropBox* contiene dos atributos que representan el nombre y la ruta, los cuales son mapeados con los datos de las instancias del componente *Speaker* -obtenidas a partir de la ruta de la página definida por los autores en el diálogo-.

Como observamos en la Figura 16, los eventos son creados en una jerarquía de nodos que se encuentran en la carpeta *content* de AEM, bajo nuestro sitio *Summit.* La jerarquía consta del número de año, mes y día del evento. A partir de esto, generamos automáticamente un nombre único para cada uno de los eventos, que básicamente constituye un auto-numerado con el prefijo *event*. La raíz de los eventos es la instancia específica del componente *Calendar*, permitiendo que cada instancia del mismo administre independientemente los eventos y reservas. Siendo esto transparente para los autores ya que no deben realizar ningún tipo de configuración especial para que ello suceda. Además, la estructura definida por *año/mes/día* facilita la búsqueda de eventos según fecha.

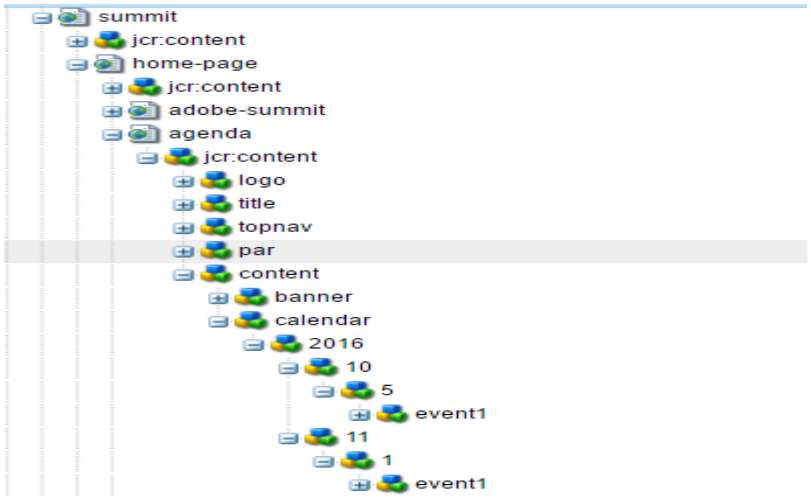


Figura 16 – Captura de pantalla del CRX la cual muestra la jerarquía de nodos que persiste los datos del calendario. Fuente: Autoría Propia

El componente utiliza scripts y estilos específicos definidos en una librería cuya categoría es *summit.calendar.*

2 Manejo global de Librerías

Los componentes *Calendar*, *Speaker*, *User Register* y *Jumbo* definen librerías propias con una categoría específica para cada una de ellas. Dichas categorías son agregadas como dependencias de la categoría global *summit.all.* De esta forma, se realiza una importación de estilos y *scripts* a una única categoría que engloba todas.

Otro enfoque posible sería incluir dentro de *summit.all* a las librerías particulares de los componentes. Si bien su funcionamiento hubiera sido el mismo, creemos que la definición de librerías separadas aumenta la independencia de los componentes y su reutilización en otros proyectos.

3 Solución Móvil

AEM tiene integrado PhoneGap [2], un *framework* creado por Adobe utilizado para desarrollar aplicaciones móviles. El desarrollador no necesita tener conocimientos de lenguaje de programación móvil, sino que sólo lenguajes de desarrollo web como HTML, CSS y Java Script. A partir de estos PhoneGap produce aplicaciones para todas las plataformas de SO móviles más populares, como iOS, Android y Windows Mobile OS, etc. El *framework* explota el concepto de Single Page Apps, el cual consiste como sugiere su nombre en crear aplicaciones de una única página web la cual simula una aplicación móvil -ver Figura 17-. De esta forma y mediante Java Script no perderá los datos que residen en la memoria y se podrá administrar la transición del contenido de un estado visual a otro. AEM ofrece un arquetipo de proyecto móvil el cual utilizamos para crear nuestra aplicación.

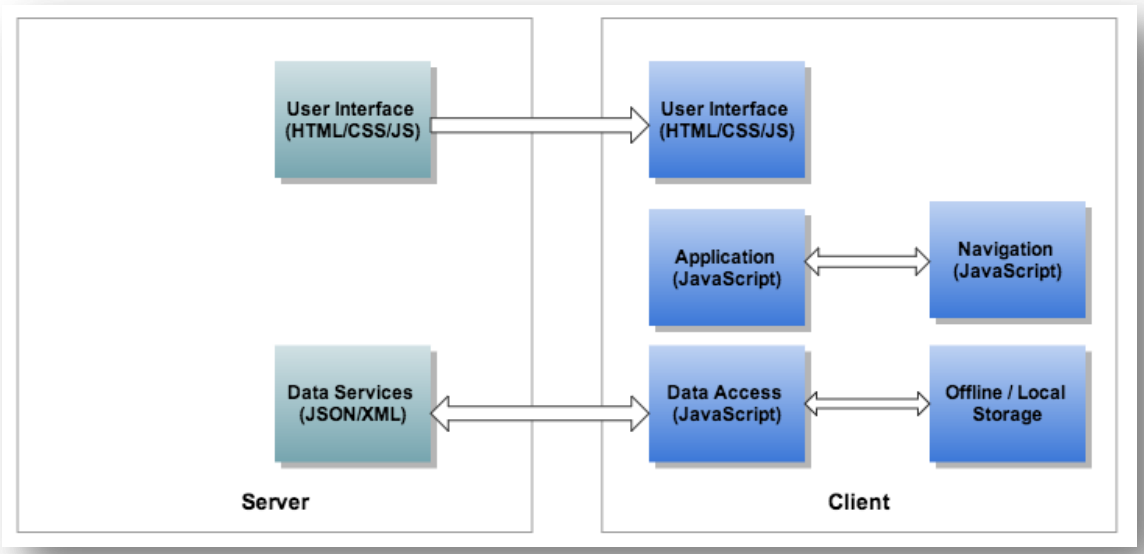


Figura 17 – Esquema de funcionamiento de una aplicación utilizando Phonegap. Fuente: [2]

3.1 Servicios

Utilizando la estructura de dicho proyecto tuvimos que crear cuatro servicios en el *bundle* del proyecto del sitio web que van a ser consumidos por nuestra aplicación móvil mediante AJAX:

3.1.1 Log In

Es el encargado de validar la combinación de usuario y contraseña ingresados por el usuario en la aplicación móvil.

3.1.2 Event List

Obtiene todos los eventos del calendario que comienzan en la fecha enviada en el pedido AJAX como parámetro.

3.1.3 Event Detail

A partir de la fecha y el identificador de un evento, ambos enviados en el pedido AJAX, el servicio devuelve los valores de las propiedades del mismo.

3.1.4 Event Reservations

El servicio se encarga de dar de crear o eliminar una reserva echa por un usuario a un evento. Los parámetros enviados son: la fecha del evento, el identificador, el nombre de usuario y el tipo de operación –alta o baja- de reserva.

3.2 Componentes

Para poder cumplir con los requerimientos de la aplicación móvil creamos tres –Login, EventList y EventDetail- componentes los cuales interactúan con los servicios. Éstos son similares a los componentes del sitio web en cuanto a desarrollo y utilización por parte de los autores.

3.2.1.1 Log In

El log in -Figura 18- de la aplicación móvil envía el usuario y contraseña mediante un servicio REST al servicio de log in antes mencionado, si el usuario y contraseña son correctos se redirecciona a la siguiente vista de la aplicación.

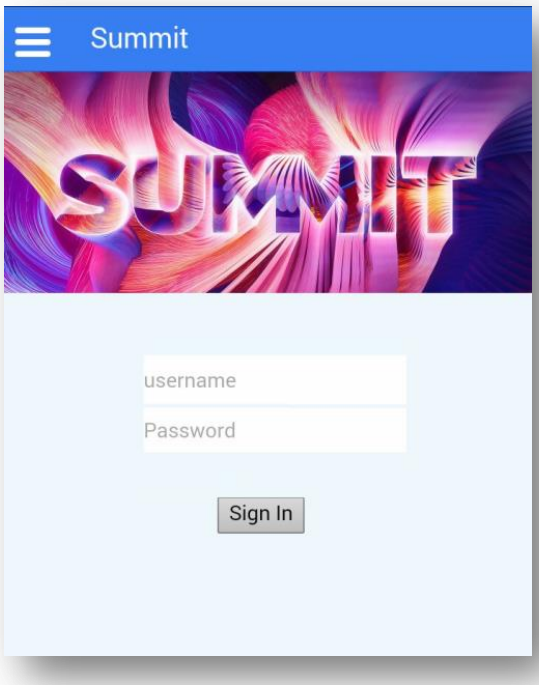


Figura 18 – Captura de pantalla de la aplicación móvil de AEM. Fuente: Autoría propia

3.2.1.2 Event List

El componente *event list* es el encargado de consultar los eventos que existen en el sitio web desde la aplicación móvil. Para lograr este cometido tiene un campo donde el usuario de la

aplicación móvil puede seleccionar una fecha y a partir de esta serán desplegados todos los eventos disponibles -Figura 19-. Por último el usuario podrá seleccionar cualquiera de los eventos desplegados para poder acceder a sus detalles.

3.2.1.3 Event Detail

Por último el componente *event detail* se encarga de desplegar los detalles del evento seleccionado por el usuario de la aplicación móvil -Figura 20-. A su vez le ofrece la posibilidad de realizar o eliminar una reserva del evento desplegado.

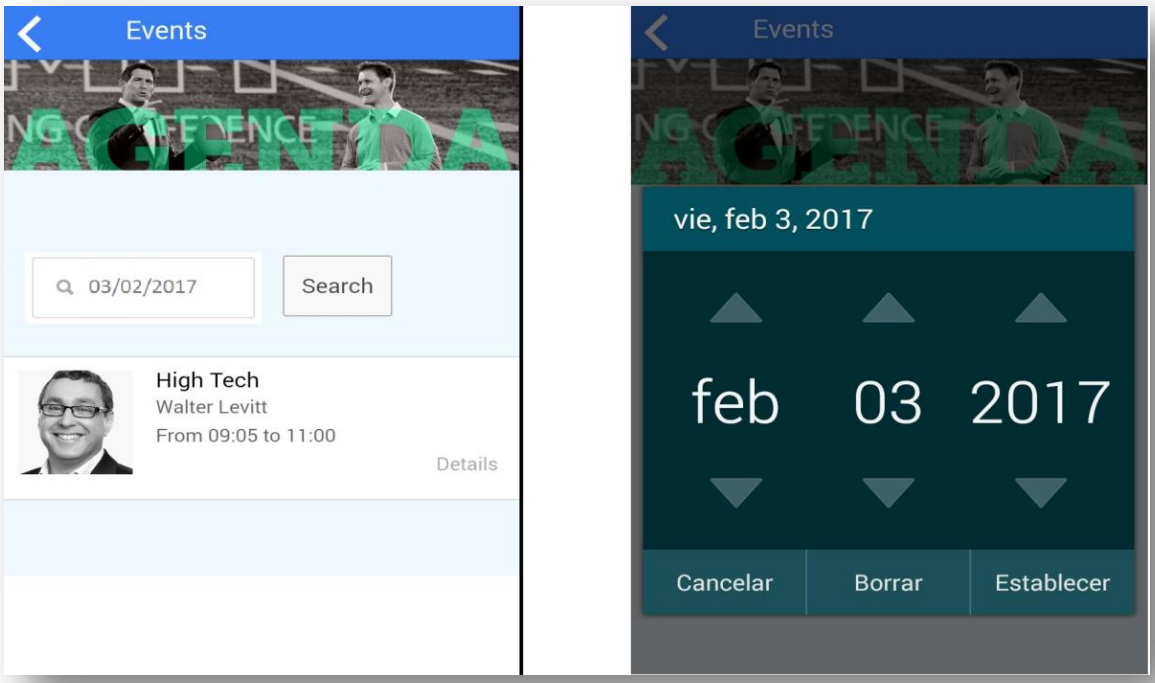


Figura 19 – Dos captura de pantallas diferentes de la aplicación móvil en AEM. Fuente: Autoría propia.

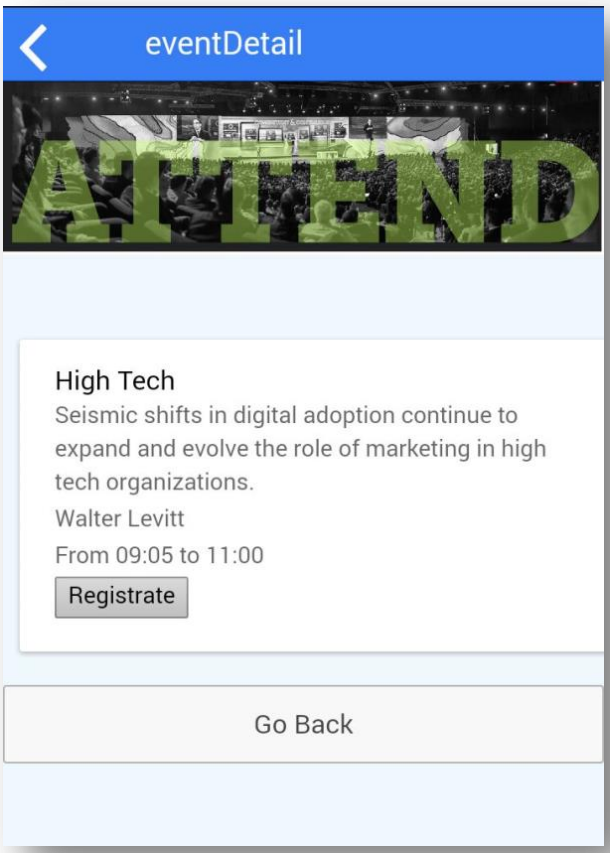


Figura 20 - Captura de pantallas diferentes de la aplicación móvil en AEM. Fuente: Autoría propia.

4 Creación de contenido

Por último y habiendo implementado los componentes definidos anteriormente, pasamos a la creación de las páginas del caso de estudio. Tal como vemos en la Figura 21, la solución está constituida por seis páginas. En primer lugar, observamos el nodo referente a nuestro sitio –*summit Site-*, con dos páginas hijas: *Home Page y Speakers.* La primera será utilizada como página raíz definida en el componente *Topnav* y en la segunda se incluyen las instancias del componente *Speaker* utilizadas por los componentes *Calendar* y *Speaker Box.* Esta última permanecerá oculta para la navegación y el componente *Topnav.*

Por último se crean las páginas *Adobe Attend*, Agenda y *Registration* como hijas de *Home Page*, donde la página *Registration* se le define la propiedad *Hide in Navigation* en verdadero. Se ilustra la jerarquía de páginas creada en la Figura 21.

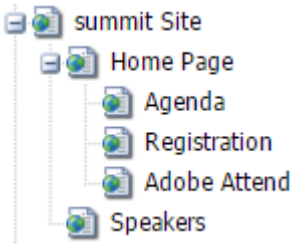


Figura 21 – Captura de pantalla de la jerarquía de las paginas creadas para el sitio. Fuente: Autoría Propia

5 Referencias

[1] https://fullcalendar.io/. Última visita: 23/2/2017.
[2] http://stage.docs.phonegap.com/tutorials/develop/02-single-page-architecture/ (6/12/2016)

Anexo 4 Documentación técnica de Sitecore

Índice

1 Objetivo

El objetivo de este documento es realizar una presentación técnica sobre Sitecore y recorrer los principios básicos que todo desarrollador debe comprender.
La capacidad clave de Sitecore es brindar una plataforma que permite que los autores, que son personas que pueden no poseen conocimientos técnicos, puedan modificar y crear contenido utilizado en distintos canales (web, móvil, social, etc.) sin la necesidad de un desarrollador.

2 Introducción

En el siguiente capítulo se detallarán los conceptos de Sitecore necesarios para comprender la implementación de cada componente de contenido del caso de estudio, especificando las tecnologías involucradas para dicho cometido. Asimismo, la presente documentación tendrá el objetivo de ser utilizada como base para que nuestro cliente, Conexio, pueda iniciarse en el desarrollo de soluciones con Sitecore.
Se presentan los conceptos fundamentales de Sitecore para tal fin, y también se incluye una descripción de su arquitectura.
A su vez luego se incluye un manual técnico que incluye una guía de instalación y configuración de todo el ambiente necesario para el desarrollo, así como ejemplos de desarrollos básicos explicados paso a paso.

3 Historia

Después de trabajar juntos en numerosos proyectos técnicos -tanto dentro como fuera del ambiente académico-, un grupo compuesto por seis graduados de la Universidad de Copenhague (Dinamarca) fundaron Pentia A/S en el año 1998. Se trata de una compañía que brindaba servicios para crear sitios web con tecnologías Microsoft. Años más tarde, la empresa comienza a desarrollar una herramienta a partir de la definición de procesos automáticos para brindar soporte para la creación y administración de sitios web (las cuales, en ese entonces, requerían de un profundo conocimiento en tecnología y programación). Este producto hoy sería clasificado como un WCMS.
Sitecore fue fundada en el año 2001 como una empresa independiente a partir del emprendimiento antes mencionado. Su principal producto es un sistema de manejo de contenido web basado en ASP.NET, construyendo un canal para que sus socios realicen consultorías y entrega de soluciones completas para sus clientes. Sitecore capitalizó el ascenso de ASP.NET a partir del clásico ASP (*Application Server Pages* sin .NET) y aprovechó las debilidades de muchas plataformas WCMS basadas en Java. Así superó a su competencia, dentro de la cual existen grandes empresas ya establecidas a nivel mundial.
Las soluciones en Sitecore son desarrolladas bajo el *framework* .NET, utilizando el lenguaje de programación C#. Al implementarse el proyecto en dicha tecnología se basará en gran parte dentro del patrón de diseño MVC (Modelo Vista Controlador), el cual será detallado más adelante.
Actualmente el producto *Sitecore Experience Platform* (XP) se encuentra en la versión 8.2 que fue lanzada el 15 de setiembre del año 2016 [1]. La versión con la que contamos para este proyecto es la 8.1, la cual fue lanzada el 20 de octubre del año 2015 [1].

4 Arquitectura

Al trabajar con este producto nos obstaculizó el hecho de disponer de muy poca información sobre su arquitectura. Esto pone de manifiesto la dificultad de cara al aprendizaje de esta herramienta, desconocida hasta el inicio de este proyecto.
A continuación se presenta un esquema muy simplificado de la arquitectura de Sitecore, el cual no tiene en cuenta la suite de marketing digital (DMS) ni la separación de entornos.

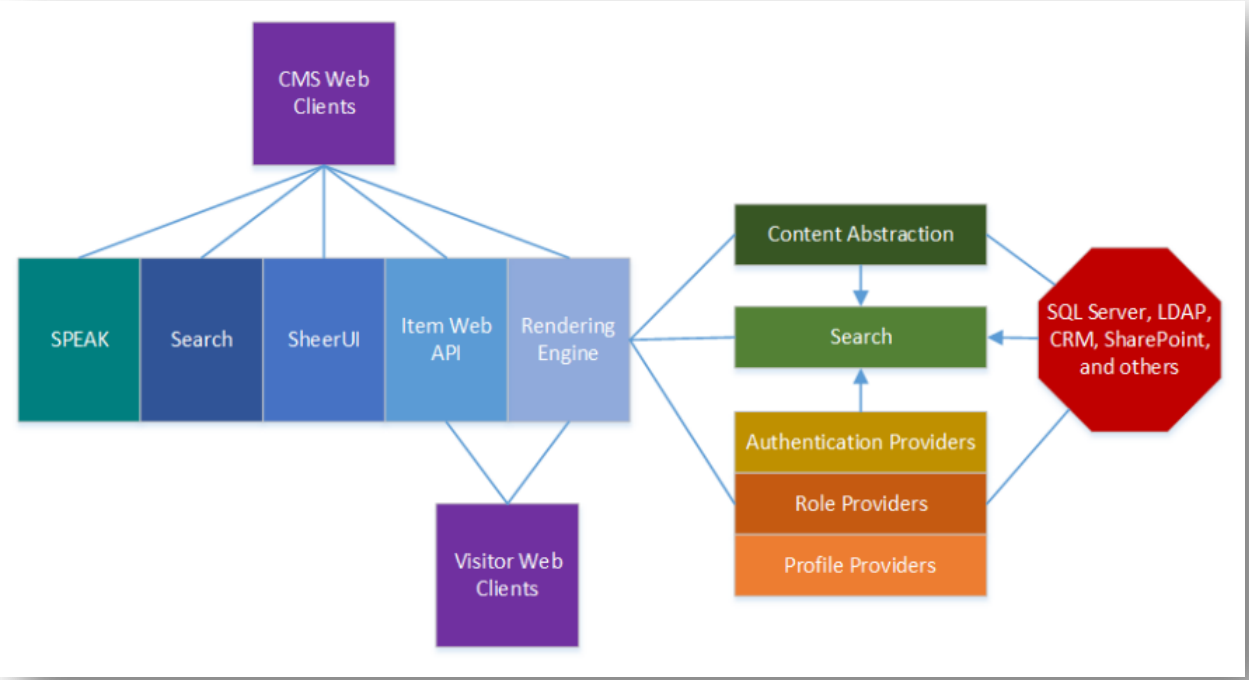


Figura 1 – Esquema de la arquitectura de Sitecore. Fuente: [2]

Tal como observamos en la Figura 1, todas las conexiones son bidireccionales. Las flechas muestran la búsqueda alimentada por la abstracción de contenido, proveedores de membresía y acceso directo a sistemas específicos.

- ***Rendering Engine***
  Encargado del ensamblado dinámico y almacenaje de árboles de control de ASP.NET.
  Arma una jerarquía de controles de presentación y luego determina si los ejecuta o los recupera de una salida guardada en caché previamente bajo las mismas condiciones de procesamiento.
- ***Item Web API***
  Es un módulo cuyo propósito es el de proveer a los desarrolladores acceso programático a la instancia de Sitecore, pudiendo manipular ítems a través de código por medio de pedidos HTTP –Servicios RestFUL-.
- ***Pipelines***
  Sitecore utiliza *pipelines* para implementar cierta cantidad de prestaciones. Cada *pipeline* implementa un procedimiento lógico individual, como por ejemplo definir el contexto de Sitecore para cada pedido HTTP o generar una lista de mensajes de advertencia para un ítem en el editor de contenido.

Cada *pipeline* consiste en una serie de clases procesadoras, donde cada una implementa un aspecto de la función definida por el mismo. Contiene un método llamado *Process*. Un procesador puede abortar un pipeline, previniendo que Sitecore invoque a los procesadores consecuentes.
Los *pipelines* y procesadores soportan separación de responsabilidades entre los aspectos del sistema, proveyendo testeo, configuración y extensión. Se pueden remover procesadores existentes, sobrescribirlos y agregar procesadores propios, teniendo precaución de no afectar la funcionalidad por defecto de Sitecore.

Los recuadros violetas representan clientes accediendo al sistema. Estos pueden ser de cualquier tipo de dispositivos, desde navegadores web a dispositivos móviles. Sobre la capa de servicios los usuarios del CMS pueden acceder a todas las prestaciones de Sitecore, que incluyen:

- Páginas (u otros datos, como RSS o JSON) generados por el motor de renderizado (*ASP.NET Web forms*, *web services*, MVC o cualquier otro).
- Datos obtenidos a través de llamadas a la *Item Web API*.
- Interfaces de usuario generadas con la tecnología Sitecore *Sheer UI*.
- Servicios de búsqueda.
- Interfaces de usuario generadas con la tecnología *Sitecore SPEAK*.

Abajo, los visitantes del/de los sitio/s publicado/s acceden a páginas renderizadas, que pueden invocar la *Item Web API.*
Todas estas técnicas para acceder a los datos gestionados en Sitecore resuelven las llamadas de la API contra la capa de abstracción de contenido de Sitecore, la capa de abstracción de búsqueda y los proveedores de membrecías. Estas capas aíslan el código de *front-end* del de los sistemas *back-end*.

4.1 Capa de Persistencia

Cada instancia de Sitecore depende de cierta cantidad de bases de datos relacionales *Microsoft SQL Server* u *Oracle*. Como el código desarrollado es a través de una capa que abstrae el mecanismo de almacenamiento subyacente (a través de una API de Sitecore), se pueden utilizar diferentes bases de datos en distintos ambientes (desarrollo, prueba) a menos que se interactúe directamente con la base de datos [3].
Sitecore por defecto crea tres bases de datos que cumplen los siguientes propósitos:

- ***Master***

  Contiene la totalidad del contenido -incluyendo los cambios que aún no han sido publicados en el sitio en línea-. En esta base de datos trabajan directamente los desarrolladores y autores antes de publicar cualquier modificación al sitio web en producción.

- ***Web***

  La base de datos web contiene todo el contenido del sitio web ya publicado y que puede ser accedido por los usuarios.

- ***Core***

  En esta base de datos se mantienen todo lo que no refiere al contenido, sino a la configuración y funcionamiento de la propia plataforma.

El esquema de las tres bases de datos es muy similar. La base de datos *Core* difiere un poco con respecto de las otras dos, ya que contiene tablas para funcionalidades adicionales.
Es posible configurar Sitecore para que utilice diferentes bases de datos para cada función.
En cuanto a la publicación en Sitecore, la misma es muy sencilla. Luego de desarrollar alguna sección que esté lista para ser publicada, simplemente se selecciona el contenido que se quiere publicar y el mismo es copiado de la base de datos *Master* a la base de datos *Web*.

4.2 Estructura del árbol de contenido en Sitecore

En la siguiente figura se muestra el árbol de contenido en Sitecore, el cual es accesible a través del editor de contenido

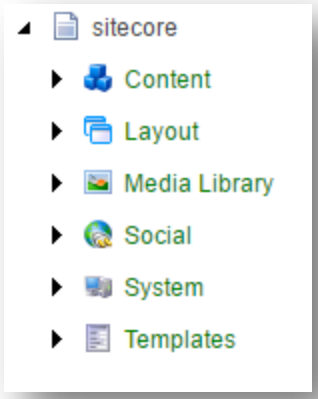


Figura 2 – Captura de pantalla recortada de la interfaz gráfica de Sitecore, muestra su estructura de carpetas. Fuente: Autoría Propia

- ***Content***
  Se almacenan las páginas y la información de las mismas. La estructura de los ítems representa la estructura del sitio web.
- ***Layout***
  Almacena todo lo concerniente a la presentación de las páginas que componen el sitio web, en particular *layouts* y *renderings*.
- ***Media Library***
  Persisten todos los archivos multimedia.
- ***Social***
  Almacena ítems relacionados a aplicaciones que utilicen redes sociales.
- ***System***
  Persisten las herramientas estándar para el editor de contenido y la aplicación web, por ejemplo los idiomas disponibles.
- ***Templates***
  Esta carpeta es la encargada de almacenar todas las plantillas disponibles.

4.3 Stack tecnológico de Sitecore

El *stack* de tecnologías necesarias para una instancia de Sitecore es el siguiente:

- ***Framework***
  ASP.NET.
- **Lenguaje de programación**
  C#.
- **Servidor web**
  IIS.
- **Servidor de base de datos**
  Microsoft SQL.
- **Sistema operativo**
  Windows.

4.4 Tecnologías usadas por Sitecore

4.4.1 .Net

Framework que define un ambiente que soporta el desarrollo y la ejecución de aplicaciones altamente distribuidas y basadas en componentes de *software*. Permite a diferentes lenguajes trabajar juntos y provee de seguridad, portabilidad, y un modelo común de programación para la plataforma de Windows.
El framework define dos entidades muy importantes. La primera es el entorno en tiempo de ejecución de lenguaje común (CLR en su sigla en inglés), es el sistema que maneja la ejecución del programa desarrollado. La segunda entidad es la librería de clases de .NET, la cual brinda acceso al entorno de ejecución [4].

4.4.2 C#

Es un lenguaje de programación de propósito general, con seguro de tipos y orientado a objetos [5]. Es neutral en cuanto a plataformas, pero fue escrito para trabajar correctamente con el *framework* .NET [5]. Además de que las bibliotecas definidas para el citado *framework* son las usadas por C#, por lo tanto los programas escritos en C# son automáticamente portables a cualquier ambiente .NET, por lo tanto aunque es teóricamente posible separar a C# del ambiente de .NET, ambos están íntimamente ligados [4].

4.4.3 IIS

Sigla correspondiente a *Internet Information Services*. Servidor web para Windows [6].

4.4.4 Microsoft Visual Studio

Entorno de desarrollo integrado con todas las características para Android, iOS, Windows, la web y la nube [7].

4.4.5 Microsoft SQL Server

Sistema de administración y análisis de bases de datos relacionales de Microsoft [8].

4.5 Conceptos de desarrollo en Sitecore

Con el fin de dar una introducción a los conceptos utilizados para la solución del caso de estudio, a continuación se definen los principales elementos que forman parte del desarrollo de un proyecto en Sitecore.

4.5.1 Ítem

Los ítems de Sitecore representan recursos individuales del CMS. Un ítem puede representar cualquier tipo de dato: páginas, secciones, metadata, etc. Cada ítem contiene un número de propiedades común a todos los lenguajes (inglés, español, etc.).
Haciendo una analogía con el paradigma de programación orientada a objetos, un ítem equivaldría a una instancia de una clase (objeto).
Cada ítem existe dentro de una jerarquía de elementos en la base de datos de Sitecore. La ruta de un ítem identifica su ubicación dentro de la jerarquía y Sitecore asigna un identificador global único (GUID, o simplemente ID) para cada ítem.
Éstos no solo contienen datos sino que además se comportan como subdirectorios capaces de contener otros ítems, esto implica que no es necesario crear un ítem determinándolo únicamente para un subdirectorio o un archivo, sino que el ítem en el futuro podría contener otros.
Todos los ítems presentan cinco propiedades en común:

- ***Nombre***
  Se utiliza para su identificación, puede no ser único.
- ***Key***
  Es generado automáticamente por Sitecore a partir del nombre.
- ***Path***
  Se define mediante el nombre del ítem y sus ancestros en orden, separados por el caracter '/'. La ruta es construida dinámicamente y puede ser modificada si el ítem es cambiado de ubicación.
- ***ID***
  Identificador único de un ítem.
- ***Data Template***
  Esta propiedad indica el ID del *data template* a partir del cual fue creado el ítem (será explicado más adelante).

Sitecore organiza todo su contenido en forma de directorio, lo que permite identificar a los ítems de manera sencilla a través de su ruta. También se puede referir a un ítem a través de su ID, ya que el mismo es único para cada instancia en Sitecore.

4.5.2 Data *template*

Los *Data Templates* (a partir de ahora *templates*) definen la estructura de los ítems. Nuevamente es posible realizar un paralelismo con el paradigma orientado a objetos, donde un *template* ocuparía el lugar de una clase, en la cual se definen los atributos. A su vez, los ítems constituirían las instancias de dichas clases donde se definen los valores de los atributos.

Dentro de los *templates* es posible definir cero o más secciones, las cuales son utilizadas para agrupar diferentes campos del *template*. Entre otras, su función es la de proveer mayor facilidad de configuración para los autores, agrupando campos que cumplen una función en conjunto.

Podemos ilustrar lo descripto anteriormente en las figuras Figura 3 y Figura 4, donde los campos son agrupados en las secciones y a cada uno se le definirá al menos un nombre. De esta manera será identificado por los autores.

Sitecore dispone de una gran variedad de tipos de datos que pueden ser utilizados para definir diferentes campos. Adicionalmente, es posible que los desarrolladores definan tipos de datos específicos. A su vez, observando nuevamente las figuras Figura y Figura 4, podemos corroborar que cada ítem puede asignar valores para los campos definidos en el *template* donde fue creado.

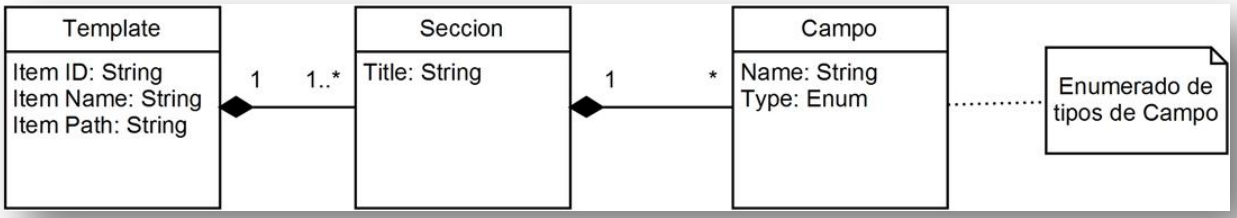


Figura 3 – Modelado de conceptos de *template*, sección, campo e ítem de Sitecore. Fuente: Autoría Propia

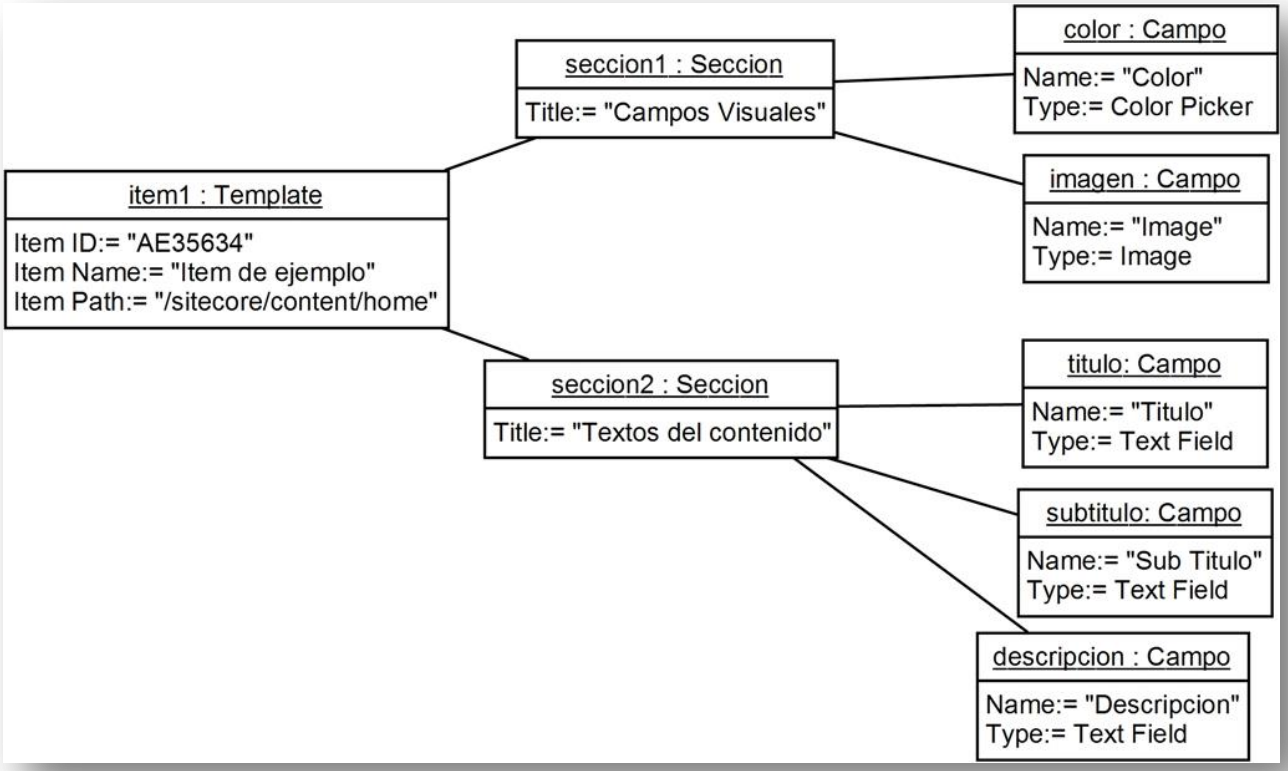


Figura 4 - Diagrama de instancias de conceptos de Sitecore. Fuente: Autoría Propia

4.5.2.1 Standard Values

Los *templates* ofrecen la posibilidad de definir valores por defecto mediante los *standard values*. Esto es posible no sólo para los campos sino también para la capa de presentación – se verá más adelante-. Recurriendo nuevamente al paralelismo con el paradigma de programación orientada a objetos, los *standard values* corresponderían a los valores asignados en la creación por defecto de una instancia.

De esta forma se evita tener que definir manualmente la configuración de cada ítem, sin embargo, los autores tienen la posibilidad de una vez creado el ítem modificar estos valor a otros que prefiera.

Como observamos en la Figura 5, los *Standard Values* son ítems persistidos como hijos del *template* al cual pertenecen y son creados de forma automática con el nombre __*Standard Values*. Dentro de este ítem el desarrollador puede definir tanto los valores por defecto para cada campo definido en el *template*, como cualquier otro tipo de configuración la cual va a ser replicada a cada ítem que se cree a partir del mismo.

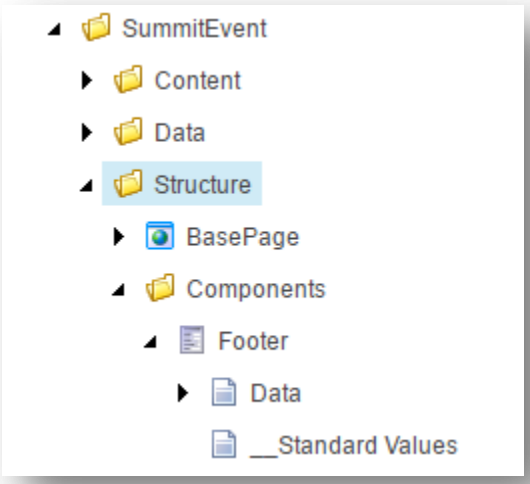


Figura 5 – Captura de pantalla mostrando la estructura de un *template* con sus *standard values*. Fuente: Autoría Propia

4.5.2.2 Herencia entre Templates

Volviendo a utilizar el paradigma de programación orientada a objetos, los *templates* de Sitecore tienen la capacidad de heredar las características de otros múltiples *templates*. Esto incluye las secciones, campos y *Standard Values*, entre otros.

4.5.2.3 Layouts Details

Es posible configurarle a cada ítem de Sitecore diferentes aspectos que refieren a su presentación visual, los cuales son denominados y agrupados como *Layout Details.* De esta manera, el desarrollador es capaz de definir la forma en que Sitecore renderiza los ítems.

Como vemos en la Figura 6, el *Layout Details* de un ítem, permiten al desarrollador o al autor definir diferentes tipos de presentaciones a partir de ítems *Layouts* -que veremos a continuación-, diferenciando según los dispositivos desde los cuales se accede.

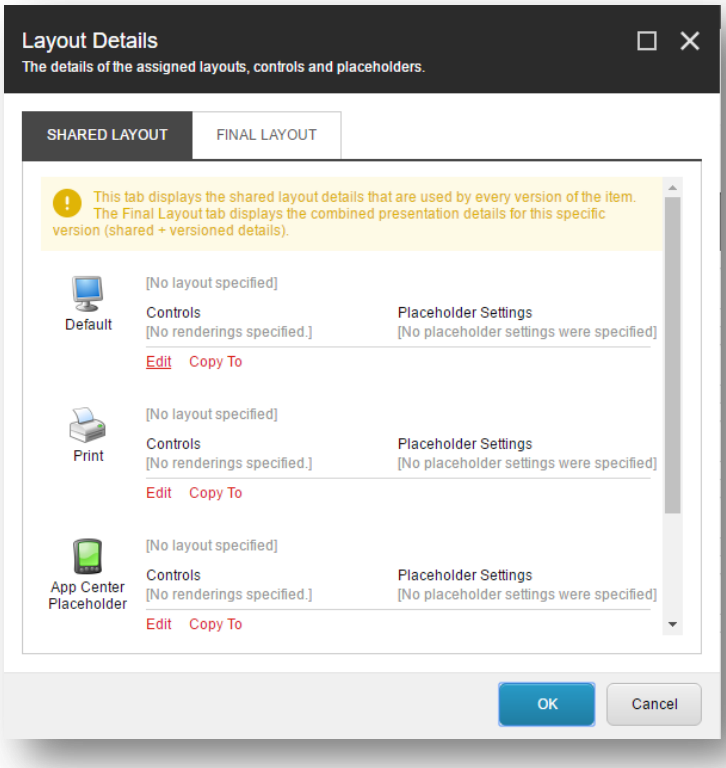


Figura 6 – Captura de pantalla mostrando la interfaz gráfica que ofrece Sitecore para configurar los *Layout* Details. Fuente: Autoría Propia

4.5.3 Layout

Sitecore provee un *template* llamado *Layout* que permite crear ítems para definir una presentación visual. Estos ítems (a partir de ahora *layouts*) son utilizados para definir el renderizado. Esto es posible a partir de la definición de su campo *path* que especifica la ruta para relacionarlo con un archivo –HTML, CSHTML, ASPX, etc.-. De esta forma el *layout* puede ser utilizado por diferentes ítems asociándolo a través de su configuración de *Layout Details* (ver Figura y Figura).

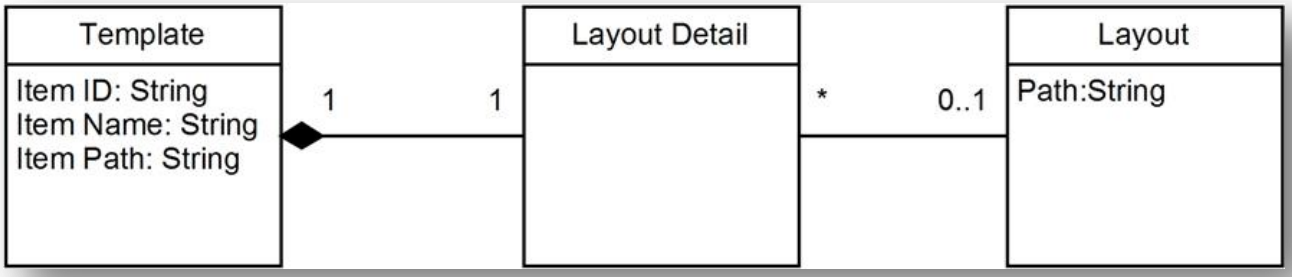


Figura 7 – Esquema de conceptos de la capa de presentación Layout Detail – Layout. Fuente: Autoría Propia

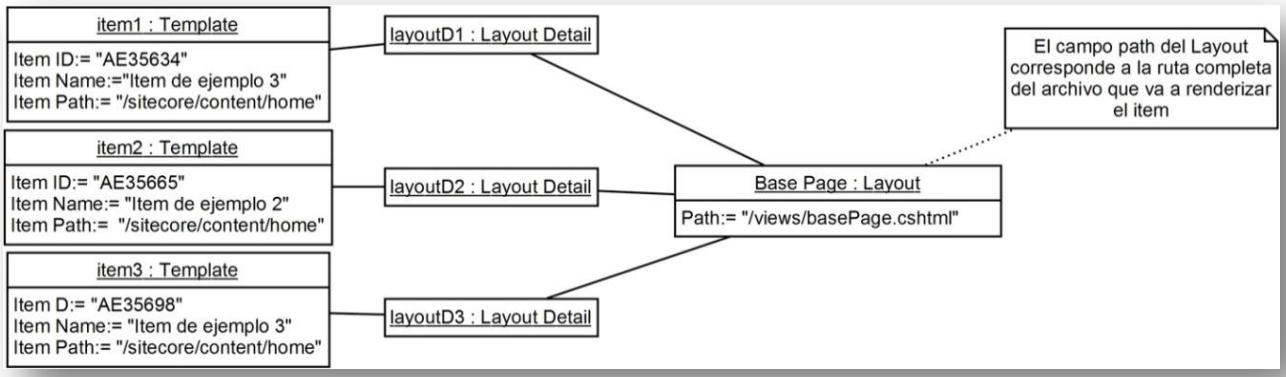


Figura 8 – Diagrama de instancias de conceptos de la capa de presentación Layout Detail de Sitecore. Fuente: Autoría Propia

Para facilitar el uso de nuestra solución, queremos asegurarnos que todos los ítems creados a partir de un *template* específico, sean inicializados con un *Layout* determinado. Esto es posible asociando el *Layout* a los *Standard Values* del *template* en cuestión. Si nosotros realizamos la configuración antes descrita nos aseguramos que todos los ítems creados a partir de dicho *template* tengan relacionado el *Layout* asociado a los *Standard Values*.

4.5.4 Rendering

Sitecore brinda un conjunto de *templates* que llamaremos *renderings*, los cuales son utilizados para construir ítems que representen componentes de presentación individual, cuya función es construir bloques para sitios web. Algunos de sus posibles usos son para:

- Renderizar contenido en una página, el típico caso de un componente en un sistema WCMS.
- Obtener datos de sistemas externos.
- Ejecutar lógica de backend con componentes no visuales, como solicitudes de registro para analítica web u otros propósitos.

Utilizaremos dos de estos *templates* recién mencionados. Los mismos se adaptan a nuestras necesidades y se basan en el patrón MVC: *View Rendering* y *Controller Rendering*.

4.5.4.1 *View rendering*

Se utiliza en los casos donde el desarrollo involucrado no contiene operaciones lógicas complejas ni procesamiento de información. Para este tipo de rendering no es necesario utilizar un controlador, por lo que los únicos componentes del patrón MVC usados son la vista y el modelo – opcional-.

El campo fundamental para la utilización de este tipo de *rendering* es denominado *path* -en el cual el desarrollador debe ingresar la ruta relativa a un archivo-. Ver figuras Figura 9 y Figura 10Figura .

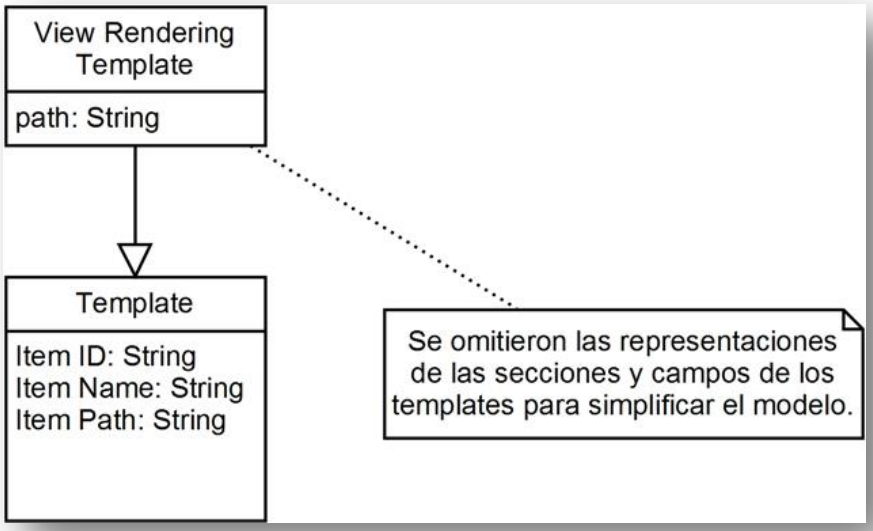


Figura 9 – Esquema de conceptos de View Rendering. Fuente: Autoría Propia

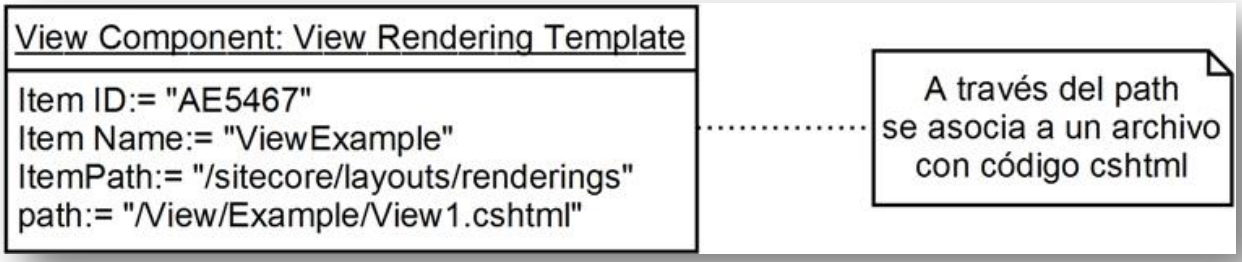


Figura 10 – Esquema de instancias de conceptos de View Rendering. Fuente: Autoría Propia

Además, es posible asociar un modelo específico al *View Rendering* -el cual es enviado automáticamente a la vista -. Con este fin, Sitecore provee un *template* llamado *Model* a partir del cual debemos crear un ítem. El mismo consta de un único campo llamado *Model Type* utilizado para

referenciar una clase C# (la que será ejecutada como modelo). La clase debe implementar la interfaz *IRenderingModel* definida por la API de Sitecore la cual cuenta con un método de inicialización que es ejecutado antes de ser enviado a la vista y que debe ser implementado para obtener los valores de los campos del *rendering*. Tal como se ilustra en las figuras Figura 11 y Figura 12Figura , el ítem modelo antes mencionado debe ser asociado con el *View Rendering* a través del campo *Model* ingresando su ID o su ruta completa.

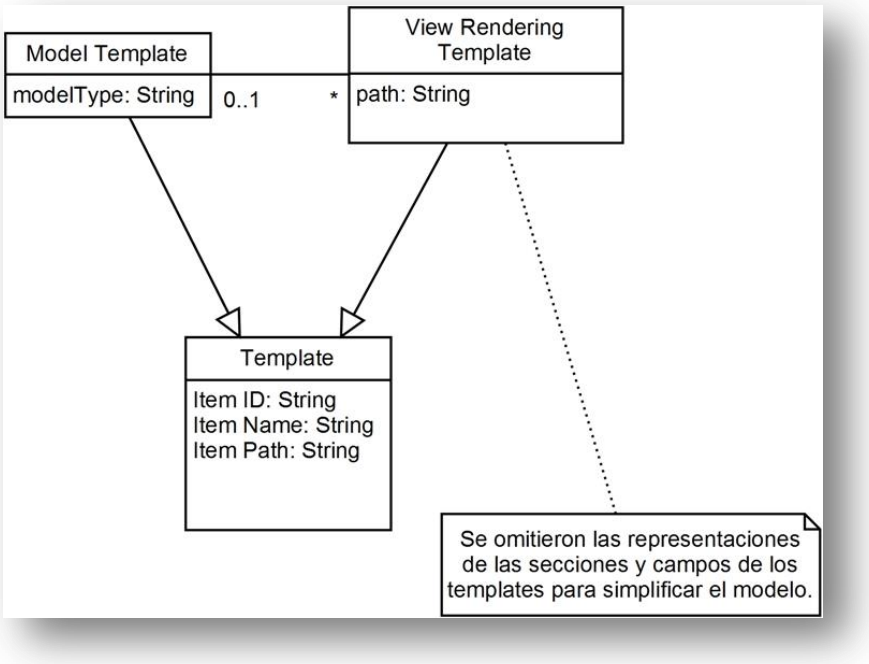


Figura 11 – Esquema de conceptos del View Rendering con Modelo. Fuente: Autoría Propia

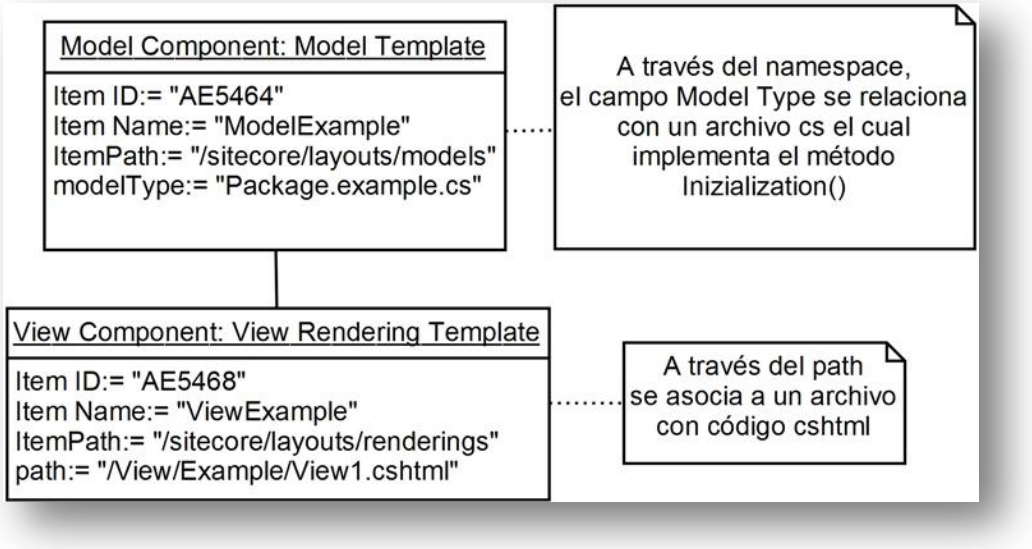


Figura 12 - Esquema de instancias de conceptos del View Rendering con Modelo. Fuente: Autoría Propia

4.5.4.2 *Controller rendering*

Referencia a una clase utilizada como controlador - archivo de extensión cs - y a un método de la misma. Existen dos campos fundamentales para configurar el *controller rendering*:

Controller

Por lo general los nombres de los controladores en proyectos ASP.NET MVC *Framework* son de la forma *NombreControladorController*, pero para su referencia desde Sitecore es suficiente con ingresar NombreControlador en el campo *controller*.

Controller Action

El campo es utilizado para ingresar el método que va a ser ejecutado cuando se renderice el *controller rendering*.

Es recomendada su utilizacion para desarrollos que requieran de operaciones lógicas, ya sean de mayor complejidad o más simples, o un mayor procesamiento de información.

A su vez este tipo de *rendering* permite crear un modelo dentro de cada método del controlador y que sea enviado a la vista completando las tres componentes del patrón MVC. Asimismo el desarrollador puede asociar varios *controller renderings* a un mismo controlador y asignarle a cada uno un método diferente (ver figuras Figura 13Figura y Figura 14).

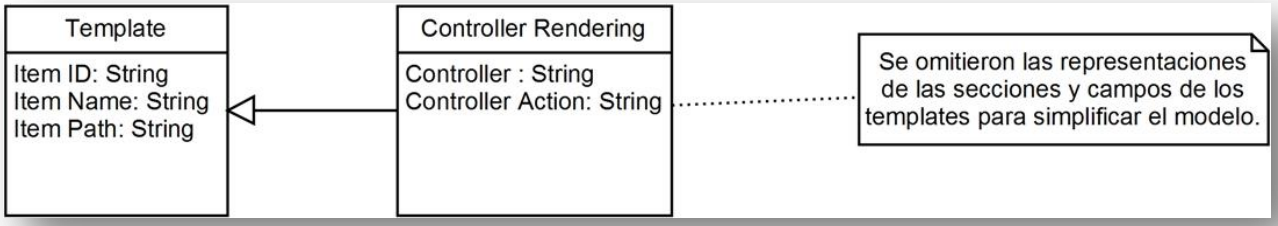


Figura 13 – Esquema de conceptos que participan en un *Controller Rendering*. Fuente: Autoría Propia

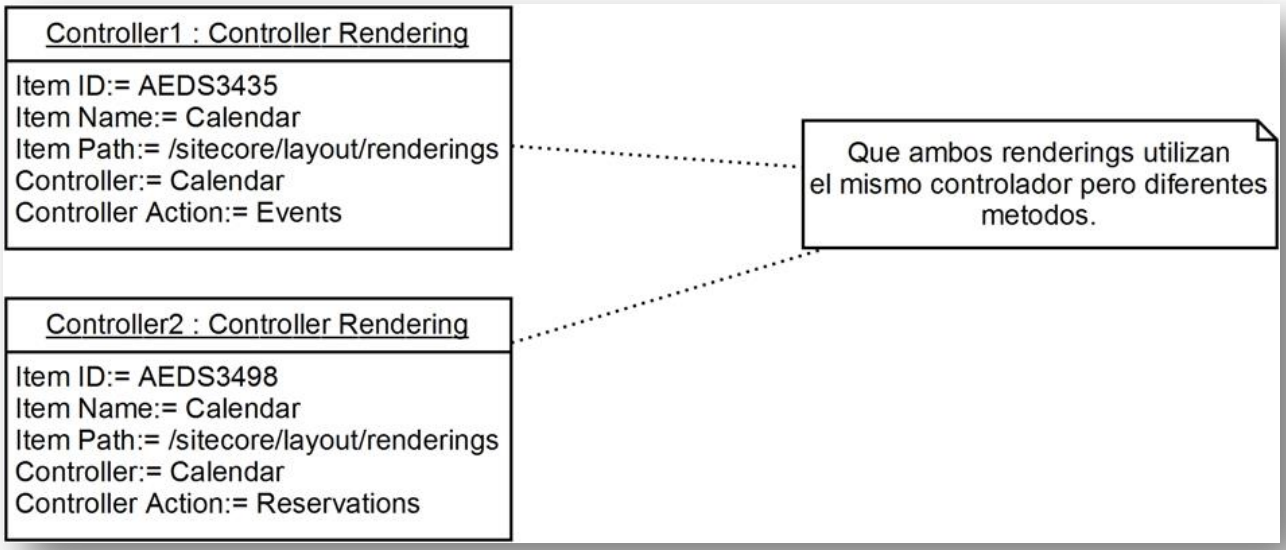


Figure 14 - Esquema de instancias de conceptos que participan en un *Controller Rendering*. Fuente: Autoría Propia

4.5.4.3 Diferentes tipos de parámetros que tiene un rendering

En Sitecore existen dos maneras diferentes de parametrizar un *rendering*, las cuales se detallan a continuación:

4.5.4.3.1 Asignación de un parameter template

Existen *templates* especiales los cuales son encargados de definir los campos de configuración o contenido de un *rendering*, los mismos deben extender como vemos en la Figura 15 del *template Standard Rendering* Parameters que ofrece Sitecore.

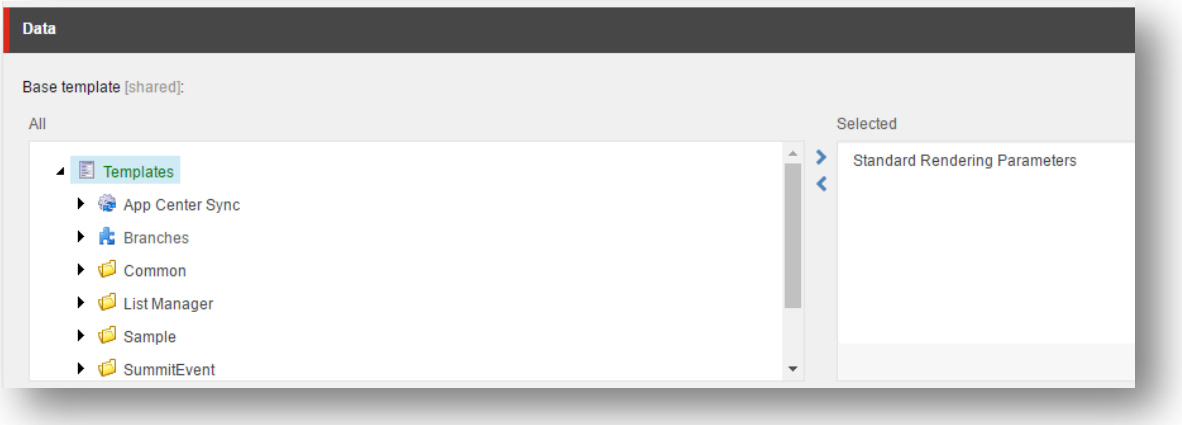


Figura 15 – Asignación de Standard *Rendering* Parameters. Fuente: Autoría Propia

El desarrollador define los campos dentro del *template* para luego asociarlos al *rendering* en cuestión que quiere parametrizar. Para asociarlos, el desarrollador debe asignar la ruta o ID específico del *template* en el campo *parameters template* del *rendering*, como se muestra en la siguiente figura (Figura 16):

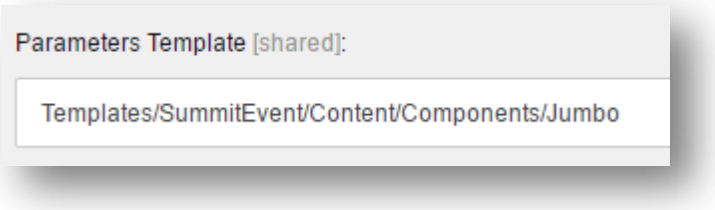


Figura 16 – Asociación de *rendering* a parametrizar mediante ruta. Fuente: Autoría Propia

De esta forma los autores tendrán una ventana específica –similar a un dialogo en AEM- para cada instancia del *rendering*, mediante la cual podrán interactuar e ingresar valores a los campos definidos por el Standard *Rendering* Parameter. Estos valores son independientes entre cada instancia del *rendering*.

4.5.4.3.2 Asignación de un ítem como fuente de datos

En la propiedad *data source* del *rendering* se introduce la ruta o ID del ítem que va a oficiar de fuente de datos, como se muestra en la Figura 17.

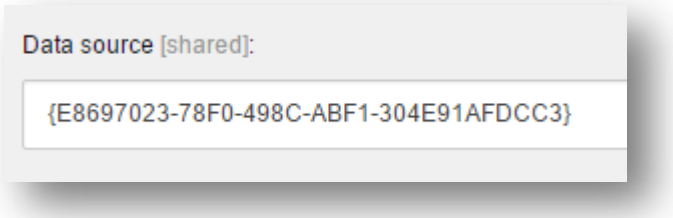


Figura 17 – Asociación a fuente de datos mediante ID. Fuente: Autoría Propia

Dicho ítem puede ser creado a partir de un *template*, el cual no tiene ninguna propiedad particular

Desde el punto de vista del autor, al utilizar ítems como fuente datos, es posible realizar edición en linea, es decir como observamos en la Figura 18 es posible editar ciertos valores sobre la vista del *rendering*.

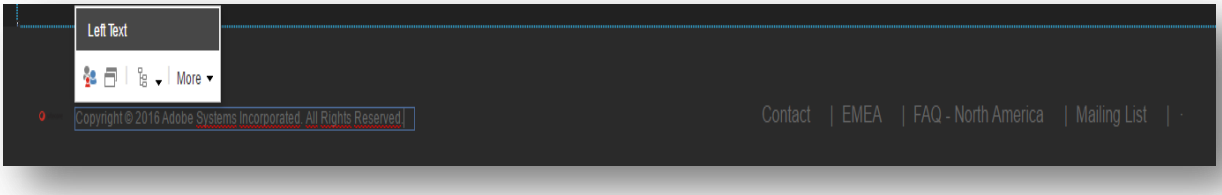


Figura 18 – Edición en línea de texto sobre vista del *rendering*. Fuente: Autoría Propia.

A su vez esta forma de parametrización permite a los desarrolladores asociar un mismo ítem de fuente de datos a varias instancias de *renderings*, provocando que dichas instancias compartan los valores de los campos del ítem antes mencionado. Por lo tanto se desprende que esta opción planteada resulta de mucha utilidad cuando se necesita desplegar el mismo contenido de un *rendering* en varios lugares de la misma página o través de distintas páginas del sitio web. Ahora bien, si el autor pretende que los valores de los campos se mantengan independientes por cada instancia del *rendering*, deberá crear un ítem de fuente de datos para cada uno.
Por último es importante destacar que las dos alternativas descritas pueden combinarse en un mismo *rendering*.

4.5.5 Placeholder

Los *placeholder* –similar a los *parsys* de AEM- son controles de ASP.NET que son utilizados como contenedores de *renderings*, de esta forma los autores son capaces de agregar, mover o quitar dichos *renderings* de forma dinámica.

Los *placholders* son definidos dentro de los archivos HTML, CSHTML, ACXP de renderizado de los *layouts; así* los desarrolladores son capaces de definir qué área de los archivos va a poder ser utilizada por los autores para agregar componentes de contenido. Asimismo los desarrolladores son los encargados de definir una *key* única para cada *placeholder*.
Adicionalmente, es posible definir un ítem a partir del *template Placeholder* definido por Sitecore para configurar el comportamiento del mismo, este consta de tres campos:

- ***Placeholder* Key**
  El campo indica la *key* del *Placeholder* que está asociado a la configuración que se está definiendo.
- **Allowed Controls**
  Este campo define los *renderings* que son permitidos utilizar en el *placeholder*.
- **Description**
  En este campo el desarrollador puede escribir información relevante para los autores en cuanto al uso del *placeholder*.

4.6 Contexto y APIs de Sitecore

4.6.1 Contexto

Define un contexto de procesamiento para cada pedido HTTP. Contiene información sobre el usuario, el ítem requerido, el dispositivo que realizó el pedido, el sitio administrado, y más información asociada con el pedido HTTP correspondiente.
Se citan algunas de las propiedades estáticas más importantes que se pueden consultar a través del contexto de Sitecore:

- **Database**
  Base de datos Sitecore asociada con el pedido HTTP, o base de datos por defecto para el contexto del sitio.
- **Item**
  Ítem solicitado en la base de datos de contexto por el cliente web (basado en la ruta de la URL solicitada)
- **User**
  Usuario autentificado o usuario anónimo en el contexto del dominio de seguridad.

4.6.2 APIs

Sitecore provee de una completa gama de APIs que permiten realizar una gran cantidad de funcionalidades sobre la instancia. Para nuestro objetivo, las operaciones más utilizadas fueron las de acceso, creación y borrado de ítems. Se enumeran las principales APIs, usadas en este proyecto.

4.6.2.1 Sitecore.Data.Database

La clase *Sitecore.Data.Database* representa una base de datos en Sitecore, que puede ser la base de datos *Master* o *Core* o una base de datos de destino de publicación.
Para etapas de desarrollo, siempre se debe acceder a alguna de las bases de datos a través de esta API y nunca directamente a través de consultas SQL.
Adicionalmente, esta API provee una función de mucha utilidad, la cual es la responsable de recuperar un ítem: se trata del método *GetItem*, el cual tiene varias opciones de uso según los parámetros que se le pase a la función.
En el caso de invocar a la función solamente con un parámetro que corresponde a su ID o ruta, se recupera el ítem en su versión actual. El método provee de otras firmas que permiten la recuperación de un ítem para una versión y/o idioma determinado.

4.6.2.2 Sitecore.Data.Items.Item

La clase Sitecore.Data.Items.Item representa un ítem en una de las bases de datos de Sitecore.
Provee de las siguientes propiedades, entre muchas otras:

- ***Children***
  Lista los hijos del ítem.
- ***Database***
  Referencia a la base de datos que contiene el ítem.
- ***Fields***
  Provee acceso a los campos del ítem.
- ***ID***
  Identificador del ítem dentro de la base de datos de Sitecore.
- ***Key***
  Nombre del ítem en letra minúscula.
- ***Language***
  Lenguaje del ítem
- ***Languages***
  Lenguajes disponibles para el ítem.
- ***Parent***
  Padre del ítem, o retorna resultado nulo si se trata del ítem raíz.
- ***ParentID***
  ID del ítem padre.
- ***Template***
  Data template asociada con el ítem.
- ***TemplateID***
  ID del *template* asociadp con el ítem.
- ***TemplateName***
  Nombre del *template* asociado con el ítem.

5 Manual técnico Sitecore

5.1 Instalación Sitecore

5.1.1 Requisitos

Sitecore tiene ciertos requisitos para poder funcionar en un sistema computador, antes que nada Sitecore es una plataforma que utiliza .NET por lo cual el sistema operativo a utilizar debe ser Windows.

Se deben tener instaladas las siguientes aplicaciones:

- ISS 8.5.
- .NET *Framework* 4.5.
- *Visual Studio* 2012 o alguna versión posterior.
- Base de Datos *Microsoft SQL Server*.
- Base de Datos MongoDB.

Las herramientas que por experiencia recomendamos que sean instaladas/activadas son:

- SQLServerManager (administrador de servidores de bases de datos Microsoft SQL Server)
- Inetmgr (administración de Servicio IIS)
- *SQL Server Manager Studio* (permite administrar la base de datos y usuarios)

5.1.2 Instalación

Utilizando el ejecutable que brinda Sitecore se abre el instalador. Esta aplicación es la encargada de instalar una instancia de Sitecore en el sistema. Se deben seguir los siguientes pasos para completar correctamente la instalación:

- En el primer paso simplemente nos va a identificar que versión de Sitecore estamos instalando e información de la patente como se ve en la Figura 19.

Figura 19 - Primer paso de instalación de Sitecore. Fuente: Autoría Propia

- Debemos indicar que queremos instalar una nueva instancia en el sistema, la segunda opción es utilizada para borrar o reparar instancias ya creadas (Figura 20).

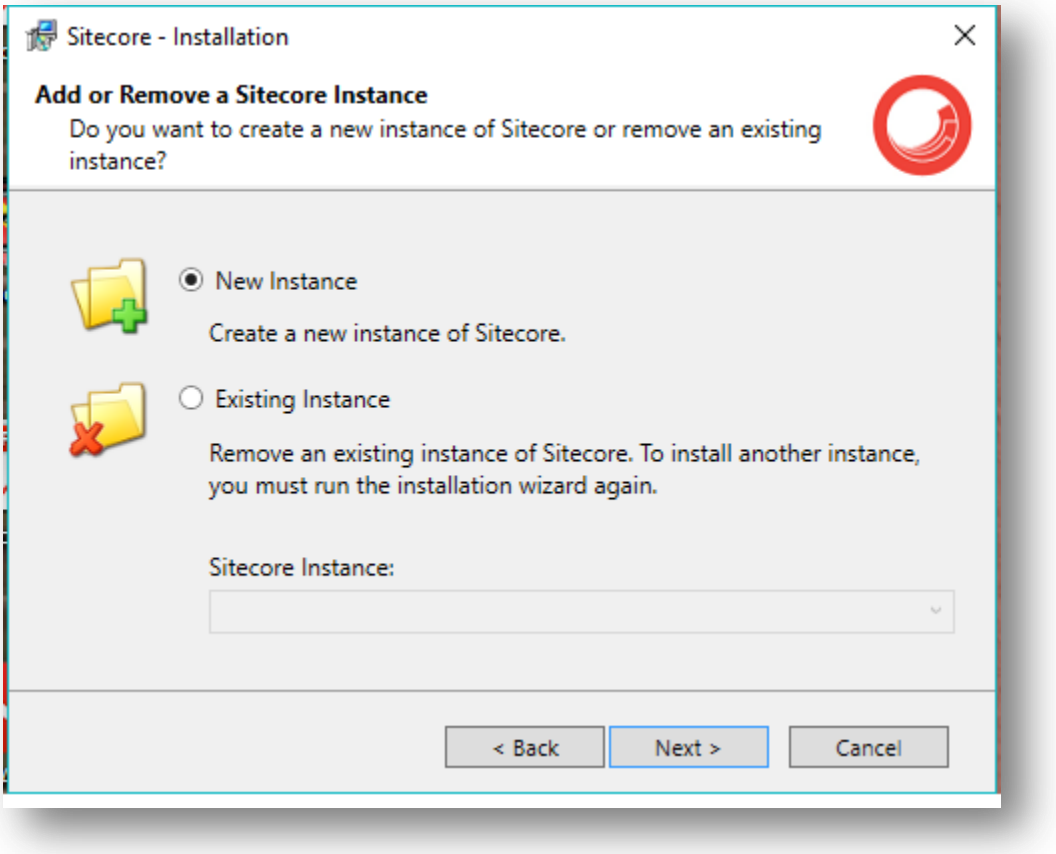


Figura 20 – Etapa de la instalación de Sitecore para creación o remoción de instancia. Fuente: Autoría Propia

- En el paso 3 debemos indicar el nombre que deseamos ponerle a nuestra instancia.

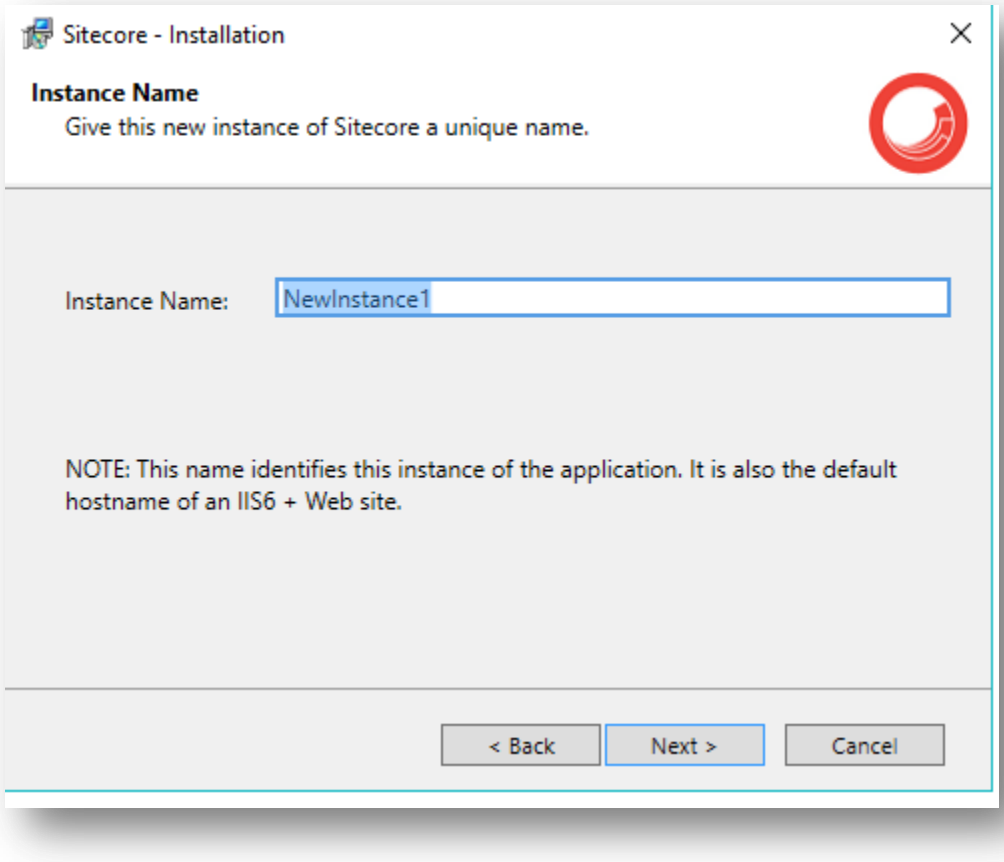


Figura 21 – Ingreso de nombre de instancia Sitecore a instalar. Fuente: Autoría Propia

- Debemos indicar que base de datos estamos usando, su ubicación y un usuario con todos los permisos (Figura 22).

```
TIP:
SQL Server tiene un el usuario 'sa' por defecto con todos los permisos, la contraseña varía según la versión de la base de dato. De todas formas es recomendable crear un usuario nuevo con el Management Studio.
```

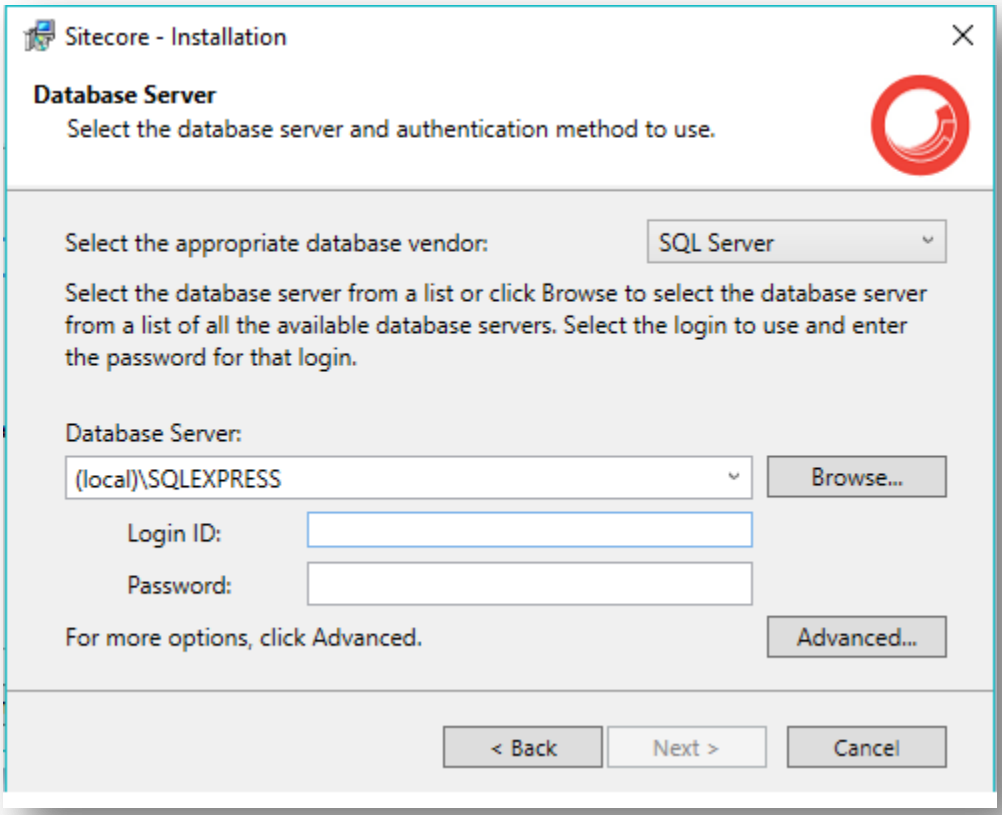


Figura 22. Selección de servidor de base de datos. Fuente: Autoría Propia

- Debemos indicar el lugar donde queremos instalar la instancia en el sistema (Figura 23), por defecto la instancia va a ser creada en *C:\Inetpub\wwwroot\NombreInstancia*

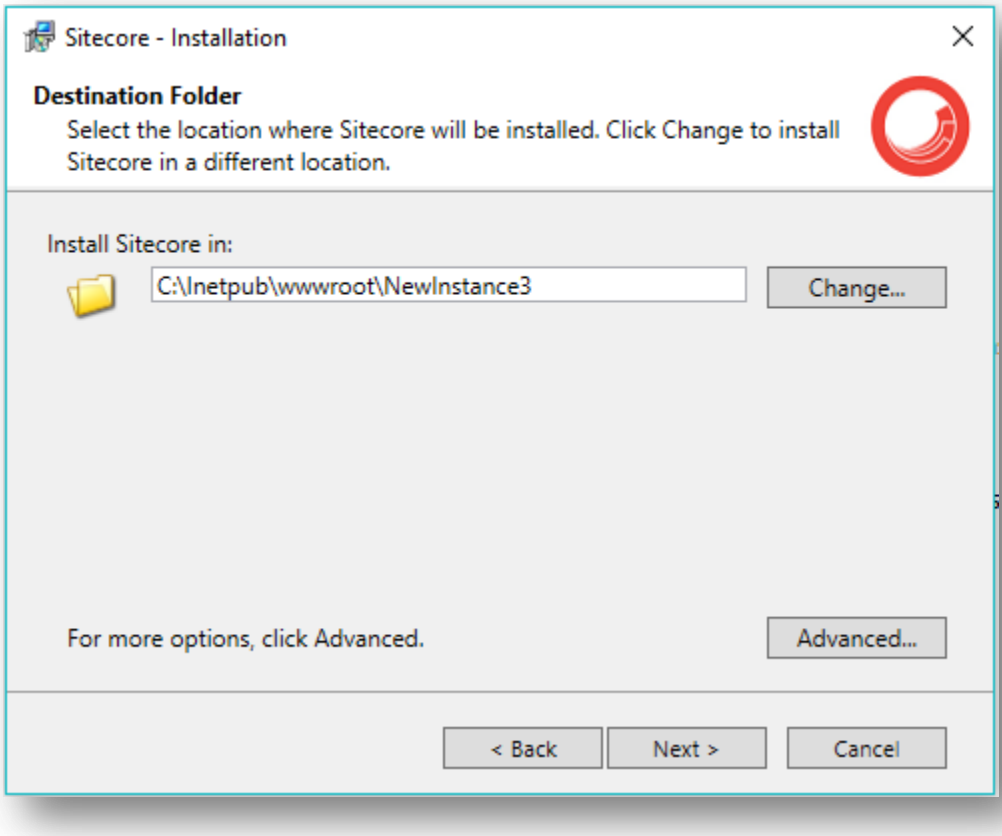


Figura 23. Selección de ubicación donde se instalará la instancia Sitecore. Fuente: Autoría Propia.

- Debemos indicar el nombre de nuestro sitio web, con este nombre vamos a acceder desde el navegador (Figura 24).

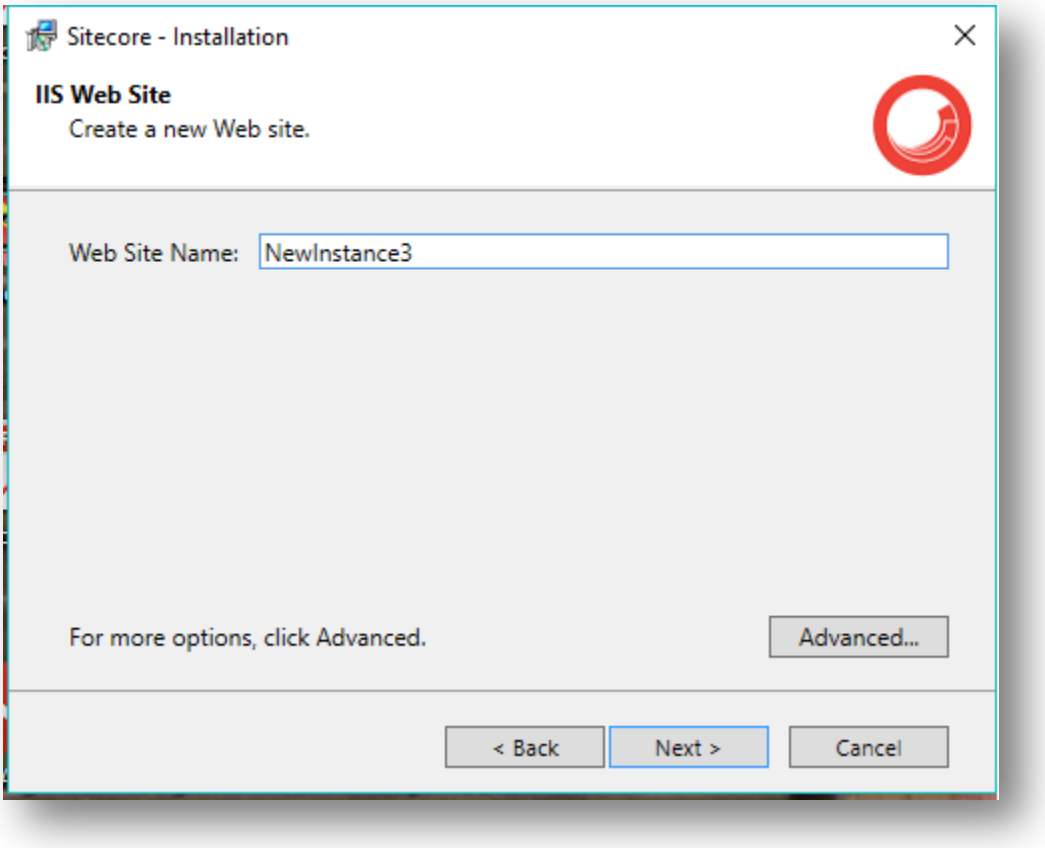


Figura 24 – Selección de nombre del sitio web al que se accederá a la instancia Sitecore. Fuente: Autoría Propia

- En el último paso Sitecore nos muestra el resumen de la instancia.

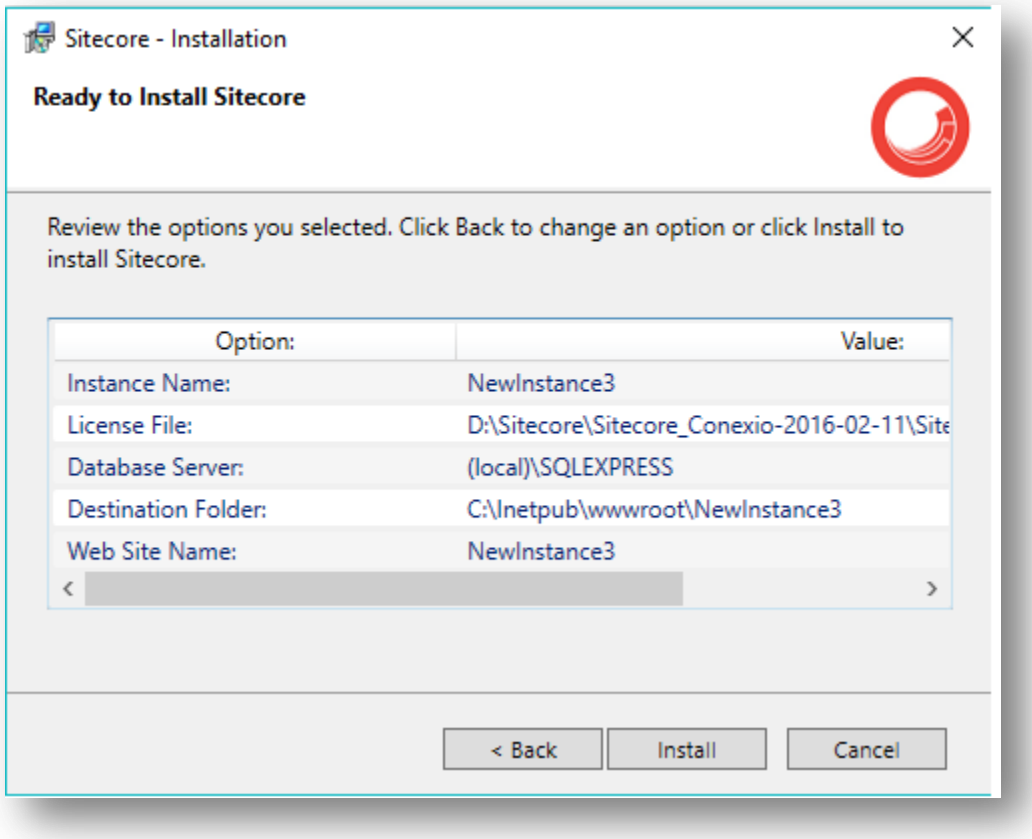


Figura 25 – Resumen de los datos de la instancia Sitecore a instalar. Fuente: Autoría Propia

- Se debe agregar la siguiente línea:
  127.0.0.1 NombreInstancia
  al archivo de configuración host de Windows, generalmente está ubicado en *C:\Windows\System32\drivers\etc*.

- Si todos los pasos fueron realizados de forma correcta ahora debemos ir a un navegador e ingresar la URL http://nombreInstancia con el mismo nombre que indicamos en la instalación. Debería mostrarnos lo que vemos en la Figura 26.

Figura 26 – Página inicial al acceder a sitio web de la instancia instalada. Fuente: Autoría Propia

5.2 Configuración de Visual Studio

Se enumeran a continuación los pasos a seguir para configurar un proyecto Sitecore en Visual Studio:

- Crear proyecto web MVC vacío.

- Sustituir los archivos *Web.config*, *Views/Web.config* y *Global.asax* (Figura 27) creados por defecto en el proyecto por los archivos creados en el proyecto Sitecore, normalmente estos se encuentran en *C:\inetpub\wwwroot\InstanciaSitecore\Website* donde *InstanciaSitecore* es el nombre de la instancia instalada.

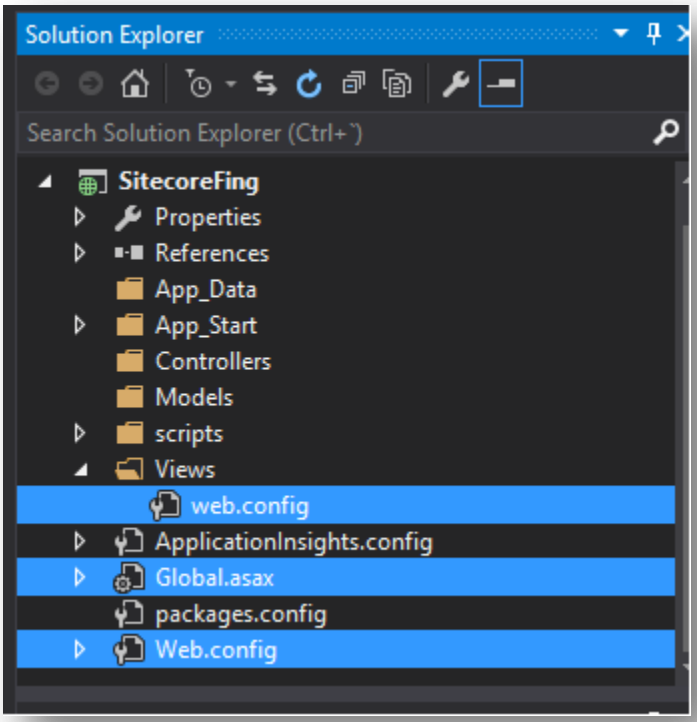


Figura 27 – Archivos a sustituir por los contenidos en la instancia Sitecore. Fuente: Autoría Propia

- Crear carpeta dentro del proyecto y colocar archivos Sitecore.Kernel.dll y Sitecore.MVC.dll que se encuentran en *C:\inetpub\wwwroot\InstanciaSitecore\Website\bin.*
- Agregar en referencias los archivos de la carpeta antes mencionada.
- Instalar *Sitecore Rocks* para su uso desde Visual Studio.
- Conectarse a través de *Sitecore Rocks*.

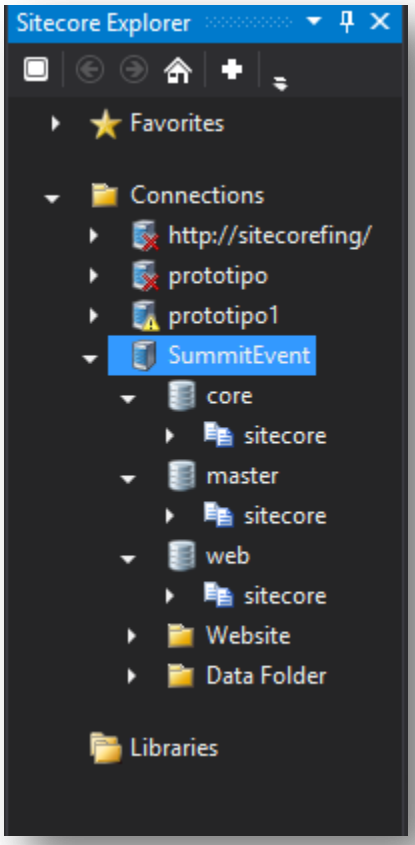


Figura 28 – Captura de imagen de conexión a través de Sitecore Rocks en Visual Studio. Fuente: Autoría Propia

- Configurar publicación del proyecto, a través de *filesystem*, eligiendo la carpeta *Website* de nuestra instancia de Sitecore instalada. Se explica en los siguientes dos pasos:

```
TIP:
Siempre es conveniente ejecutar Visual Studio como administrador
para evitar problemas de acceso.
```

- Se debe seleccionar el tipo *filesystem* como muestra la figura 29.

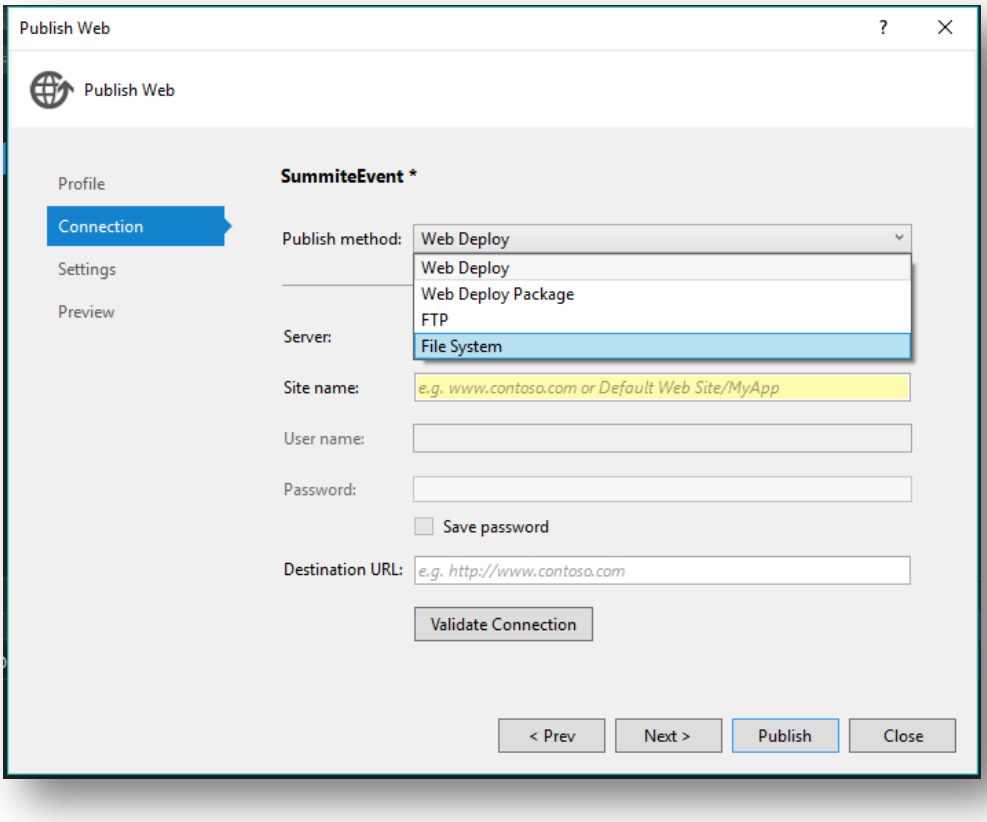


Figura 29 – Selección de método de publicación de proyecto web. Fuente: Autoría Propia

- Se debe indicar la carpeta donde quedó alojada la instancia Sitecore, generalmente en *C:\inetpub\wwwroot\InstanciaSitecore\Website.*

5.2.1 Estructura

Ahora que tenemos pronta nuestra instancia Sitecore, vamos a generar una estructura de carpetas para poder organizar mejor el proyecto. Para eso vamos a entrar a la instancia mediante un navegador y haremos lo siguiente:

- Crearemos una carpeta bajo *Layout/Layouts* con el nombre de nuestro proyecto, en nuestro caso el proyecto se llama *SummitEvent*. Como vemos en la Figura 30, nos paramos en la carpeta *Layouts* y hacemos clic sobre el botón *Layout Folder* para crear nuestra carpeta.

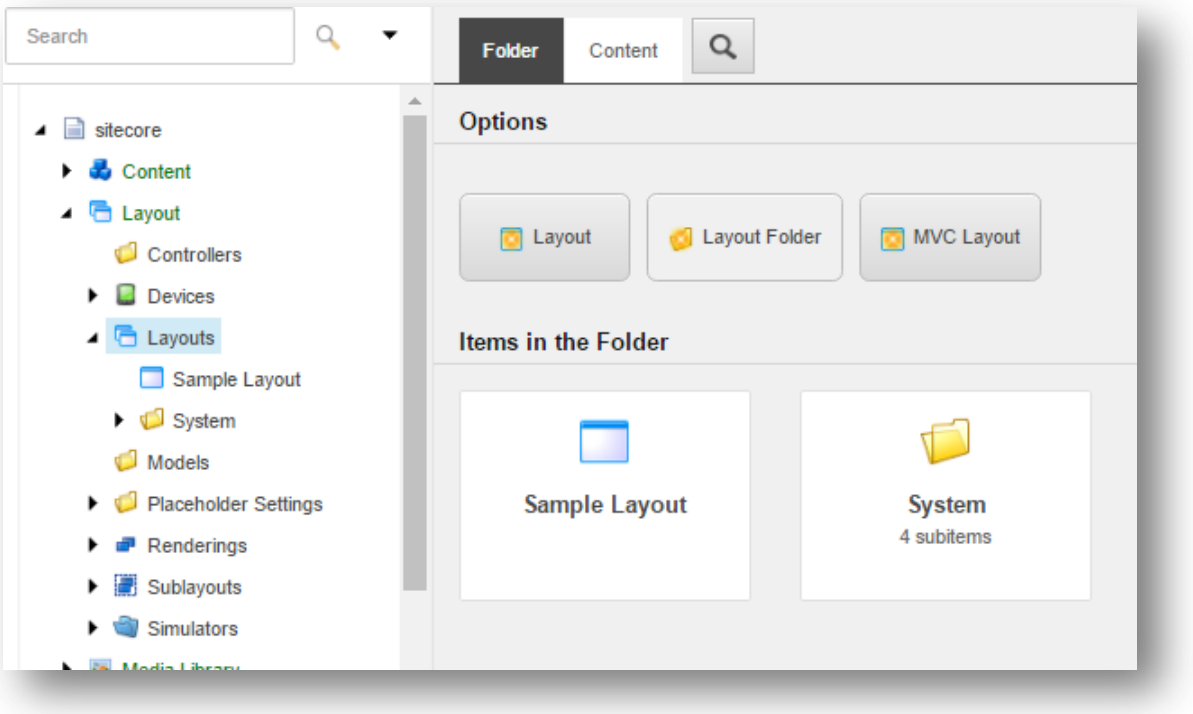


Figura 30 – Captura de imagen de carpeta *Layouts* en instancia Sitecore. Fuente: Autoría Propia

- Vamos a realizar el mismo procedimiento pero en la carpeta *Layout*/*Renderings* y en *Templates*, creando también una carpeta con el nombre del proyecto.

  En estas carpetas vamos a guardar los ítems específicos del proyecto que vayamos creando.

5.3 Instalación y configuración de Unicorn

Unicorn es una herramienta para Sitecore para facilita un ambiente colaborativo entre un equipo de desarrollo, esto en Sitecore no es trivial ya que todos el contenido es manejado por bases de datos relacionales. Esto se convierte en un problema cuando los desarrolladores tienen sus propias instancias locales - los paquetes son propensos a errores y tienden a ser olvidados en el camino a la producción. *Unicorn* resuelve este problema mediante la escritura de copias en serie de ítems de Sitecore en el disco junto con él.

Básicamente mediante una configuración se puede establecer que ítems de la instancia de Sitecore exportar en archivos .yml. Estos archivos luego pueden ser llevados a otra instancia y serán persistidos en la base de datos de la misma.

5.3.1 Instalación

- Para instalar *Unicorn* primero debemos entrar a nuestro proyecto en Visual Studio y abrir la consola de instalación de paquetes NuGet, como vemos en la Figura 31:

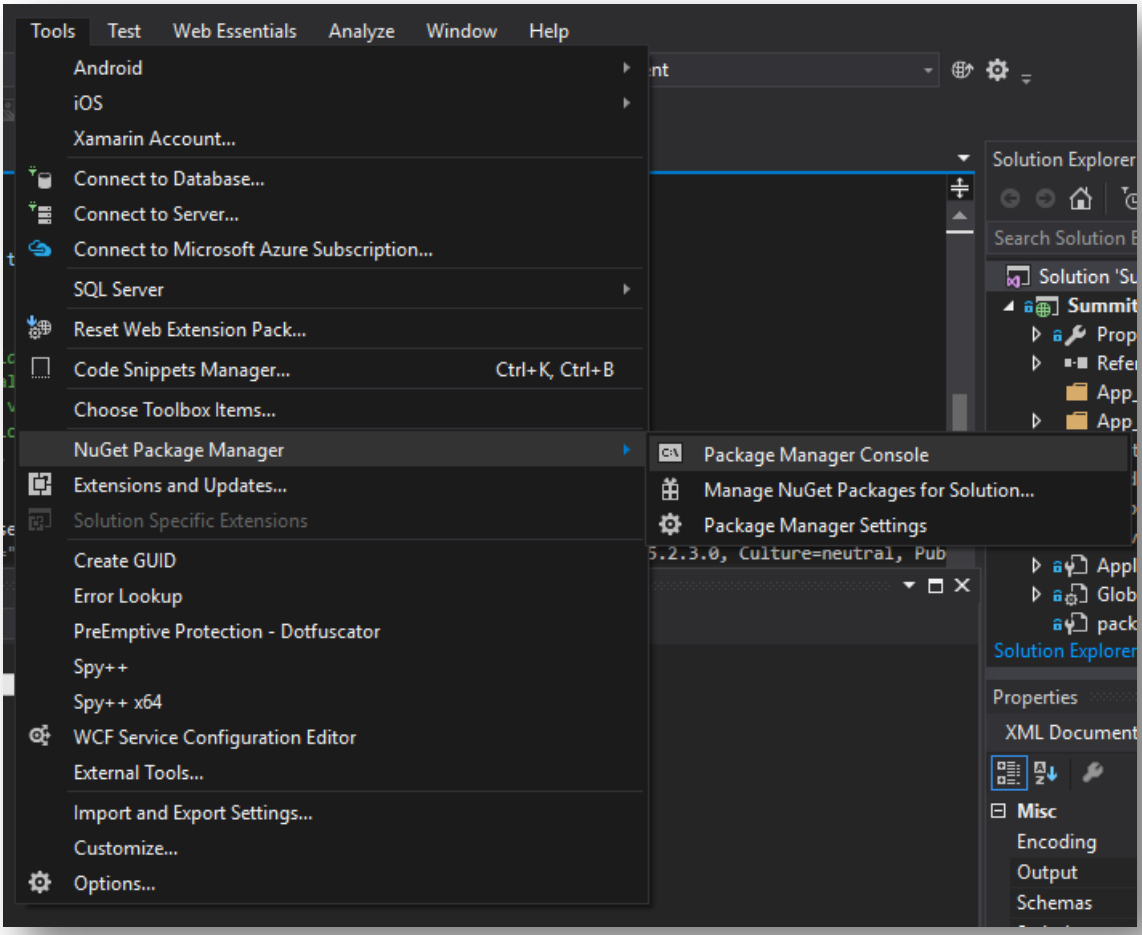


Figura 31 – Acceso a consola de administración de paquetes. Fuente: Autoría Propia

- En la consola ejecutamos *Install-Package Unicorn* para instalar el paquete en nuestro proyecto.

- Si la instalación fue correcta podremos ver en nuestro proyecto, en la carpeta */App_Config/Include/Unicorn* todos los archivos de configuración de la herramienta, como vemos en la Figura 32.

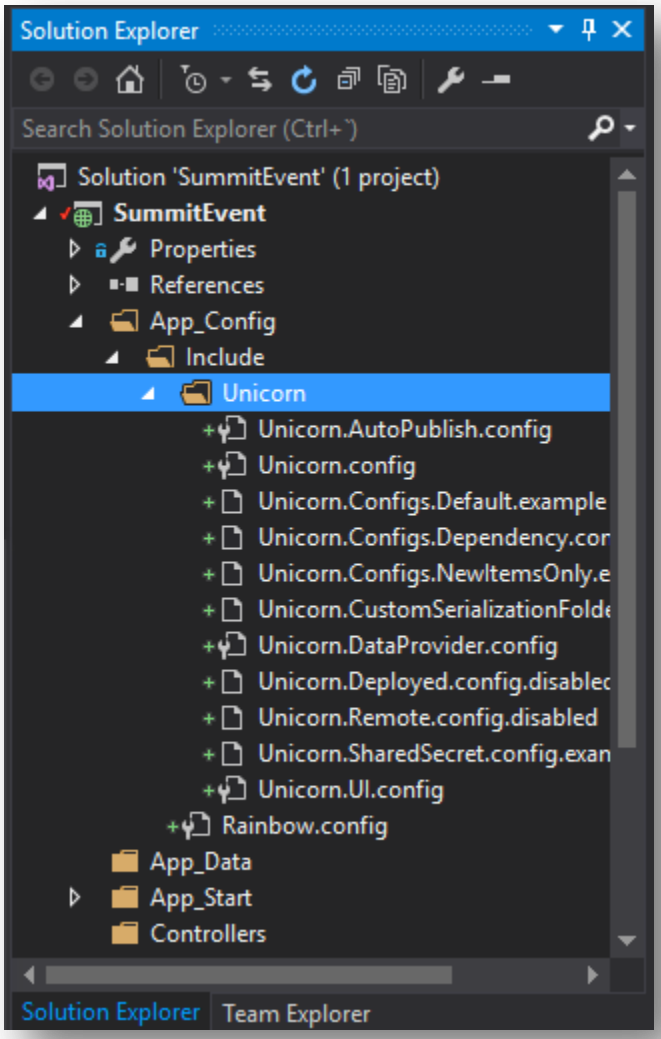


Figura 32 – Archivos de configuración de Unicorn. Fuente: Autoría Propia.

- Por último publicar las modificaciones del proyecto en la instancia de Sitecore, para chequear que haya quedado correctamente instalada en nuestra instancia basta con poner *http://nombreInstansia/unicorn.aspx* en el navegador. Deberíamos ver el siguiente mensaje de la Figura 33:

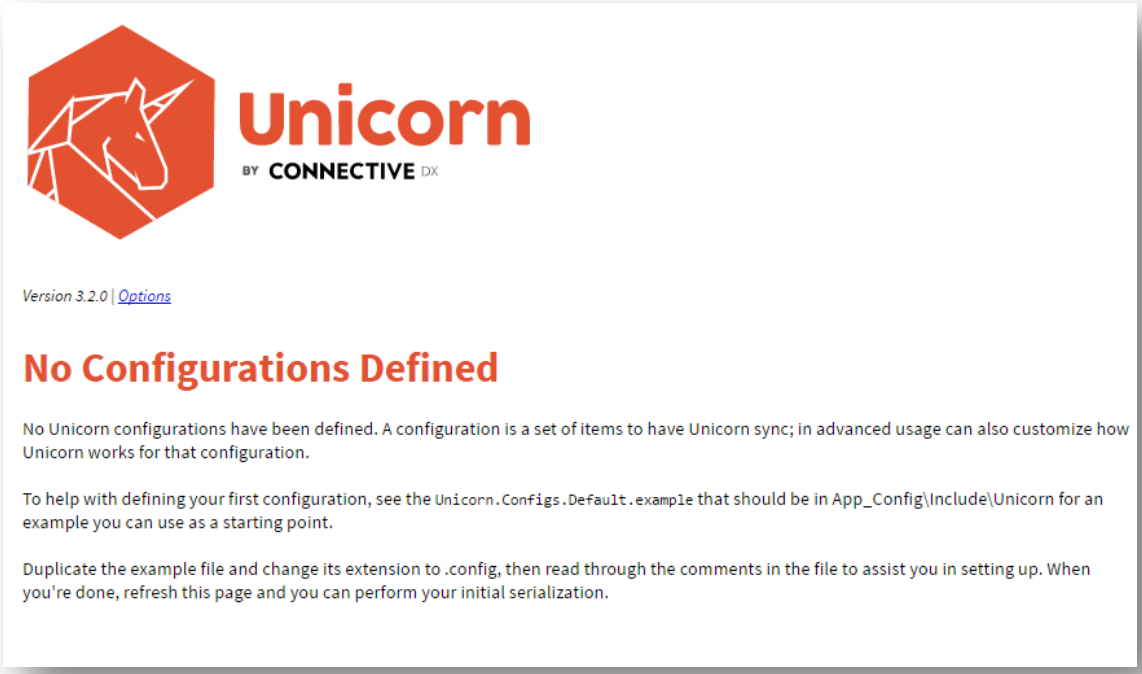


Figura 33 – Pantalla que se muestra si instalación de *Unicorn* fue exitosa. Fuente: Autoría Propia

5.3.2 Configuración

Para poder comenzar a trabajar con *Unicorn* debemos crear un archivo de configuración para indicarle que carpetas debe exportar o sincronizar dentro de nuestra instancia Sitecore.

- Para eso primero debemos entrar a nuestro proyecto e ir a la carpeta */App_Config/Include/Unicorn*, vamos a encontrar un archivo llamado *Unicorn.Configs.Default.Example*, el cual vamos a renombrar cambiando la extensión a Unicorn.Configs.Default.config. Como vemos en la Figura 34 es un archivo XML en el cual le indicaremos a *Unicorn* con los *tags* <*include*> de que base de datos y que carpeta debe tener en cuenta para recolectar y sincronizar ítems.

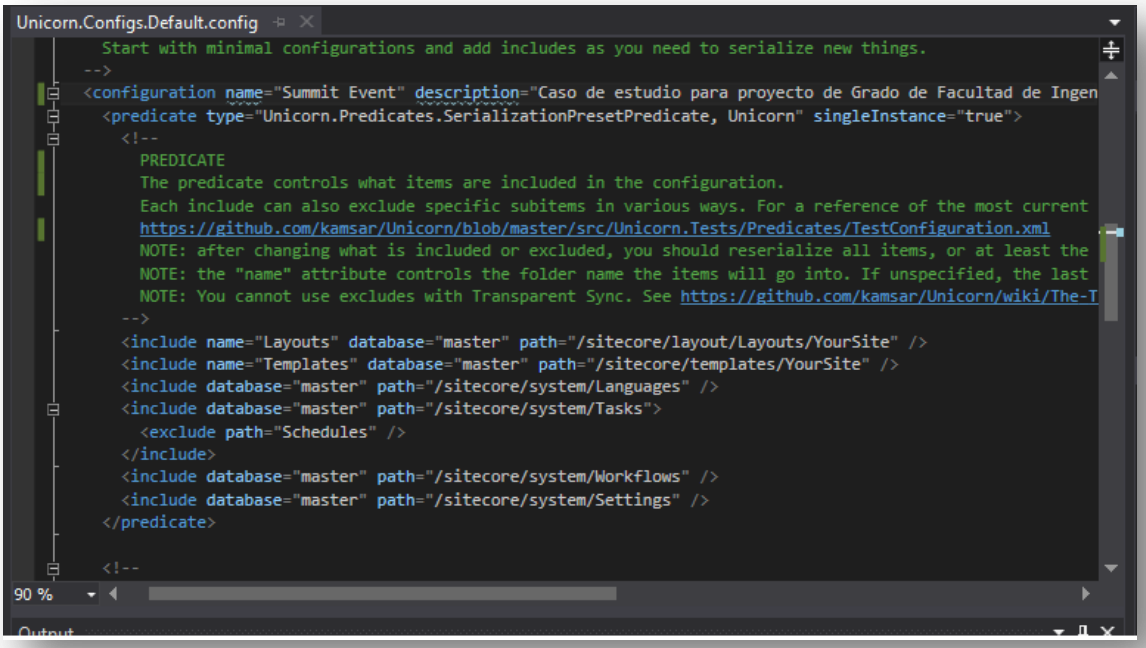


Figura 34 – Configuración de parámetros en Unicorn. Fuente: Autoría Propia

- Vamos a modificar los *include* por defecto para que *Unicorn* se base en las carpetas creadas con el nombre de nuestro proyecto. En nuestro caso quedaría como en la Figura 35.

- También vamos a agregar dos *include* para poder sincronizar los *renderings* y la pagina Home de nuestro proyecto.

```
    Start with minimal configurations and add includes as you need to serialize new things.
-->
<configuration name="Summit Event" description="Caso de estudio para proyecto de Grado de Facultad de
  <predicate type="Unicorn.Predicates.SerializationPresetPredicate, Unicorn" singleInstance="true">
    <!--
      PREDICATE
      The predicate controls what items are included in the configuration.
      Each include can also exclude specific subitems in various ways. For a reference of the most cu
      https://github.com/kamsar/Unicorn/blob/master/src/Unicorn.Tests/Predicates/TestConfiguration.xm
      NOTE: after changing what is included or excluded, you should reserialize all items, or at leas
      NOTE: the "name" attribute controls the folder name the items will go into. If unspecified, the
      NOTE: You cannot use excludes with Transparent Sync. See https://github.com/kamsar/Unicorn/wiki/
    -->
    <include name="Layouts" database="master" path="/sitecore/layout/Layouts/SummitEvent" />
    <include name="Templates" database="master" path="/sitecore/templates/SummitEvent" />
    <!--
    Agregamos las carpetas para los rendering y los Items de Home
    -->
    <include name="Renderings" database="master" path="/sitecore/layout/Renderings/SummitEvent" />
    <include name="Home" database="master" path="/sitecore/Content/Home" />
    <!--
       Agregamos las carpetas para los rendering y los Items de Home
    -->
    <include database="master" path="/sitecore/system/Languages" />
```

Figura 35 – Configuración de *Unicorn* realizada. Fuente: Autoría Propia

- Publicamos los cambios a la instancia local.

- Ahora cuando volvamos a entrar en el navegador de la consola de *Unicorn* deberían aparecen nuestras carpetas seleccionadas para importar y exportar ítems como en la Figura 36.

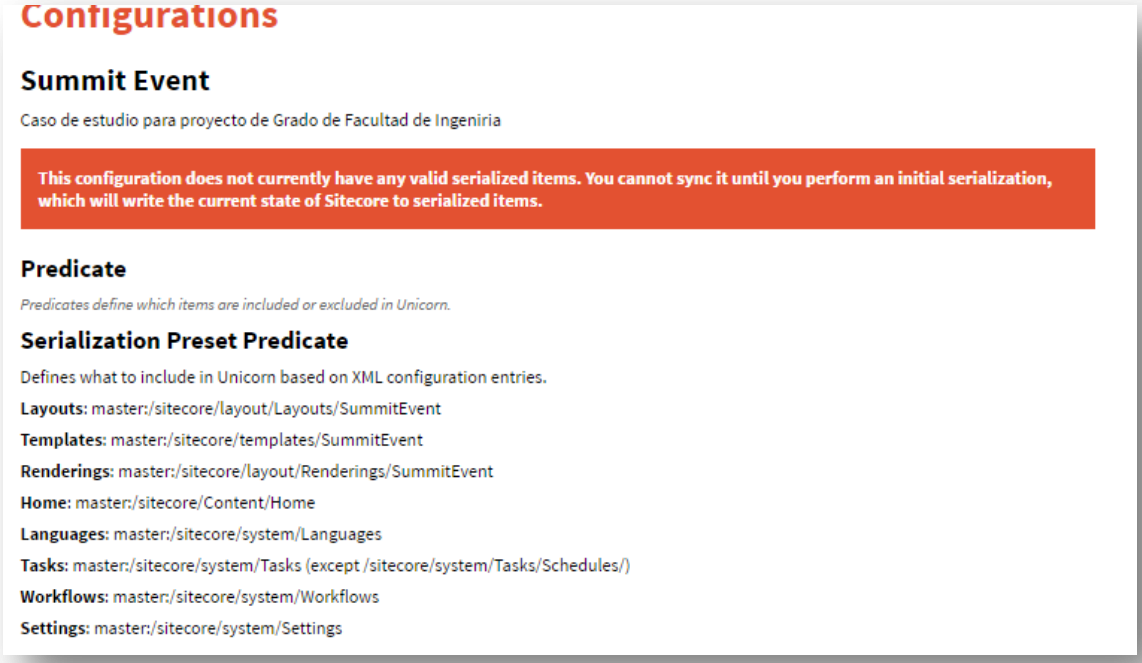


Figura 36 – Vista de carpetas de importación y exportación de ítems. Fuente: Autoría Propia

Por ultimo debemos clicar en la opción *Perform Initial Serialization of Summit Event* para comenzar a utilizar *Unicorn* en la instancia.

5.3.3 Configuración de la Instancia

Lo que hace *Unicorn* es como ya hemos descripto, guardar cada ítem y paquete que se encuentran en las carpetas del archivo de configuración, un archivo .yml para que estos puedan ser exportados a otra instancia. Por defecto estos archivos son creados bajo una carpeta llamada *Unicorn*, en la carpeta de datos de la instancia.

- Para mayor comodidad vamos a cambiar la ruta de la carpeta de datos, por defecto en nuestra instancia es *C:\inetpub\wwwroot\SitecoreFing\Data*, vamos a modificarla para la carpeta *Website*, de esta forma vamos a poder sincronizarla más fácilmente con nuestro proyecto de *Visual Studio*.

- En nuestro proyecto de Visual Studio, vamos a crear una carpeta llamada Sitecore bajo */App_Config/Include* como podemos ver en la figura 37.

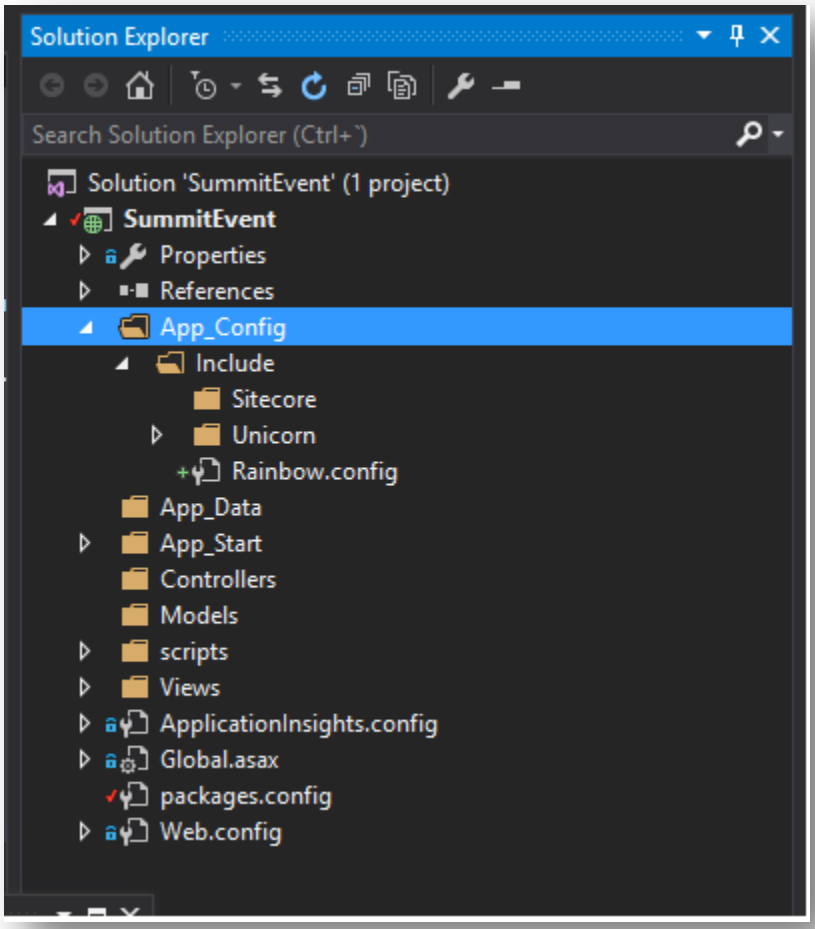


Figura 37 – Configuración de Unicorn en Visual Studio. Fuente: Autoría Propia

- Vamos a crear un archivo XML con el nombre *DataFolder.config* con el siguiente contenido:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
      <sitecore>
            <sc.variable name="dataFolder">
                  <patch:attribute
            name="value">/App_Data/data</patch:attribute>
            </sc.variable>
      </sitecore>
</configuration>
```

Figura 38 – Archivo DataFolder.config. Fuente: Autoría Propia

- Vamos a publicar esta nueva carpeta con su archivo de configuración a nuestra instancia de Sitecore.

- Al cambiar la carpeta de datos de nuestra instancia Sitecore necesitamos mover manualmente todo el contenido de la carpeta de datos anterior a la nueva. Para eso vamos a copiar el contenido desde *C:\inetpub\wwwroot\SummitEvent\Data* a la nueva ruta *C:\inetpub\wwwroot\SummitEvent\Website\App_Data\Data* de esta forma no generaremos ninguna inconsistencia cuando accedamos a nuestra instancia local.

- Por último vamos a incluir en nuestro proyecto la carpeta de *Unicorn*, para esto debemos crear una carpeta *Data* bajo la carpeta *App_Data* del proyecto, y dentro de *Data* creamos la carpeta *Unicorn*. Luego hacemos clic derecho en la carpeta y elegiremos la opción de reemplazar con la carpeta del servidor.

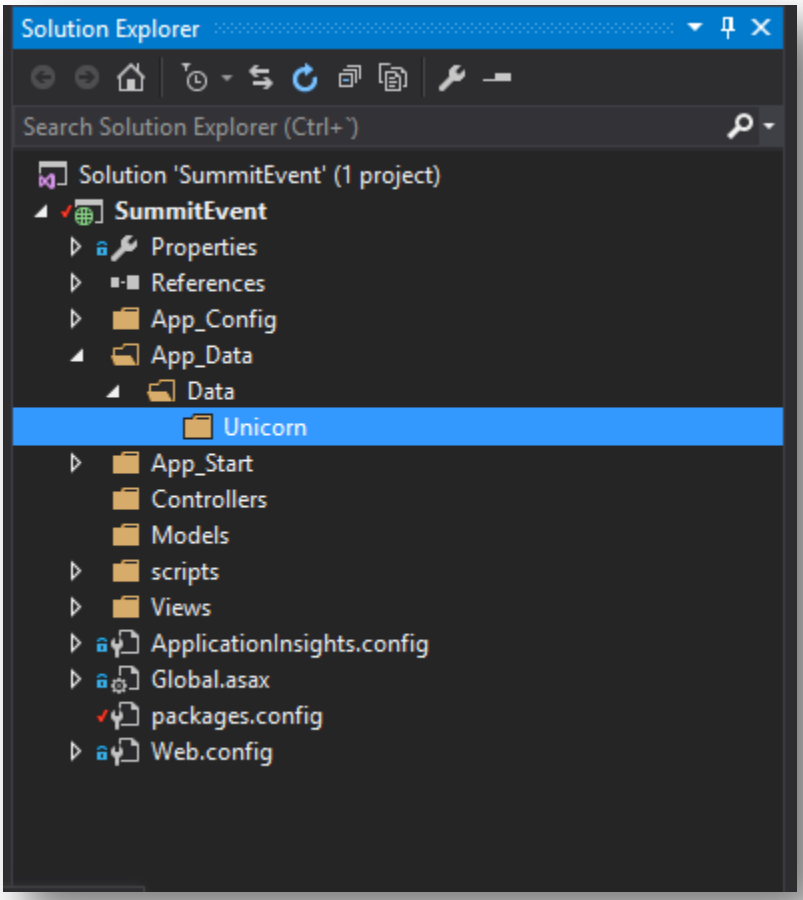


Figura 39. Inclusión de carpeta Unicorn en proyecto. Fuente: Autoría Propia

- Ahora debemos agregar las carpetas que nos trajimos del servidor y se encuentran dentro de *Unicorn*, para eso vamos a hacer clic en la opción de mostrar todo los archivos (esto nos muestra todos los archivos incluso los que no pertenecen a nuestro proyecto) como se ve en la Figura 40.

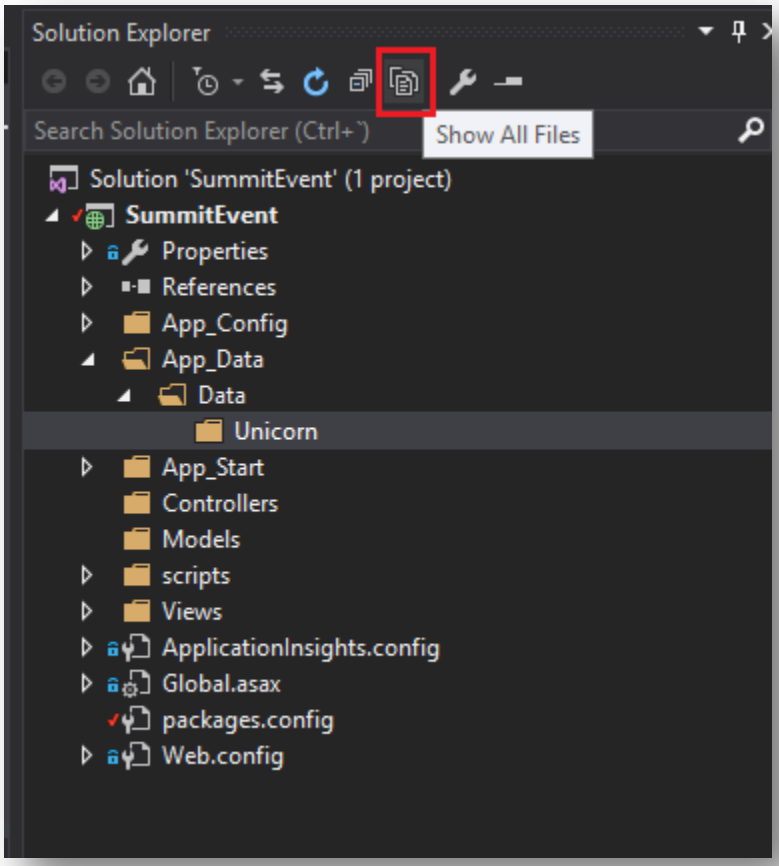


Figura 40 – Botón para mostrar todos los archivos de un directorio en Visual Studio. Fuente: Autoría Propia

- Vamos a poder ver las carpetas no incluidas en el proyecto, las vamos a agregar haciendo clic derecho y seleccionando la opción de incluir en el proyecto, como vemos en la Figura 41, todas las carpetas agregadas al proyecto.

TIP:
Es posible que a medida que se agreguen carpetas al proyecto y queramos agregarlas a nuestra configuración de Unicorn para poder compartirlas se deba repetir el proceso anterior.

TIP:
A menos que tengamos nuestra configuración especifica no es necesario incluir toda la configuración por defecto que viene en Sitecore dentro de la Unicorn, es recomendado removerla (ocurre lo mismo para los *Workflows*).

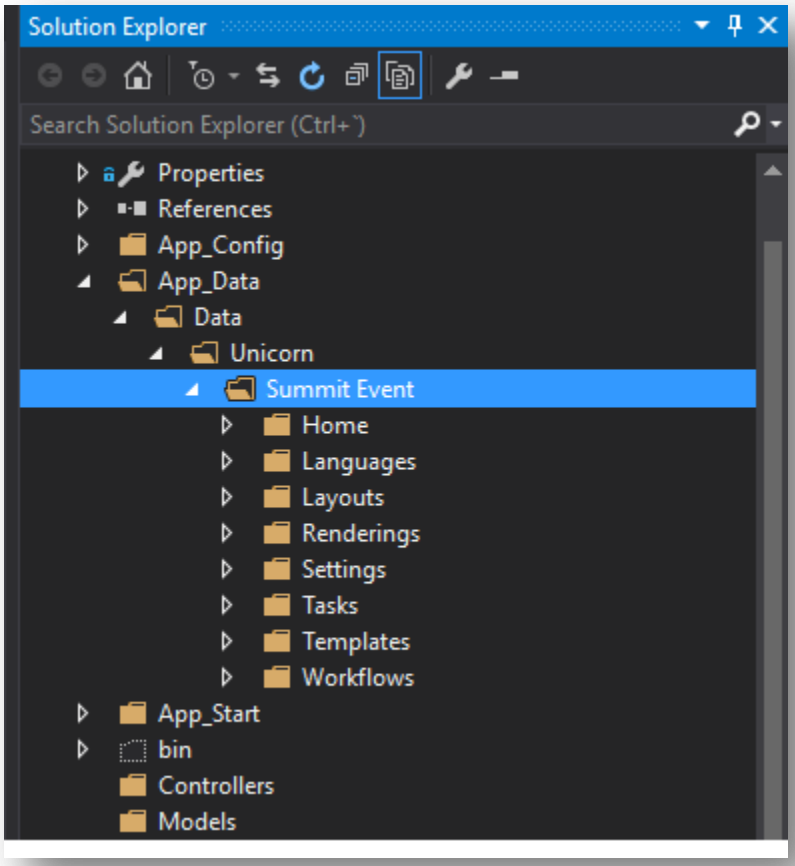


Figura 41 – Se visualizan todas las carpetas agregadas al proyecto para la sincronización de ítems. Fuente: Autoría Propia

- A partir de aquí tenemos un ambiente que puede ser distribuido en cualquier servidor como GitHub, Bitbucket o SVN.

6 Uso de Plug-ins

Para el desarrollo de ciertos componentes a veces es necesario algún tipo de campo que nuestra instancia no posee de forma nativa al instalar. Para satisfacer este tipo de necesidades, se pueden encontrar en la web, tanto en el marketplace de Sitecore [9] como en otros sitios, módulos o paquetes.

Daremos un ejemplo de instalación de un paquete en nuestra instancia de Sitecore, ya que el procedimiento es análogo para cualquier paquete que se quiera instalar.

El paquete en cuestión agrega un campo llamado Color Picker, el cual despliega una paleta de colores y el color seleccionado es automáticamente pasado al código hexadecimal correspondiente.

En primer lugar accedemos al modo de escritorio, luego a *Development Tools* y por último a *Installation Wizard* como se muestra en la imagen siguiente:

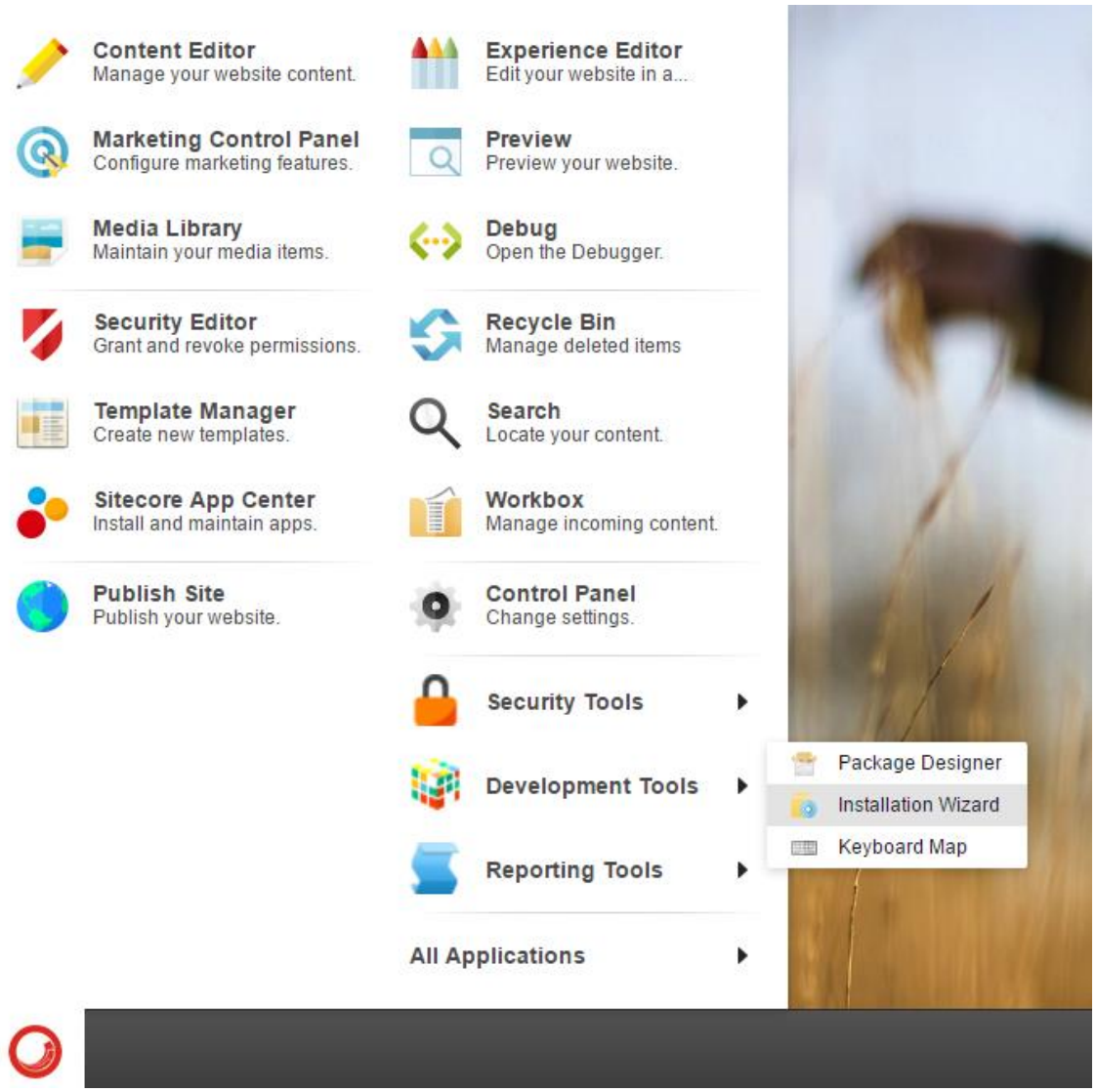


Figura 42 – Acceso al asistente de instalación en la instancia Sitecore. Fuente: Autoría Propia

Luego hay que seleccionar el botón *Upload package* para subir el paquete deseado. A continuación hay que seleccionar el paquete (archivo zip) previamente disponible en nuestro equipo y damos clic en el botón *Update*. Luego de la confirmación, se cierra el cartel y nos muestra lo siguiente:

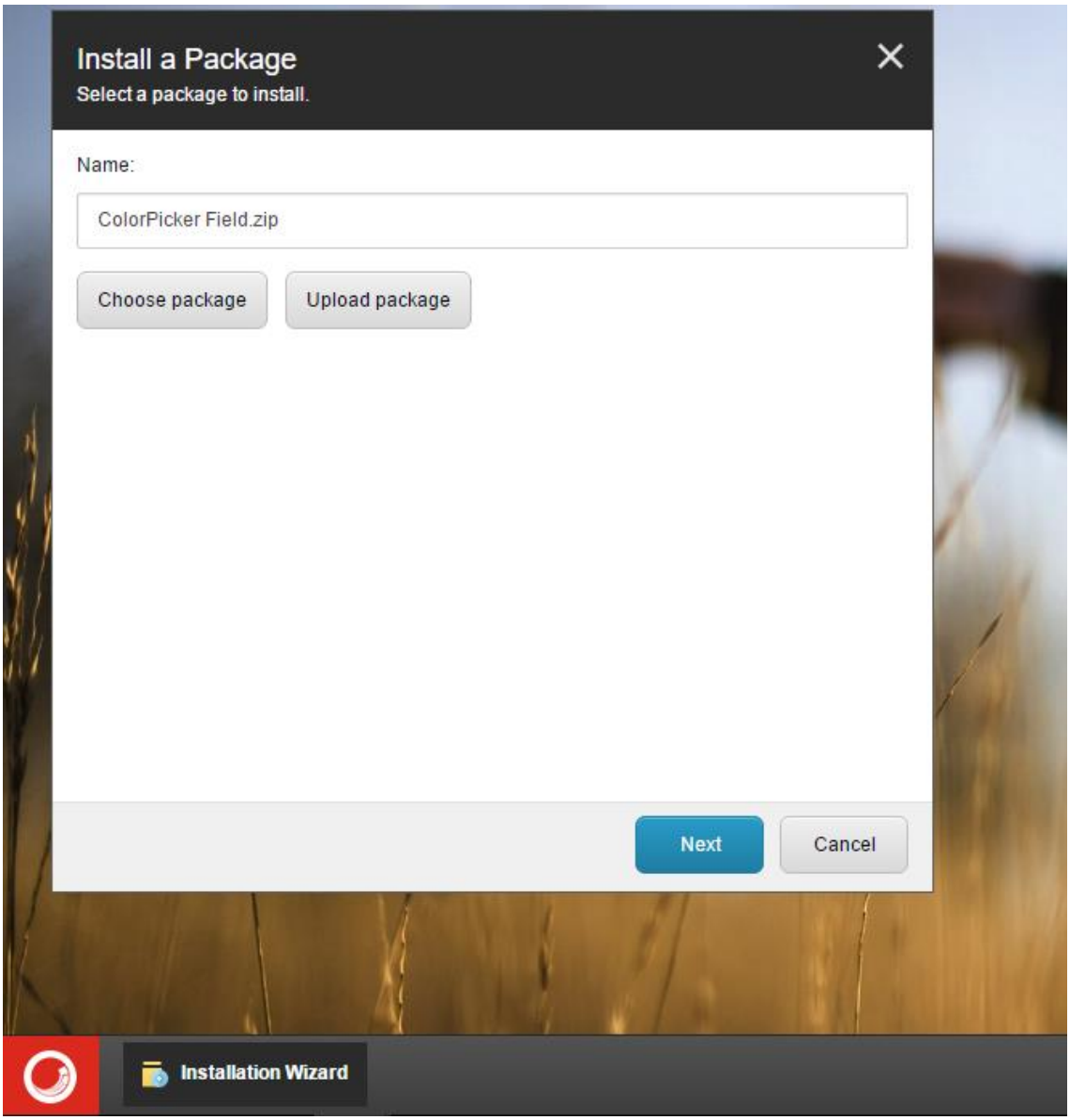


Figura 43 – Selección de paquete a instalar. Fuente: Autoría Propia

Ya con el paquete seleccionado, al dar clic en *Next* se instala el paquete. Puede variar un poco la duración del mismo, dependiendo del tamaño del paquete, entre otros factores.

7 Manual de desarrollo

Vamos a ver ejemplos prácticos para desarrollar partes de un sitio web con Sitecore. Va a estar organizado en secciones con dificultad incremental, cada sección va a tener objetivos y vamos a realizar una guía paso a paso de cómo lograrlos.

7.1 Conceptos clave

Template
Item
Placeholder
Layout
Rendering

7.2 Sección 1

7.2.1 Objetivo

Crear un *template* a partir del cual se creen las páginas de nuestro sitio.

7.2.2 Desarrollo

7.2.2.1 Estructura Base

Lo primero que vamos a hacer es crear lo que Sitecore llama *Layouts*, esto es la capa de vista de nuestro sistema, para esto vamos a abrir nuestro proyecto en *Visual Studio* y conectarnos a nuestra instancia por *Sitecore Rocks*. Vamos a crear una carpeta en nuestro proyecto llamada *Structure* bajo *Views* como vemos en la Figura 44.

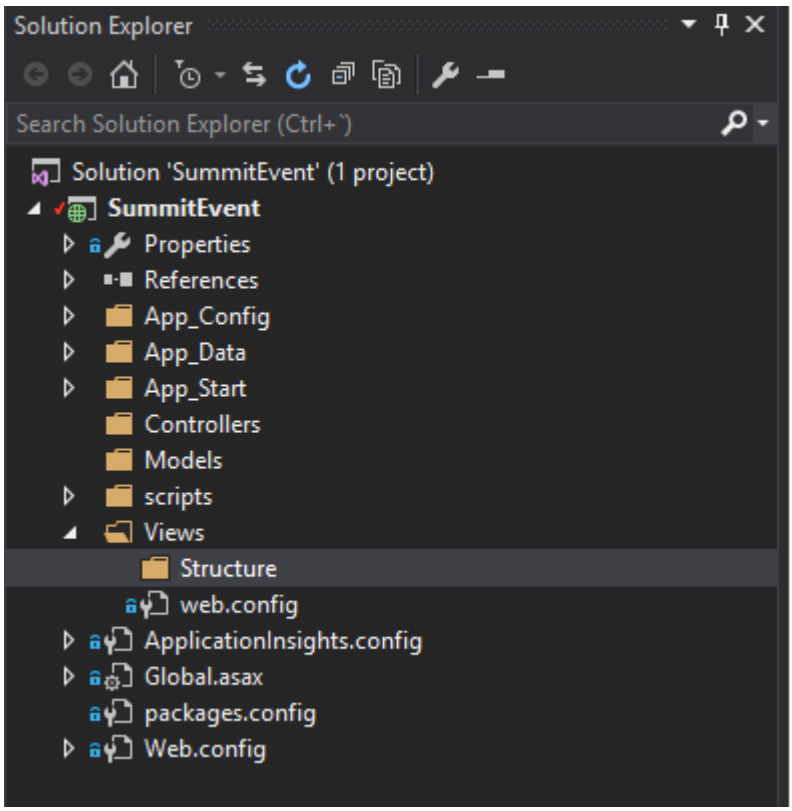


Figura 44 – Carpeta *Structure* creada dentro de carpeta Views en Visual Studio. Fuente: Autoría Propia

Vamos a crear una carpeta *Structure* en nuestra instancia de Sitecore, bajo *master/layout/Layouts/SummitEvent*, esto se puede hacer desde *Visual Studio*, abriendo la vista *Sitecore Explorer* y nos paramos en la ruta antes mencionada, clic derecho y seleccionamos la opción *Add -> Layout Folder*, como vemos en la Figura 45.

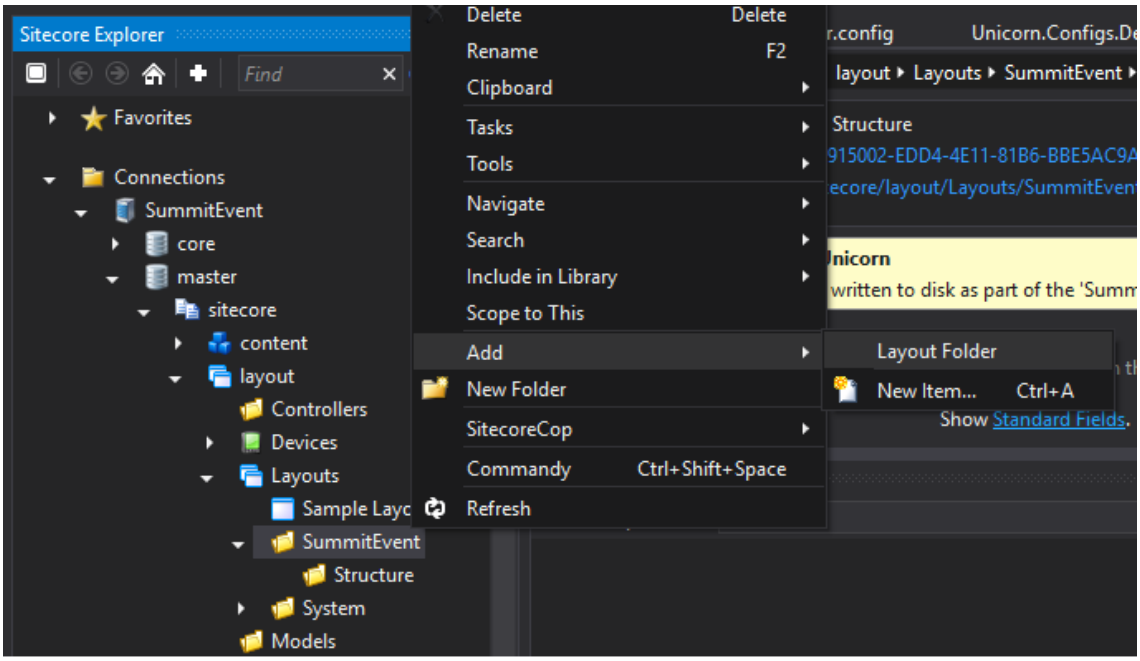


Figura 45 – Creación de carpeta *Structure* en instancia Sitecore a través de *Sitecore Rocks*. Fuente: Autoría Propia

Debemos sincronizar la creación de la carpeta hecha en *Visual Studio* con nuestra instancia Sitecore. Para eso nos paramos sobre la base de datos *master* y clicamos en el botón *Smart Publish* como vemos en la Figura 46. Podemos chequear en un navegador si se sincronizó correctamente con nuestra instancia y se creó la carpeta *Structure*.

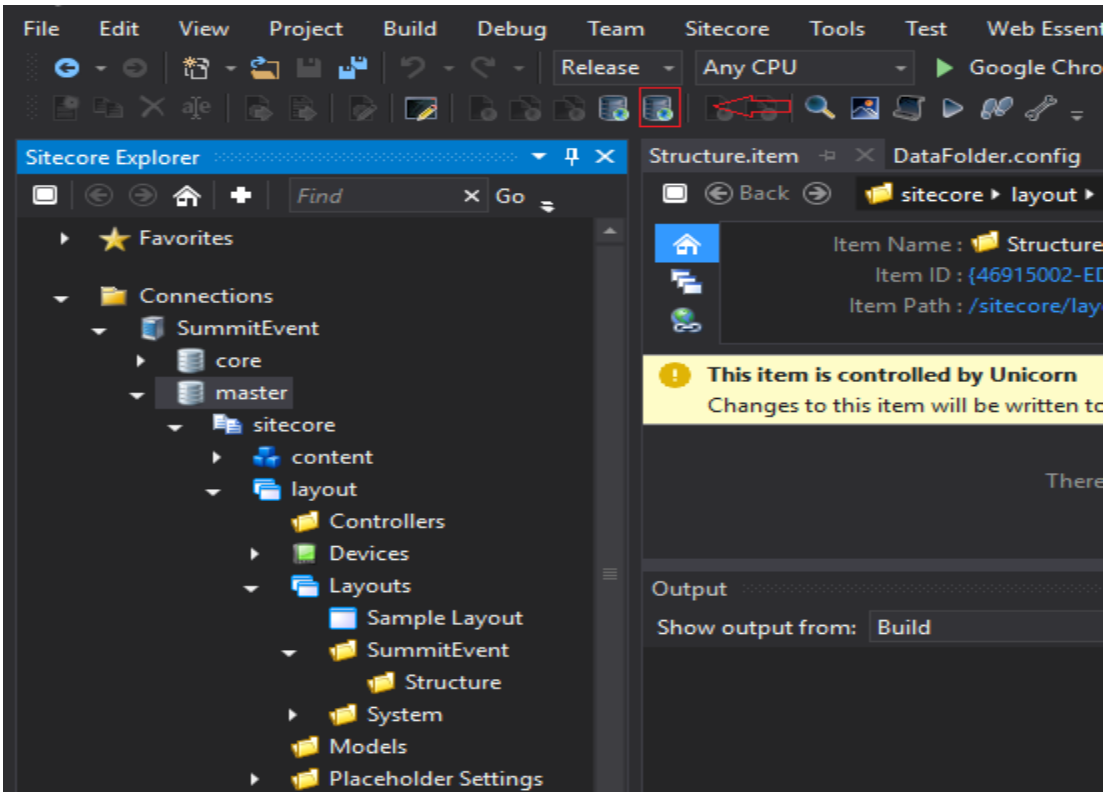


Figura 46 – Botón Smart publish para Sitecore Rocks. Fuente: Autoría Propia

AEM:
Básicamente lo que estamos haciendo es creando lo que sería un *template* de página en AEM, hay que tener cuidado ya que los conceptos de *Template* en AEM y Sitecore son muy distintos, mientras que el uso que le da AEM a los *templates* es para definir páginas y sus propiedades, los *Templates* de Sitecore definen cualquier tipo de objeto de la instancia, desde componentes hasta datos o páginas, todo ítem creado en Sitecore debe tener asociado un *template*, cuando uno define un *Template* en Sitecore define un "tipo" de objeto.

7.2.2.2 Creación de Layouts

Vamos a seguir con la creación de nuestro *layout* para nuestro *template* de página base, el *plug-in* de *Sitecore Rocks* nos permite crear archivos en nuestro proyecto y asociarlos a ítems de Sitecore. Para eso vamos a hacer clic derecho sobre la carpeta recientemente creada *Structure* y seleccionamos la opción *Add -> New Item* como vemos en la Figura 47:

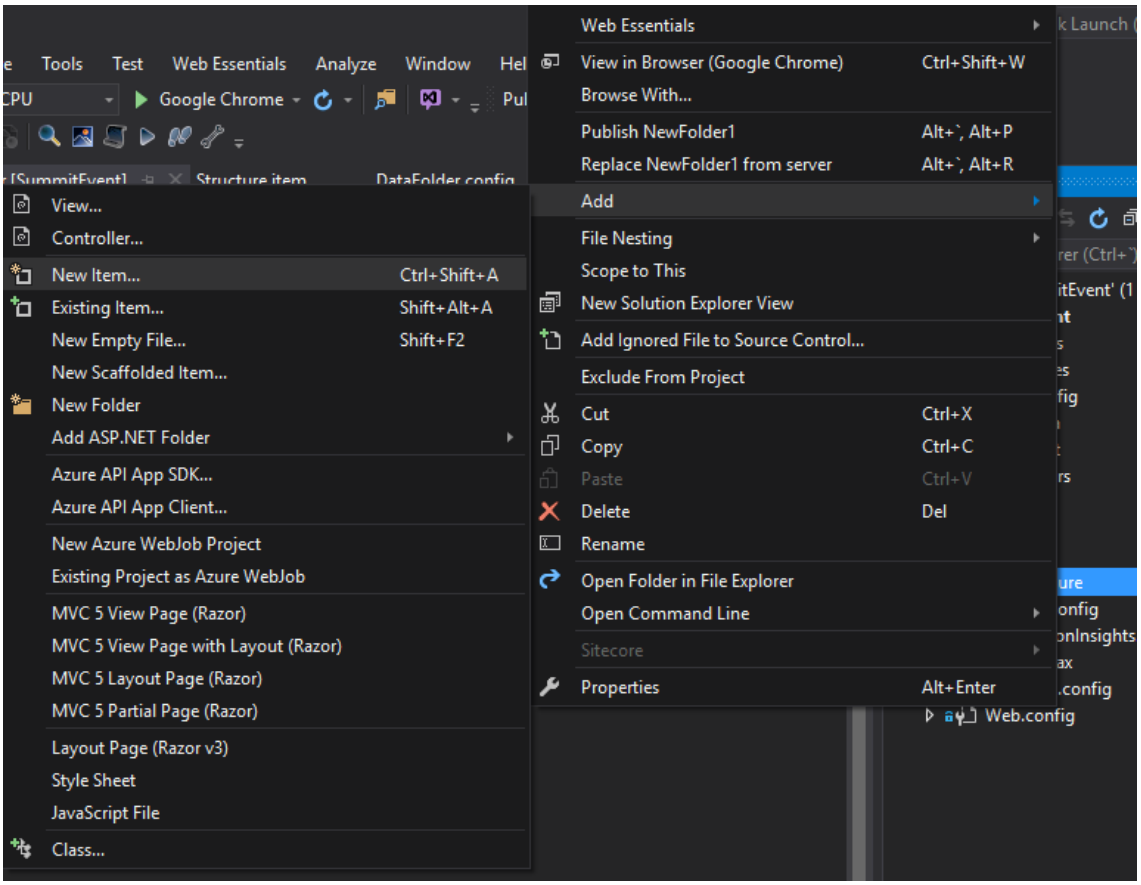


Figura 47 – Agregado de ítem en carpeta del proyecto en Visual Studio. Fuente: Autoría Propia

Nos va a abrir un diálogo, donde seleccionaremos la opción *Sitecore -> MVC* y vamos a crear un *Sitecore View Layout* con el nombre de BasePage.cshtml como vemos en la Figura 48.

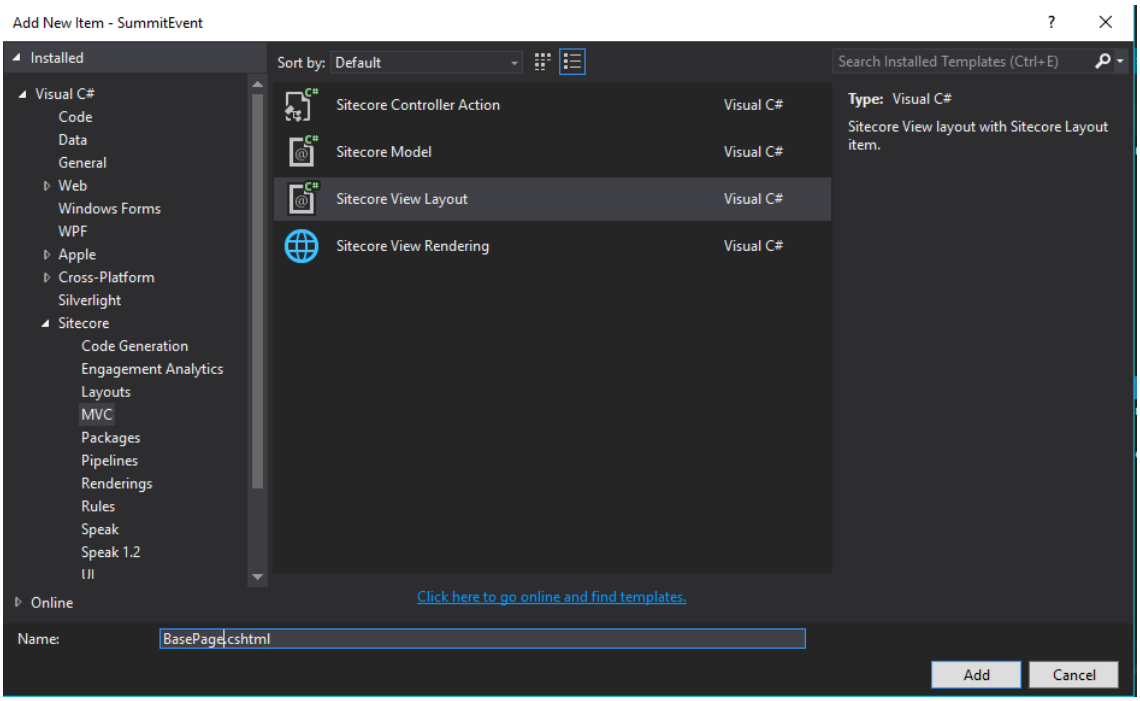


Figura 48 – Selección de tipo de ítem a agregar. Fuente: Autoría Propia

Cuando seleccionamos crear nos va a abrir la opción de ya crear el *layout* correspondiente al archivo recién creado en Sitecore. Aceptamos y elegimos la ubicación homóloga dentro de la carpeta *master/layout/Layouts/SummitEvent/Structure* como vemos en la Figura 49*,* luego realizamos el *Smart Publish* y publicamos también nuestro proyecto*.*

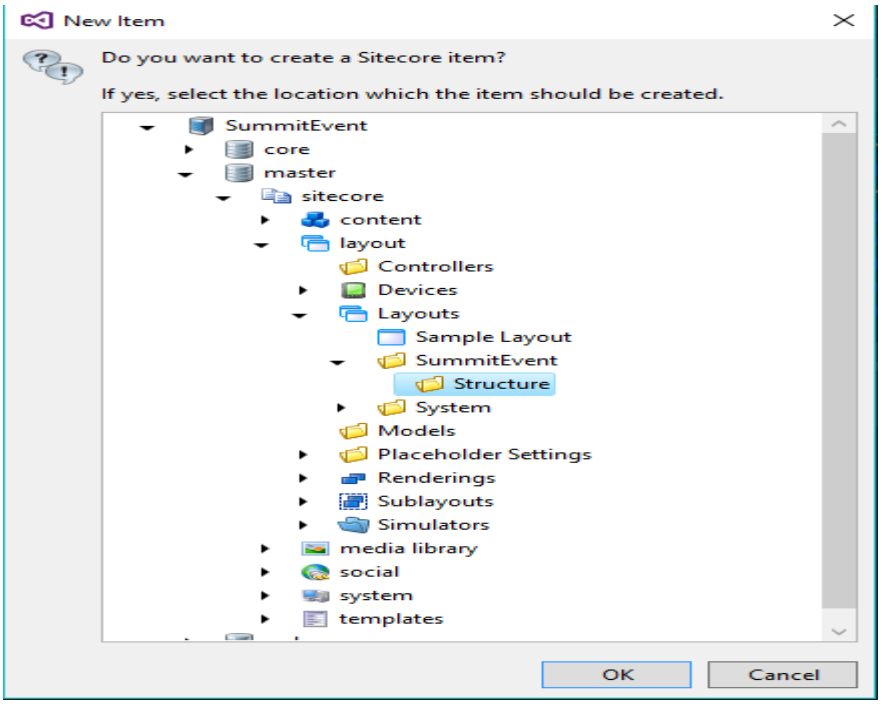


Figura 49 – Cuadro desplegado por visual Studio para crear un ítem en la instancia Sitecore a partir del ítem creado en el proyecto en Visual Studio. Fuente: Autoría Propia

Vamos a sustituir el contenido creado por defecto de nuestro BasePage.cshtml por el siguiente:

```
@using Sitecore.Mvc
@using Sitecore.Mvc.Analytics.Extensions
@using Sitecore.Mvc.Presentation
@model RenderingModel
@{
  Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    @Html.Sitecore().Placeholder("headlibs")
</head>
<body>
    <div>
        @Html.Sitecore().Placeholder("topnav")
    </div>
    <div>
        @Html.Sitecore().Placeholder("main")
    </div>
    <footer>
        @Html.Sitecore().Placeholder("footer")
    </footer>
    @Html.Sitecore().Placeholder("footlibs")

</body>
</html>
```

De esta forma definimos cinco *placeholders* para nuestro *layout*, uno para el menú de navegación, otro para el *footer*, dos para librerías y el *main* donde va a ir nuestro contenido.

> **AEM:**
> Los *placeholders* de Sitecore son similares a los *parsys* de AEM, son contenedores, sirven para que los autores puedan agregar contenido, también se les puede agregar restricciones para limitar que tipo de contenido le permitimos a los autores que agreguen en dicha página.

Lo que queremos es que al final nuestro *template* cargue las librerías, el *footer* y el *topnav*, por lo que vamos a comenzar con las librerías. Vamos a crear dos componentes, uno para las librerías del *footer* y otro para las librerías del *header*.

Para eso vamos a crear en nuestro proyecto una nueva carpeta dentro de *Structure* llamada *Libs* y en Sitecore vamos a crear una carpeta *Structure* dentro de *master/layout/Renderings/SummitEvent* y dentro de *Structure* creamos otra carpeta llamada *Libs*, la jerarquía debería quedar como en la Figura 50.

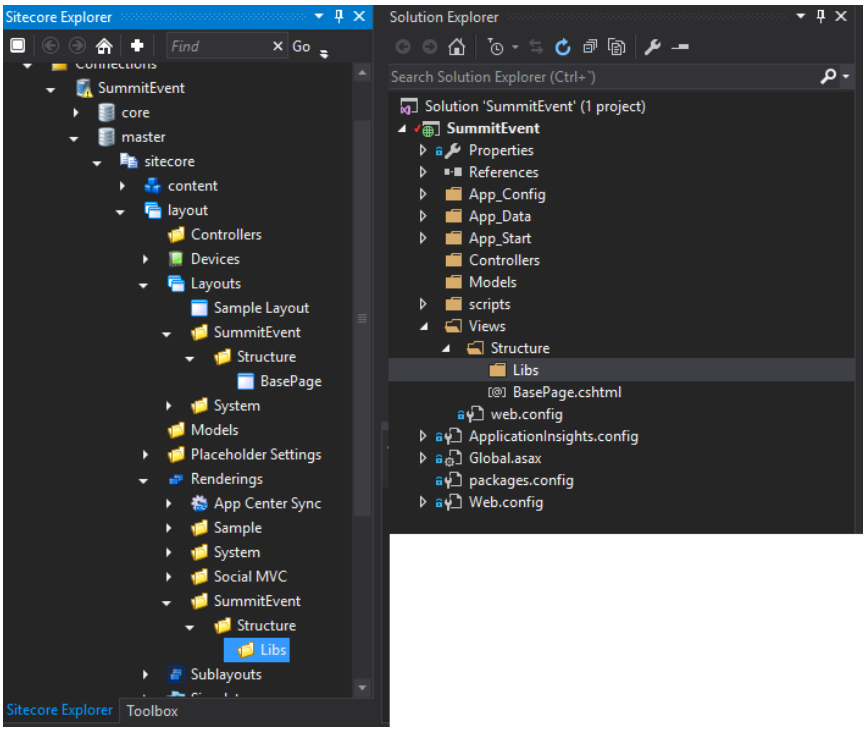


Figura 50 – Jerarquías de archivos en proyecto de Visual Studio y en instancia Sitecore correspondiente. Fuente: Autoría Propia

Vamos a crear dos view *renderings* simples para poder agregarlos a nuestro *BasePage layout*. Para eso nos paramos en la carpeta *Libs* de nuestro proyecto y hacemos clic derecho y elegimos a opción para agregar un *Sitecore View Rendering* con el nombre headlibs.cshtml y creamos el ítems asociado en la

instancia de Sitecore en la misma carpeta *Libs* que creamos anteriormente. Hacemos lo mismo creando el *footlibs*.

> **AEM:**
> Los *renderings* de Sitecore son similares a los componentes de AEM, básicamente se basan en el Patron MVC, se pueden crear *renderings* solo con un archivo cshtml, con un controlador y un cshtml o con un controlador, un modelo y un cshtml. El archivo cshtml sería el equivalente a slightly en AEM, es un html con ciertas funcionalidades extra, más parecido a un jsp. El controlador es similar a la clase java que usan los componentes en AEM.

En la carpeta *Scripts* de nuestro proyecto vamos a descargar y guardar una versión de *Bootstrap* para poder utilizar en nuestro ambiente. Debemos incluir los js y css del mismo. La idea es que en esta carpeta estén los estilos globales que son utilizados en toda nuestra web. Importamos los css en nuestro archivo headlibs.cshtml y nuestros archivos js en footlibs.cshtml como vemos en la Figura 51.

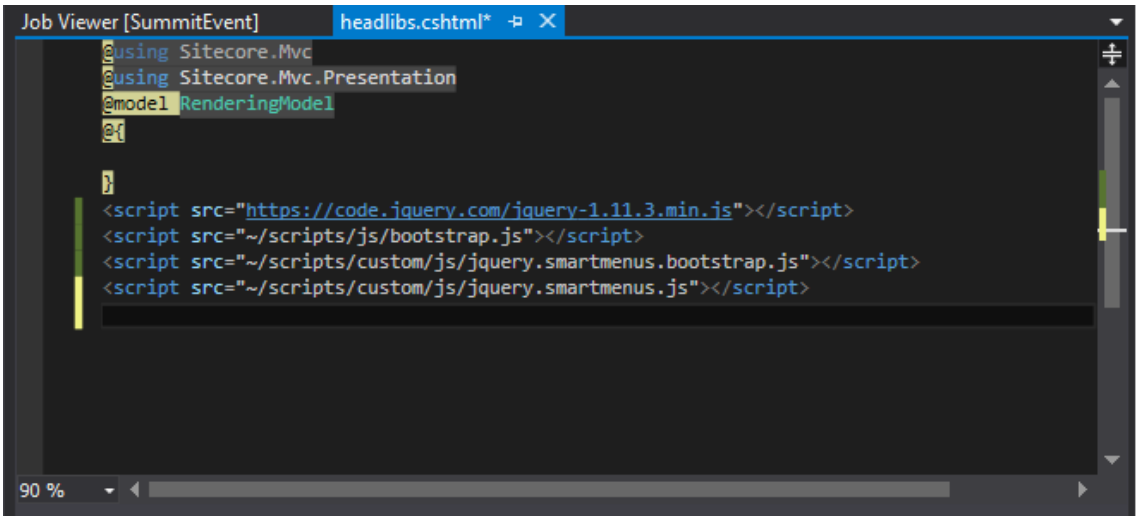


Figura 51 – Importación de archivos JavaScript en archivo headlibs.cshtml. Fuente: Autoría Propia

> TIP:
> *Visual Studio* nos permite agregar las importaciones de una forma fácil, simplemente podemos por ejemplo arrastrar un archivo .js hacia un cshtml o html y automáticamente nos creara el *import* correspondiente a dicho archivo.

Ahora que tenemos creado los dos *renderings* con las librerías los vamos a agregar de forma estática a nuestro *layout* BasePage.cshtml. Para eso vamos a necesitar saber el ID de nuestros componentes generados, para eso debemos hacer doble clic en el ítem *headlibs* en el *Sitecore Explorer*, como vemos en la Figura 52.

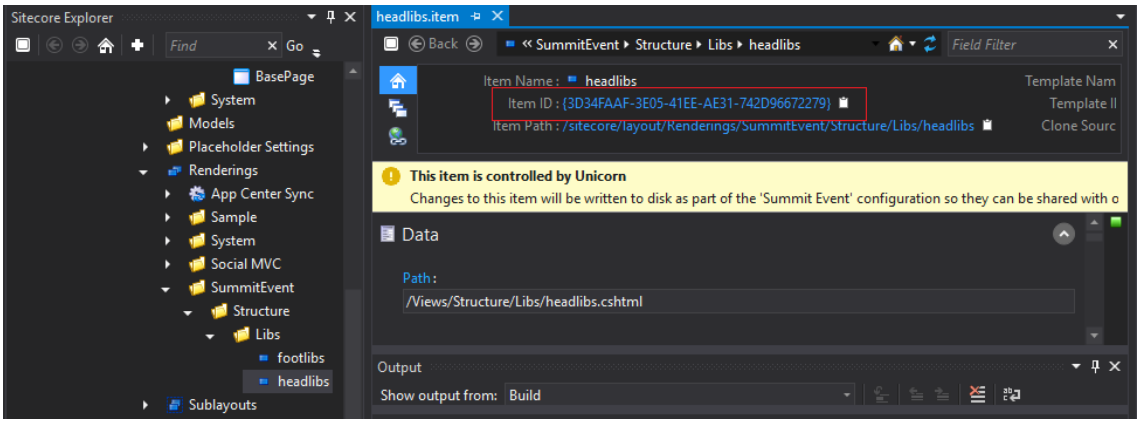


Figura 52 – Identificación de ID de ítem en Sitecore correspondiente al *rendering* headlibs. Fuente: Autoría Propia

Vamos a modificar nuestro *placeholder footlibs* y *headlibs* por una inclusión estática de ambos componentes en BasePage.cshtml quedando el código de la siguiente manera:

```
@using Sitecore.Mvc
@using Sitecore.Mvc.Analytics.Extensions
@using Sitecore.Mvc.Presentation
@model RenderingModel
@{
  Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    @Html.Sitecore().Rendering("{3D34FAAF-3E05-41EE-AE31-742D96672279}", new { Cacheable
= true, Cache_VaryByData = true })
</head>
<body>
    <div>
        @Html.Sitecore().Placeholder("topnav")
    </div>
    <div>
        @Html.Sitecore().Placeholder("main")
    </div>
    <footer>
        @Html.Sitecore().Placeholder("footer")
    </footer>
    @Html.Sitecore().Rendering("{41DEC776-F50A-4EEA-8822-389FD65D3199}", new { Cacheable
= true, Cache_VaryByData = true })
</body>
</html>
```

Donde el *string* entre paréntesis es el ID de los componentes *footlibs* y *headlibs*.

Hasta este momento tenemos dos componentes, *headlibs* que cuenta con las importaciones de archivos css y por otro lado *footlibs* que cuenta con las importaciones de archivos js. Tenemos un *layout* que utiliza ambos de forma estática, vamos a proceder a crear un *template* que utilice nuestro *layout*. Al código del *layout* vamos a agregarle un título para saber que el ítem está utilizando nuestro *layout*.

Vamos a agregar una carpeta *Structure* dentro de *master/templates/ SummitEvent***,** vamos a hacer clic derecho en la carpeta recientemente creada y vamos a crear un *template* con el nombre *BasePage*. A este *template* vamos a crearle los s*tandard values*, para eso le damos clic derecho y elegimos la opción *Create Standard Values*. De esta forma las propiedades que nosotros configuremos en los *standard values* afectarán a todos los ítems creados bajo este *template*.

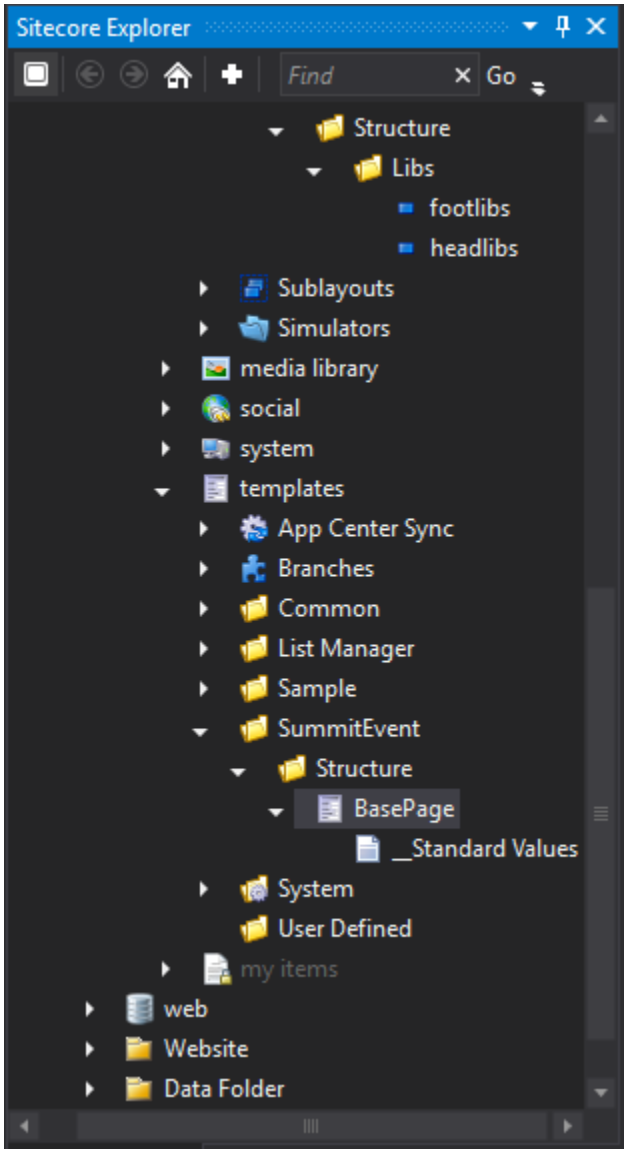


Figura 53 – Configuración de valores estándar para BasePage. Fuente: Autoría Propia

Ahora vamos a asociar a los *standard values* un *layout*, para eso seleccionamos con clic derecho el ítem _Standard Values y elegimos dentro de *Task* la opción *Design Layout*.

En la ventana que nos abre hacemos clic sobre el botón *Browse*, y elegimos nuestro *layout BasePage* como vemos en la Figura *54.*

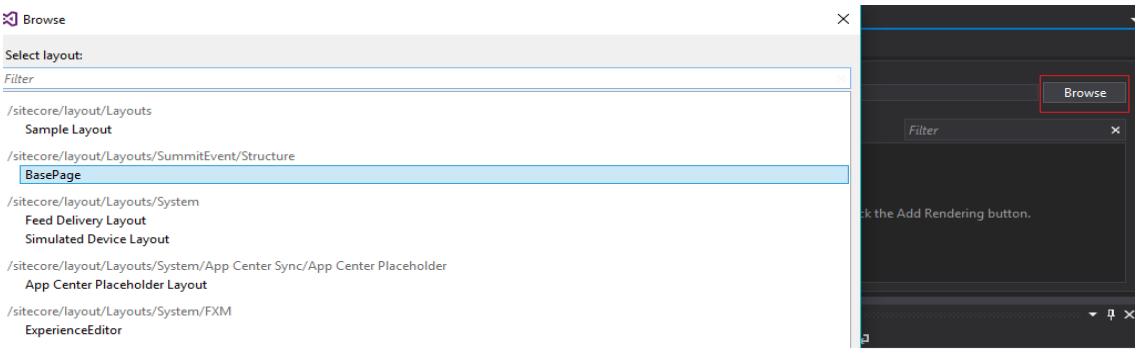


Figura 54 – Selección de *layout* para *BasePage*. Fuente: Autoría Propia

Publicamos en la instancia de Sitecore con *Smart Publish* y publicamos nuestro proyecto al *inetpub*. Si todo es correcto deberíamos poder crear un ítem con el *template BasePage* y poder ver una página con el título *Base Page Layout* y las librerías importadas como vemos en la Figura 55.

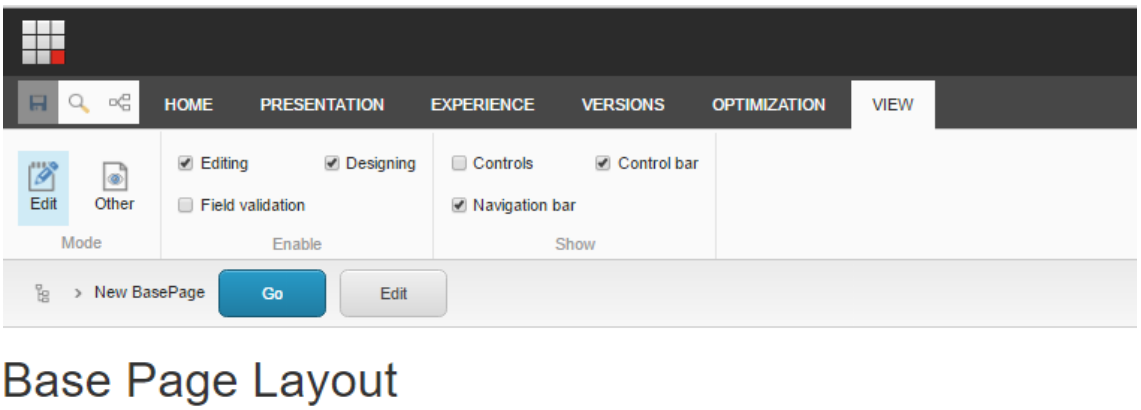


Figura 55 – Ejemplo de página utilizando la plantilla BasePage. Fuente: Autoría Propia

Por ultimo vamos a exportar nuestros nuevos ítems con *Unicorn*.

Para eso vamos a entrar a la consola de *Unicorn* por un navegador y seleccionamos la opción reserealizar, en nuestro proyecto vamos a hacer clic derecho sobre la carpeta */App_Data/Data/Unicorn* y seleccionamos reemplazar con el contenido del servidor. Vamos a ver que aparecen carpetas que no

están incluidas, debemos hacer clic en las carpetas correspondientes y agregarlas al proyecto como vemos en la Figura 56.

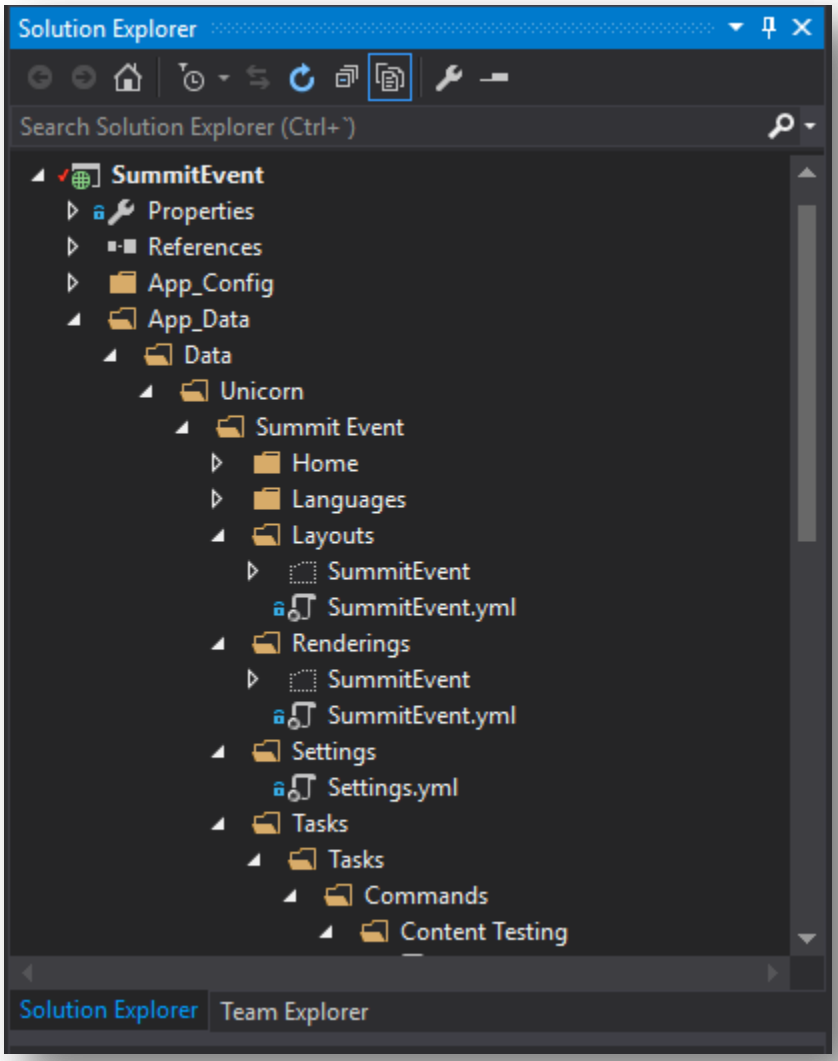


Figura 56 – Se muestran elementos no incluidos en el proyecto con el contorno en línea punteada. Fuente: Autoría Propia

7.3 Sección 2

7.3.1 Objetivo

Vamos a crear un componente llamado *Jumbo*, el cual tiene varios campos, de varios tipos.

Al componente se le puede configurar un título, subtitulo, un RTE, también se le puede asignar una imagen de fondo o simplemente un color, así como elegir el color de fuente de los textos desplegados. En la Figura 57 se puede observar una instancia del componente, ya personalizado.

Figura 57 – Vista de una instancia de componente Jumbo. Fuente: Autoría Propia

7.3.2 Desarrollo

En primera instancia se desarrolló utilizando un *Controller Rendering*, el cual hace referencia a un controlador y un método del mismo en *Visual Studio* (Figura 53), dado que aunque el *rendering* no implementa lógica de negocio si debe procesar un conjunto considerable de campos además del manejo de la imagen que posee este componente (mediante API de Sitecore). Es importante notar que aunque el nombre del controlador sea *JumboController*, para referenciarlo en Sitecore solamente se pone el nombre sin la palabra *Controller* (Figura 58).

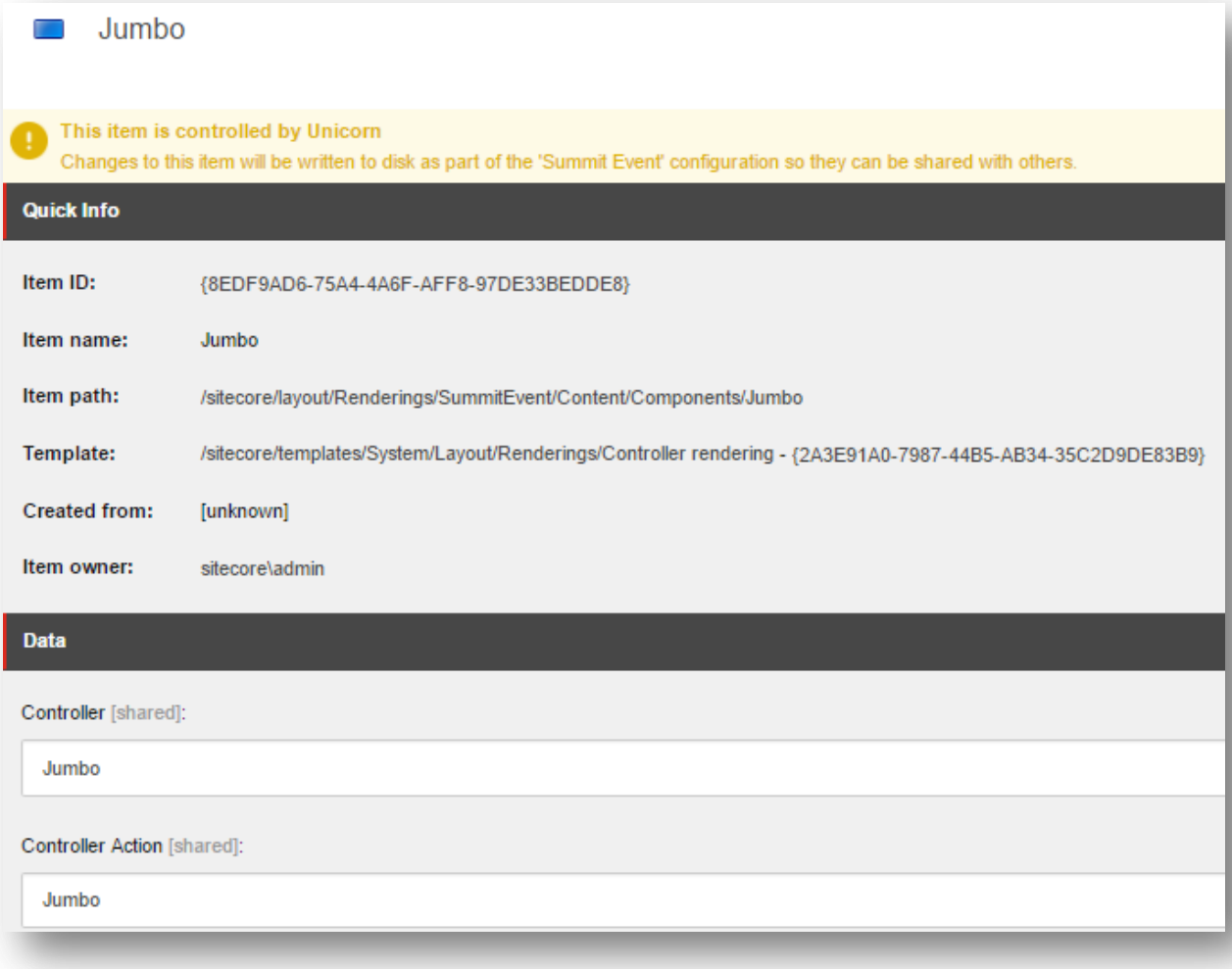


Figure 58 – Referencia a Controlador JumboController y método Jumbo. Fuente: Autoría Propia

Para la parametrización en Sitecore se utiliza un *Parameter Template*, que es referenciado en las propiedades del *rendering*.

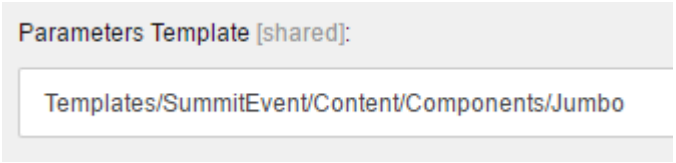


Figura 59 – Referencia a *template* que contiene los parámetros del componente. Fuente: Autoría Propia

El *template* referenciado en la imagen anterior, fue creado con los siguientes campos, como se puede observar en la Figura 60:

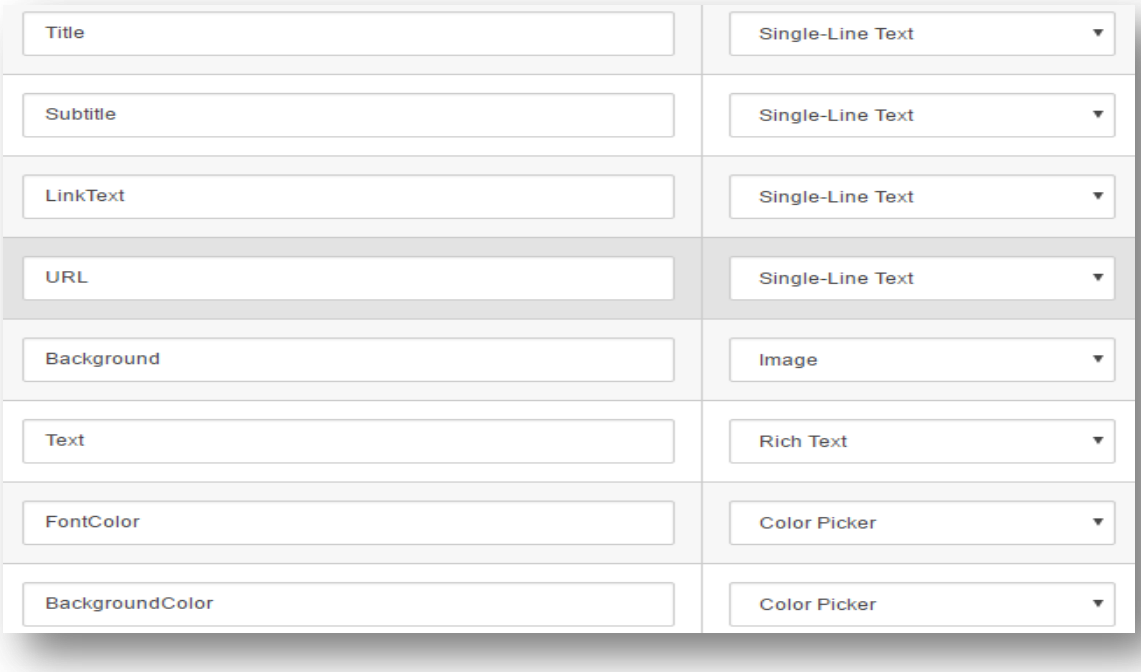


Figura 60 – Captura de pantalla que muestra los campos del *template Jumbo*. Fuente: Autoría Propia

Para la selección de color se utilizó un campo de tipo *Color Picker*, para el cual ya fue instalado mediante un paquete, como fue descrito en la sección *Uso de Plug-ins*.

También es importante configurar el *template* de forma de que herede como *template* base a *Standard Rendering Parameter* (Figura 61). De esta manera es posible que el *rendering* sea parametrizable para los autores a la hora de realizar contenido.

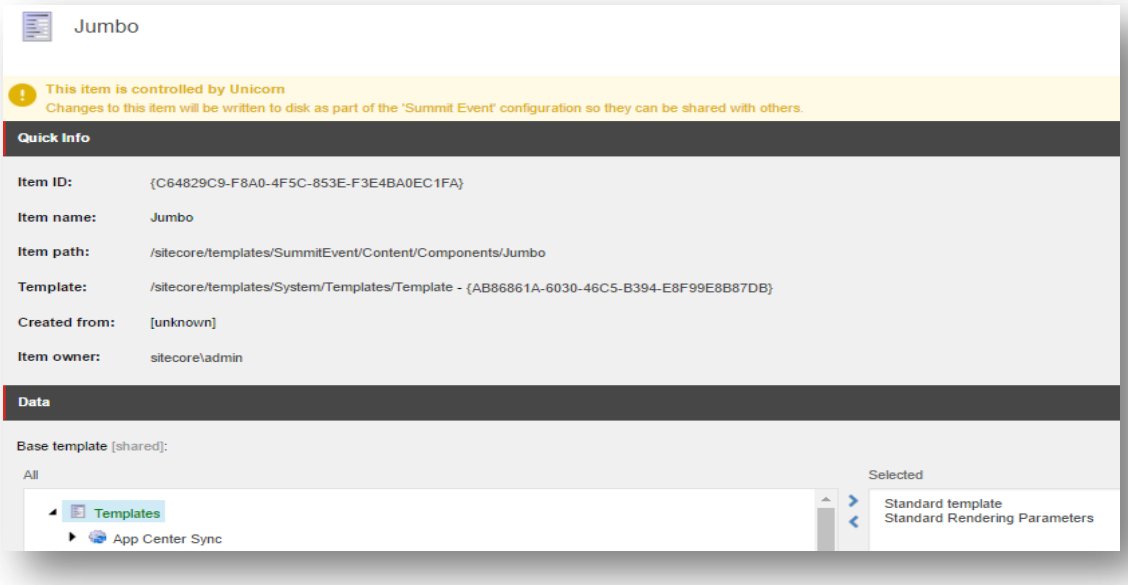


Figura 61 – Herencia de *Standard Rendering Parameters* en template *Jumbo*. Fuente: Autoría Propia

Luego de realizar todos los pasos descritos anteriormente en SItecore, se describe lo hecho en *Visual Studio*. El código del método *Jumbo* del controlador *JumboController* el siguiente, qué básicamente obtiene los datos luego de ser introducidos en el componente - en el ambiente de autor de Sitecore- así como la imagen de fondo.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Sitecore.Data.Items;
namespace SummitEvent.Controllers.Structure.Content.Components
{

public class JumboController : Controller

  {
    // GET: Jumbo
    public ActionResult Jumbo()
    {
      Sitecore.Data.Database master = Sitecore.Data.Database.GetDatabase("master");
      String image = "";
      String title = "";
      String sub = "";
      String linkText = "";
      String url = "";
      String text = "";
```

```
        String fontColor = "";
        String backgroundColor = "";
        string mediaUrl = String.Empty;
        var rc = Sitecore.Mvc.Presentation.RenderingContext.CurrentOrNull;
        try
        {
           if (rc != null)
           {
              var parms = rc.Rendering.Parameters;
              title = parms["Title"];
              sub = parms["Subtitle"];
              linkText = parms["LinkText"];
              url = parms["URL"];
              image = parms["Background"];
              text = parms["Text"];
              fontColor = parms["FontColor"];
              backgroundColor = parms["BackgroundColor"];
              if (image != null && !image.Equals(""))
              {
                 String[] processImage = image.Split('');
                 if (processImage.Length > 1)
                 {
                    Item imageItem = master.GetItem(processImage[1]);
                    if (imageItem != null)
                          mediaUrl = Sitecore.Resources.Media.MediaManager.GetMediaUrl(imageItem);
                 }
              }
           }
        }
        catch (Exception e)
        {
           Sitecore.Diagnostics.Log.Error("Excepcion" + e.Message, e);
        }
        ViewData["Image"] = mediaUrl;
        ViewData["Title"] = title;
        ViewData["Subtitle"] = sub;
        ViewData["LinkText"] = linkText;
        ViewData["URL"] = url;
        ViewData["Text"] = text;
        ViewData["FontColor"] = fontColor;
        ViewData["BackgroundColor"] = backgroundColor;
        return View("~/Views/Content/Components/Jumbo/Jumbo.cshtml");
     }
  }

}
```

En el caso de que también se utilizara un modelo, el *controller rendering* no posee una opción para una referencia a un modelo en Sitecore, por lo tanto el mismo debería ser inicializado en el código del controlador, y devuelto junto con la vista. Un ejemplo de lo dicho anteriormente sería la siguiente línea de código:

```
return View("~/Views/Structure/Components/TopNavigation/Topnav.cshtml", topnavModel);
```

donde *topnavModel* es una instancia de un modelo creada en el controlador.

Es importante notar que el pasaje de parámetros del controlador a la vista se realiza a través del diccionario *ViewData*. También hay otras dos opciones que son *ViewBag* y *TempData*.

Como se verá a continuación en el código correspondiente a la vista (Jumbo.cshtml) se verá como se recuperan los parámetros pasados en el controlador y se trabaja con ellos en el código HTML:

```
@using Sitecore.Mvc
@using Sitecore.Mvc.Presentation
@model RenderingModel
@using Sitecore.Data.Items;
@using Sitecore.Data.Fields;

@{
    var imageUrl = ViewData["Image"];
    var title = ViewData["Title"];
    var subTitle = ViewData["Subtitle"];
    var linkText = ViewData["LinkText"];
    var url = ViewData["URL"];
    var text = ViewData["Text"];
    var fontColor = ViewData["FontColor"];
    var backgroundColor = ViewData["BackgroundColor"];
}

@Html.Partial("~/Views/Partials/_RenderingName.cshtml")

<div >
    <section class="jumbo-container" style="background-image:url(@imageUrl);background-
color:@backgroundColor">
        <div class="jumbo col-md-8 col-md-offset-2" style="color:@fontColor">
            <h2>@title</h2>
            <h3>@subTitle</h3>
            <h4>@text</h4>
            <p>
                <a href=@url>@linkText</a>
            </p>
        </div>
    </section>
</div>
```

7.4 Desarrollo Móvil

Vamos a ver ejemplos prácticos para desarrollar partes de una aplicación móvil conectándonos a la instancia de Sitecore con la que venimos trabajando previamente, a través de la API Sitecore.MobileSDK. Se detallarán las operaciones más importantes que se pueden realizar con la API mencionada. Durante este capítulo se hará referencia solamente a desarrollo para sistemas Android.

7.4.1 Conceptos clave

Item

API

Sesión

7.4.2 Requisitos y configuración inicial

7.4.2.1 Requisitos

Para poder trabajar con tecnologías móviles desde Visual Studio, se deben tener instaladas las siguientes herramientas:

1. *Java Development Kit* (JDK)
2. Android SDK
3. Android NDK
4. *Xamarin for Visual Studio*
5. Paquete NuGet Sitecore.MobileSDK
6. Paquete NuGet Sitecore.PasswordProvider

7.4.2.2 Configuración

Se deben establecer las rutas de ubicación de JDK, Android SDK y Android NDK en las propiedades de Visual Studio (Figura 62)

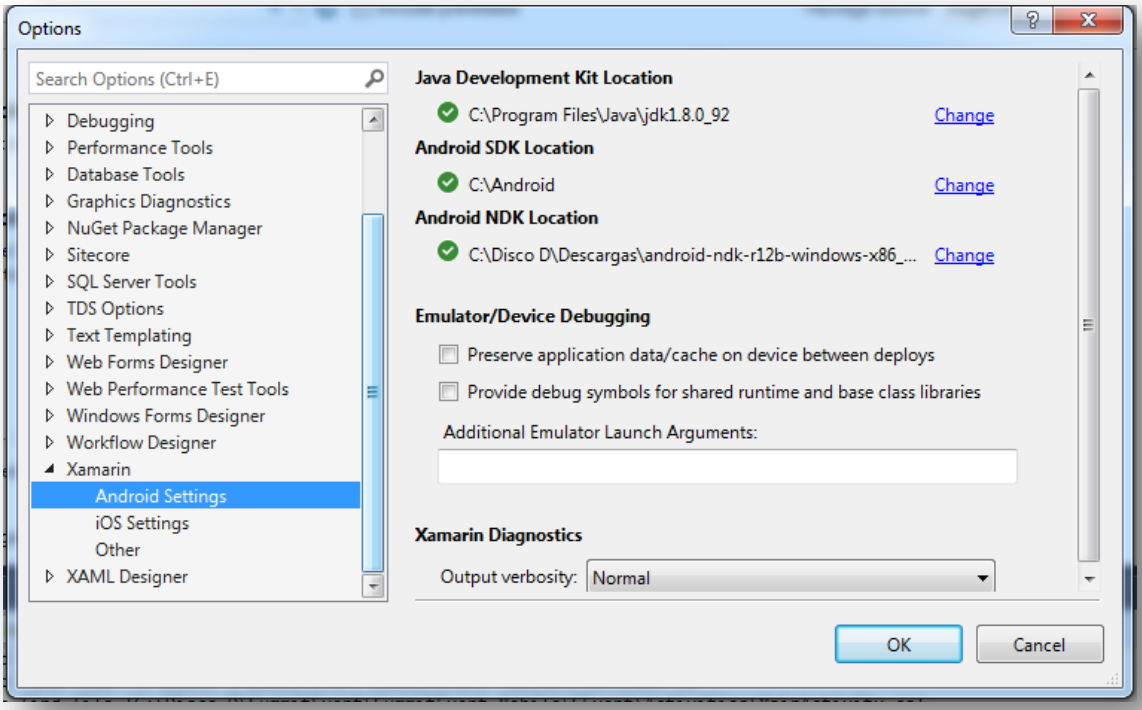


Figura 62 – Configuración de JDK, SDK y NDK en Visual Studio. Fuente: Autoría Propia

Además. Para poder interactuar con los ítems de nuestra instancia de Sitecore, hay que realizar unas modificaciones en el archivo *Sitecore.ItemWebApi.config*, ubicado en la carpeta *C:\inetpub\wwwroot\InstanciaSitecore\Website\App_Config\Include* donde *instanciaSitecore* es el nombre de su proyecto. Lo configuramos de la siguiente forma:

```
<site name="website">
  <patch:attribute name="itemwebapi.mode">StandardSecurity</patch:attribute>
  <patch:attribute name="itemwebapi.access">ReadWrite</patch:attribute>
  <patch:attribute name="itemwebapi.allowanonymousaccess">false</patch:attribute>
</site>
```

Figura 58 – Configuración de archivo Sitecore.ItemWebApi.config

Si solo se quieren obtener ítems de la instancia de Sitecore, la propiedad *access* puede ser *ReadOnly* en vez de *ReadWrite*.

Para poder acceder a la instancia de Sitecore a través de un dispositivo móvil, son necesarios los siguientes pasos:

- En el administrador de IIS, en sitios se hace clic derecho sobre nuestra instancia de Sitecore y se selecciona la opción *Modificar enlaces…* (Figura 63)

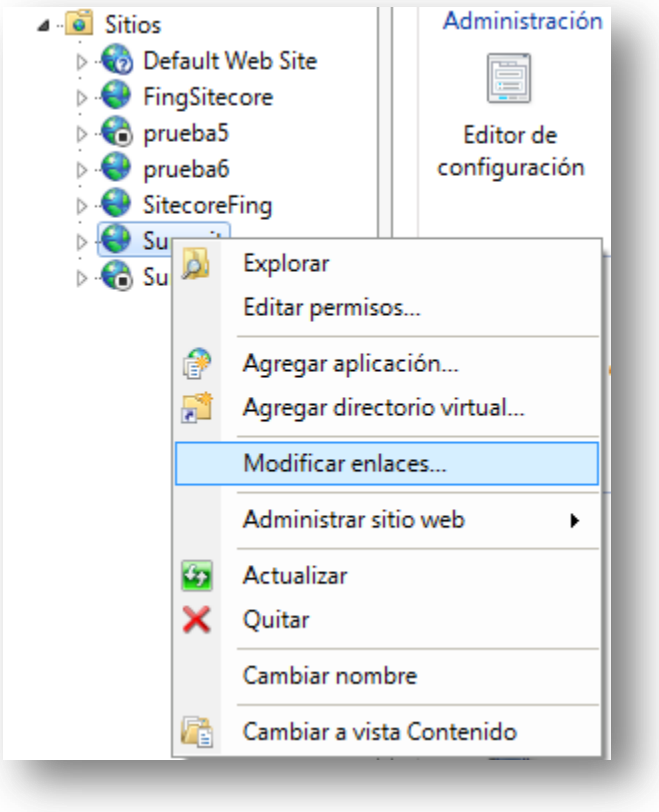


Figura 63 – Configuración de enlaces para un sitio web a través de IIS. Fuente: Autoría Propia

- Averiguamos la dirección IP nuestro equipo (mediante el comando *ipconfig* desde una consola de línea de comandos) y agregamos esa dirección y un puerto a nuestra elección (nombre del host es opcional) como se muestra en la Figura 64.

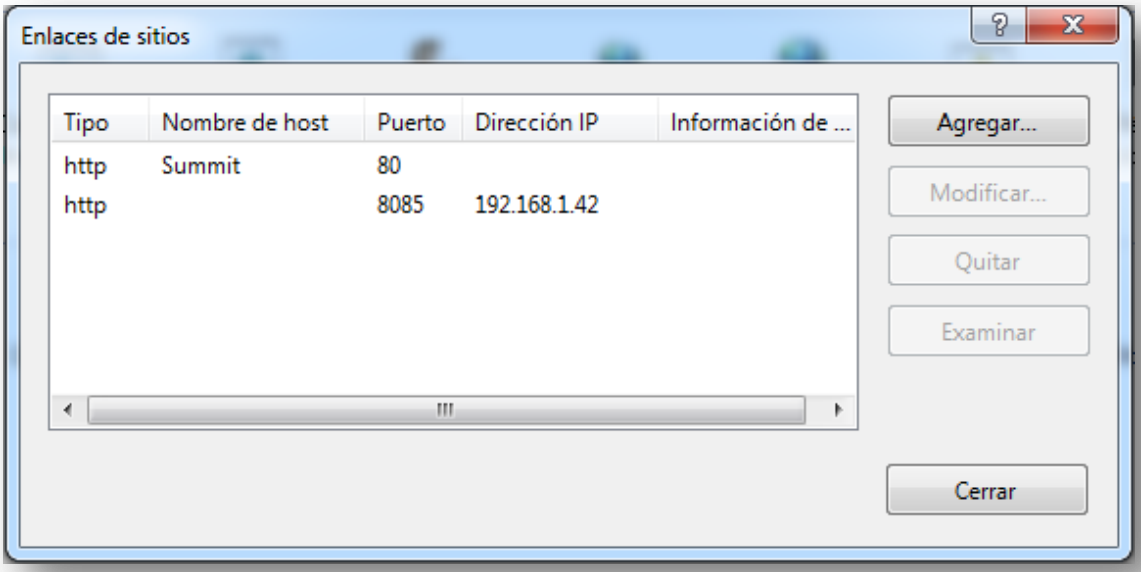


Figura 64 – Agregado de un enlace. Fuente: Autoría Propia

- Por último, hay que agregar una regla de entrada en nuestro *Firewall*. Para esto ingresamos a *Firewall* con seguridad avanzada, a la izquierda seleccionamos Reglas de entrada y sobre la derecha damos clic en Nueva regla (Figura 65)

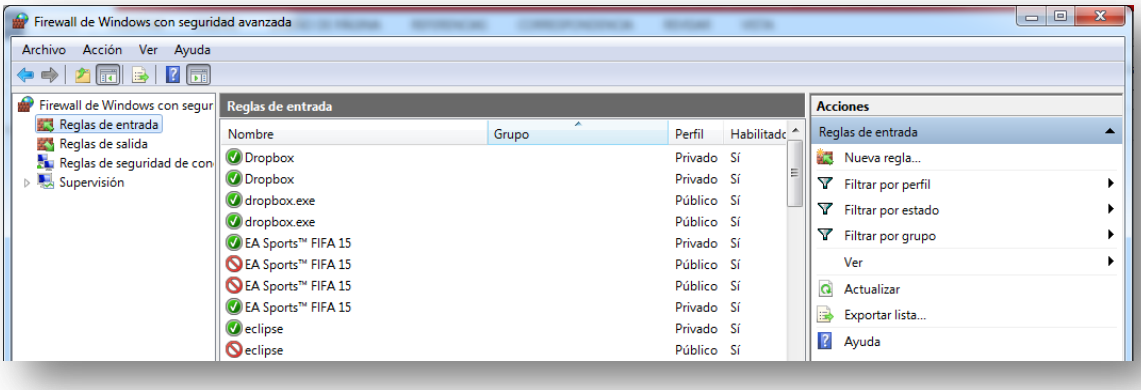


Figura 65 – Firewall de Windows con seguridad avanzada. Fuente: Autoría Propia

- Se abrirá un diálogo en el cual preguntará que tipo de regla queremos agregar, seleccionamos Puerto. Al avanzar nos pregunta por la conexión (TCP o UDP), seleccionamos TCP y un número de puerto, en donde ingresamos el mismo puerto que seleccionamos en el enlace que agregamos en IIS a nuestra instancia (Figura 66).

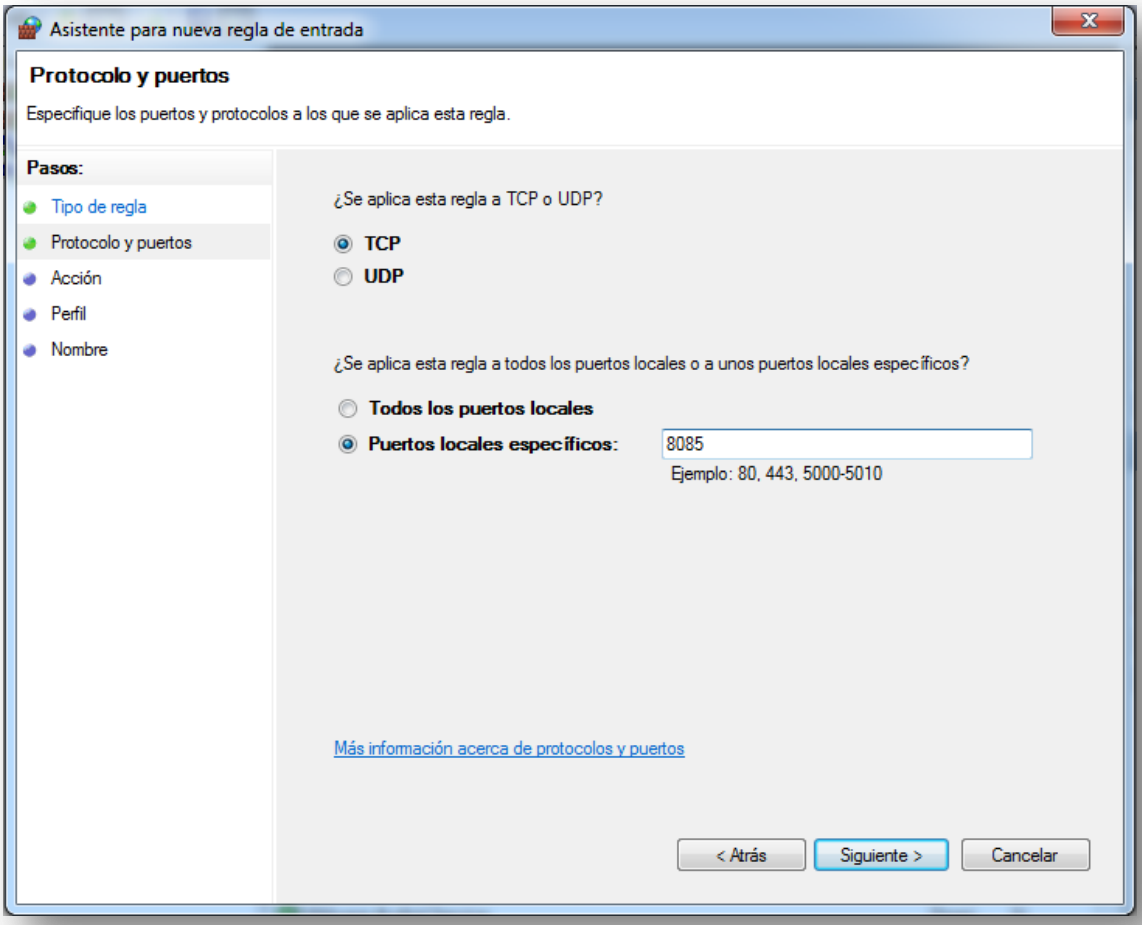


Figura 66 – Configuración de regla de entrada en firewall de Windows. Fuente: Autoría Propia

- Para las siguientes pantallas (*Acción* y *Perfil*) no es necesario hacer modificaciones, por lo tanto avanzamos hasta la última en donde se pide nombrar la regla y damos clic en finalizar.

Con estos pasos, ya estamos en condiciones de poder acceder desde nuestro dispositivo móvil a nuestra instancia de Sitecore. Esto lo podemos comprobar abriendo un navegador e ingresando en la URL 192.168.1.42:8085/sitecore (IP y puerto del enlace agregado en el administrador de IIS) y si se despliega la pantalla de *login* de Sitecore, ya estamos listos para trabajar.

7.4.3 Desarrollo

La API utilizada para la conexión con la instancia Sitecore, dispone (entre otras) de funciones de lectura, modificación, creación, borrado de ítems, descarga de recursos multimedia.

A continuación se detallarán las funciones y clases involucradas para llevar a cabo las funciones mencionadas anteriormente.

7.4.3.1 Iniciar una sesión

Para realizar cualquier operación sobre ítems, se debe iniciar una sesión en primer lugar. Para esto se utiliza la clase *SitecoreWebApiSessionBuilder*, la misma contiene dos métodos que son para iniciar una sesión anónima o autentificada, *AuthenticatedSessionWithHost* *AnonymousSessionWithHost* respectivamente, generando un elemento de tipo *ISitecoreWebApiSession* (variable *session* en este caso, que usaremos a lo largo de todos los ejemplos siguientes).

En la siguiente figura se muestra un ejemplo de inicio de sesión:

```
        var session =
SitecoreWebApiSessionBuilder.AuthenticatedSessionWithHost("http://192.168.1.42:8085")
                                    .Credentials(new SecureStringPasswordProvider(user, pass))
                                    .WebApiVersion("v1")
                                   .DefaultLanguage("en")
                                   .MediaLibraryRoot("/sitecore/media library")
                                   .MediaPrefix("~/media/")
                                   .DefaultMediaResourceExtension("ashx")
                                   .BuildSession();
```

El *string* que se pasa por parámetro en el método AuthenticatedSessionWithHost es la dirección IP y puerto que se configuraron en el enlace del administrador de IIS (descrito en configuración), además de ir precedido por http://.

Si se usa inicia una sesión autenticada, es necesario agregar a continuación el parámetro *Credentials* del tipo IWebApiCredentials, para el cual se le pasan dos *strings* correspondientes al nombre del usuario y contraseña.

Se cuentan con varios parámetros auxiliares, para los cuales no vamos a hondar en detalles, pero es importante comentar que pueden ir en distinto orden luego de establecidos los parámetros de la instancia y las credenciales.

Al ser creada la instancia de sesión (bajo la variable *session* en la imagen), se puede proceder a realizar acciones sobre los ítems de la instancia de Sitecore. Es importante destacar que los métodos que brinda la instancia creada, son todos asincrónicos, por lo tanto todos métodos implementados que interactúen con ítems serán asíncronas, y lo devuelto en dichas funciones deberá ser precediro por la palabra clave *await* (se verá en ejemplos a continuación).

7.4.3.2 Buscar un ítem

Antes de poder realizar una lectura de un ítem, es necesario realizar el pedido, el cual es hecho a través de la clase *ItemWebApiRequestBuilder*, habiendo tres opciones, pasando en las tres opciones como parámetro un *string*:

- **Por ID del itém**

```
        var request = ItemWebApiRequestBuilder.ReadItemsRequestWithId(id)
```

```
        .AddScope(scope)
        .Payload(payload)
        .Database(database)
        .Build();
```

- **Por ruta del ítem en la jerarquía de Sitecore**

```
var request = ItemWebApiRequestBuilder.ReadItemsRequestWithPath(path)
                .AddScope(scope)
               .Payload(payload)
               .Database(database)
               .Build();
```

- **Por consulta**

```
var request = ItemWebApiRequestBuilder.ReadItemsRequestWithSitecoreQuery(query)
               .AddScope(scope)
               .Payload(payload)
               .Database(database)
               .Build();
```

Para las tres variantes presentadas, se observa que hay tres parámetros utilizados (opcionales), los cuales son

- ***AddScope***
  Alcance del item, el cual puede ser:
  - *self* (el mismo ítem).
  - *default.*
  - *children* (hijos del ítem)
  - *parents* (padres del ítem).
- ***Payload***
  Se presentan también cuatro opciones, min, *default*, *content* y *full*.
- ***Database***
  Base de datos a la cual se realiza el pedido de lectura, si no se pone este parámetro, por defecto es la base *master*.

Adicionalmente hay otros dos parámetros (también opcionales):

- **Language**
  Lenguaje en el que el contenido ítem es presentado al realizar la lectura
- **Fields to Read**
  Selección de campos de un ítem que se quieren leer (en caso de no querer obtener todo e ítem)

Finalmente, podemos conseguir el ítem solicitado mediante:

```
var response = await session.ReadItemAsync(request);
```

donde *response* es un objeto de tipo *ScItemsResponse*. A continuación podemos acceder finalmente a los ítems buscados:

```
ISitecoreItem  item = response[0];
```
(primer ítem, depende del alcance y de la cantidad encontrada)

Y finalmente se accede al mismo y a sus campos de la siguiente forma:

```
string title = item["Title"].RawValue;
```
(el ítem tiene un campo de nombre *Title*)

Es importante notar que solamente se pueden obtener campos de información en forma de *string*, si originalmente no son de ese tipo, hay que hacer el procesamiento necesario para obtener los datos como los precisamos.

7.4.3.3 Buscar una imagen

La clase *ItemWebApiRequestBuilder* provee el método *DownloadResourceRequestWithMediaPath(string)*, el cual al pasarle cómo parámetro la ruta del archivo de imagen en la jerarquía de Sitecore, se construye el pedido de la misma. Se ejemplifica a continuación:

```
var request = ItemWebApiRequestBuilder.DownloadResourceRequestWithMediaPath(path).Build();
```

A continuación, se recupera el archivo de imagen (de tipo *System.IO.Stream*) con la siguiente invocación:

```
var image = session.DownloadMediaResourceAsync(request);
```

A partir de este momento se puede operar con el flujo de datos perteneciente a la imagen. Por ejemplo, si se quisiera mostrar la foto, vinculándola a un campo (variable *imageView1*) de una vista de nuestra aplicación móvil, se podría hacer de la siguiente manera:

```
Drawable d = Drawable.CreateFromStream(image, null);
imageView1 = FindViewById<ImageView>(Resource.Id.imageView1);
imageView1.SetImageDrawable(d);
```

7.4.3.4 Crear un ítem

La clase *ItemWebApiRequestBuilder* provee dos métodos:

- **CreateItemRequestWithParentPath(path)**
  Se le pasa cómo parámetro la ruta dentro de la jerarquía de Sitecore del padre del ítem a crear (*string*).
- **CreateItemRequestWithParentPath(id)**
  Se le pasa como parámetro el ID correspondiente al padre del ítem a crear (*string*).

Se plantea a continuación un ejemplo de consulta a través de la ruta del padre:

```
var request = ItemWebApiRequestBuilder.CreateItemRequestWithParentPath(path)
                    .ItemTemplateId(templateID)
                    .ItemName(itemName)
                    .AddFieldsRawValuesByNameToSet(fields)
                    .Database(database)
                    .Build();
```

Se cuenta con dos parámetros obligatorios:

- **ItemTemplateId, ItemTemplatePath o ItemTemplateBranch**
  Identifica el *template* en el que se basa el ítem a crear. Como se menciona por los parámetros, se puede realizar por ID del *template*, por ruta o por rama. Importante, solo usar una de las tres opciones en un pedido, sino se retornará un error.
- **ItemName**
  Nombre del nuevo ítem.

Se cuenta con otros parámetros auxiliares:

- **Language**
  Lenguaje del contenido del ítem.
- **Database**
  Base de datos en donde va a ser almacenado el ítem creado.
- **AddFieldsRawValuesByNameToSet(string key, string rawValue) o AddFieldsRawValuesByNameToSet(IDictionary)**
  Información a agregar en los campos del ítem a crear. Se puede agregar un parámetro por campo (primera opción) o se puede crear una instancia de *Dictionary* donde ya se incluyen todos los campos y su respectiva información (segunda opción).

Ejemplo de primera opción:

```
var request = ItemWebApiRequestBuilder.CreateItemRequestWithParentPath(parentItemPath)
                    .ItemTemplatePath(“Sample/Sample Item”)
                    .ItemName(“new item”)
                    .AddFieldsRawValuesByNameToSet("Title", “Device”)
                    .AddFieldsRawValuesByNameToSet("Text", “Smartphone”)
                    .Build();
```

Ejemplo de segunda opción:

```
var fields = new Dictionary<string, string>();
fields.Add(“Title”, “Device”);
fields.Add(“Text”, “Smartphone”);
var request = ItemWebApiRequestBuilder.CreateItemRequestWithParentPath(parentItemPath)
            .ItemTemplatePath(“Sample/Sample Item”)
            .ItemName(“new item”)
```

```
                .AddFieldsRawValuesByNameToSet(fields)
                .Build();
```

Finalmente se crea el ítem realizando la siguiente invocación:

```
ScItemsResponse createResponse = await session.CreateItemAsync(request);
```

7.4.3.5 Borrar un ítem

Como en la lectura de un ítem, la clase *ItemWebApiRequestBuilder* ofrece tres opciones, pasando en las tres opciones como parámetro un *string*:

- **Por ID del ítem a eliminar**

```
var request = ItemWebApiRequestBuilder.DeleteItemRequestWithPath(itemPath)
                .Database("master")
                .Build();
```

- **Por ruta del ítem en la jerarquía de Sitecore**

```
var request = ItemWebApiRequestBuilder.DeleteItemRequestWithId(id)
                .Database("master")
                .Build();
```

- **Por consulta**

```
var request = ItemWebApiRequestBuilder.DeleteItemRequestWithPath(query)
                .Database("master")
                .Build();
```

Como parámetros opcionales se cuenta con:

- **Database**
  Base de datos de la que se va a eliminar el ítem.
- **AddScope**
  Permite borrar el ítem solamente, sus hijos y/o sus padres. Se agrega un ejemplo:}

```
var request = ItemWebApiRequestBuilder.DeleteItemRequestWithSitecoreQuery(queryString)
                .AddScope(ScopeType.Self, ScopeType.Children)
                .Build();
```

Finalmente se elimina el ítem realizando la siguiente invocación:

```
ScItemsResponse response = await session.DeleteItemAsync(request);
```

8 Referencias

[1] http://www.sitecore.net/. Último acceso: 20/10/2016.
[2] https://community.sitecore.net/technical_blogs/b/sitecorejohn_blog/posts/oversimplified-conceptual-architecture-of-the-sitecore-asp-net-cms. Último Acceso 11/12/2016.
[3] John West. *Professional Sitecore Development.* John Wiley & Sons, Inc, 2012.
[4] Herbert Schildt. *C# 4.0: The Complete Reference.* McGraw-Hill. 2010.
[5] Joseph Albahari, Ben Albahari. *C# 5.0 in a nutshell.* O'Reilly Media, Inc. 2012.
[6] https://www.iis.net. Último acceso: 26/10/2016.
[7] https://www.visualstudio.com/es/vs/. Último acceso: 17/10/2016.
[8] https://msdn.microsoft.com/es-es/library/bb545450.aspx. Último acceso: 17/10/2016.
[9] https://marketplace.sitecore.net/. Último acceso: 17/10/2016.

Anexo 5: Solución completa del caso de estudio en Sitecore

Índice

1 Introducción

Previamente a describir el desarrollo de cada componente, se hará una breve introducción a las tecnologías usadas y ambiente de desarrollo utilizado así como las principales decisiones tomadas al respecto.

Se describirán con mayor detalle los desarrollos correspondientes al *template BasePage* y componente *Topnav*, ya que cubren gran parte de las técnicas utilizadas para desarrollar en dicha plataforma.

En cuanto al desarrollo de los componentes, se detallará el tipo de *rendering* utilizado, la estrategia tomada para la parametrización, y detallar sus parámetros.

2 Ambiente de desarrollo

2.1 Ambiente de desarrollo integrado

En cuanto al ambiente de desarrollo integrado, se realiza el proyecto en Microsoft Visual Studio. Es la herramienta por excelencia para el desarrollo de aplicaciones de tecnologías Microsoft, como es en este caso.
Como no existe una estructura oficial de proyecto recomendada por Sitecore –existen recomendaciones-, la misma tuvo que ser creada de cero para la implementación del caso de estudio.

Las principales herramientas que vamos a utilizar para desarrollar el caso de estudio son:

- *IDE* Microsoft Visual Studio Community 2015 versión 14.0.2 Update 2
- *Microsoft .NET framework* versión *4.5.*

2.1.1.1 Logview4net

Herramienta gratuita para monitoreo y visualización de registro [1].
A través de esta herramienta, y configurando un *listener* de tipo UDP con el puerto obtenido en el archivo Sitecore.config del proyecto de *Visual Strudio*, es posible monitorear en tiempo real el registro de Sitecore. Resultando de mucha utilidad para la resolución de problemas, por ejemplo, colocando mensajes (en este caso de tipo de error o advertencia) entre líneas de código a evaluar.

2.1.2 Frameworks

2.1.2.1 AngularJS

Framework para aplicaciones web dinámicas. Permite usar HTML como lenguaje de plantilla y extender la sintaxis de HTML [2].

2.1.2.2 Bootstrap

Framework HTML, CSS y JavaScript para desarrollo web [3].

2.1.2.3 FullCalendar.io

Calendario de eventos hecho en JavaScript. Personalizable y de código abierto.
Despliega un calendario en tamaño grande de eventos que se pueden arrastrar y soltar [4].
Para la solución descrita en este informe, se adaptó el mismo a fin de conseguir el funcionamiento deseado con las tecnologías con las que se ha trabajado (en particular para Sitecore).

2.1.2.4 Glass.Mapper versión 4.2.0.184

Framework que realiza mapeo objeto-relacional (ORM en inglés) que permite el mapeo de datos a modelos [5].

2.1.2.5 Sitecore Rocks versión 2.0.39.0

Framework que se integra a *Visual Studio* y provee una experiencia de desarrollo rápido y coordinado [6]. A través de esta herramienta se puede acceder a las tres bases de datos que proporciona Sitecore (descrito en sección *Publicación* del *Anexo 4*) y por consiguiente se tiene acceso a todo el contenido de la instancia sin tener que acceder a ella a través de un navegador web.

2.2 Ambiente de desarrollo integrado

2.2.1 Base de datos

Para la base de datos se decide trabajar con *Microsoft SQL Server* con el fin de trabajar con tecnologías del mismo proveedor y así evitar algún eventual problema de compatibilidad.

2.2.2 Repositorio

Bitbucket fue el repositorio utilizado para el proyecto. El mismo es un sistema Git de control de versiones distribuido [7]. Dicha plataforma ofrece un repositorio privado (a diferencia de Github por ejemplo) y gratuito para una cantidad limitada de desarrolladores. Como este proyecto está integrado solamente por dos personas, es una opción más que adecuada.

2.2.3 Servidor web

Se utiliza IIS, ya que es el servidor por defecto para tecnologías Microsoft, además de ser el requerido por Sitecore para su funcionamiento.

2.2.4 Trabajo colaborativo en Sitecore

En un proyecto donde coexisten dos o más desarrolladores que cuentan con una instancia local de Sitecore, la sincronización de la solución se vuelve una tarea compleja, ya que Sitecore no cuenta con una funcionalidad especifica o herramienta propia para sincronizar proyectos en una misma instancia en diferentes equipos.

Por este motivo debimos buscar herramientas externas a la plataforma para facilitar dicha tarea, la cual se reduce a sincronizar bases de datos relacionales.

En primera instancia, nos encontramos con una herramienta llamada TDS (*Team Development for Sitecore*) [8] la cual no sólo ofrece la sincronización de proyectos sino que también otras funcionalidades específicas. La misma fue descartada ya que es una herramienta propietaria y por lo tanto requiere de una licencia para su uso. Otra posible solución recomendada por Sitecore, propone que todos los desarrolladores pertenecientes al proyecto configuren las instancias locales para que compartan una misma base de datos alojada la nube, esto también fue descartado ya que no contamos con dicha infraestructura.

El resultado de la búsqueda se redujo a tres herramientas *open-source.* Las mismas proveen funcionalidades específicas o parciales para la sincronización de proyectos en Sitecore y son las siguientes:

2.2.4.1 Sitecore Ship

Sitecore Ship [9] permite instalar paquetes que contienen diferentes objetos de Sitecore tales como ítems, *templates* y *renderings* con sus valores correspondientes a través de un pedido

HTTP. Ésta funcionalidad resultaría más útil en caso de tener un servidor remoto para el proyecto en sí, sin embargo no ofrece opciones respecto a la sincronización, es recomendable la utilización de dicha herramienta en combinación con otras.

Sitecore Courier

Sitecore Courier [10] tiene como objetivo acortar la brecha entre el desarrollo y los entornos de producción cuando la construcción de sitios web es realizada con Sitecore. La herramienta permite crear paquetes analizando artículos de Sitecore serializados y empaquetando únicamente los que fueron modificados. Estos paquetes son persistidos de forma independiente al código, limitando el control y versionado de los mismos.

2.2.4.2 Unicorn versión 3.3.1

Unicorn [11] resuelve este problema de sincronización escribiendo copias serializadas de los elementos de Sitecore en el disco junto con el código. De esta manera, una copia de los elementos de la base de datos necesarios para la solución forma parte del código fuente del proyecto. Los mismos son versionados en el repositorio GIT, lo cual ofrece una ventaja fundamental por la cual decidimos utilizar *Unicorn versión 3.3.1* como herramienta de sincronización. Por otra parte, también proporciona una interfaz web integrada con Sitecore para sincronizar o serializar los ítems que los desarrolladores consideren necesarios.

2.3 Tecnologías utilizadas

2.3.1 AJAX

Acrónimo de **A**synchronous **J**avaScript **a**nd **X**ML. Técnicas de programación de web que usan escritura HTTP para cargar información a demanda sin necesidad de refrescar páginas [12].

2.3.2 CSS

Lenguaje que describe el estilo de un documento HTML y como los elementos del mismo deben ser desplegados [13].

2.3.3 JavaScript

JavaScript es un lenguaje de programación de alto nivel, dinámico, no tipado e interpretado que es adecuado para estilos de programación orientada a objetos y funcional.
JavaScript es parte de las tres de tecnologías básicas para el desarrollo web: HTML para especificar el contenido de páginas web, CSS para especificar la presentación de las páginas web y JavaScript para especificar el comportamiento de las páginas web [12].

2.3.4 jQuery

Librería de JavaScript rápida, pequeña y rica en prestaciones [14].

2.3.5 Razor versión 3.2.3

Razor es una sintaxis de marcado que permite incorporar código basado en servidor (*Visual Basic* y C #) en las páginas web. Está basado en ASP.NET, y diseñado para la creación de aplicaciones web. Tiene el poder de marcado ASP.NET tradicional, pero es más fácil de utilizar, y fácil de aprender.
El código basado en el servidor puede crear contenido web dinámico sobre la marcha, mientras que una página web se escribe en el navegador. Cuando una página web se llama, el servidor ejecuta el código basado en un servidor dentro de la página antes de que devuelva la página al navegador. Mediante la ejecución en el servidor, el código puede realizar tareas complejas, como el acceso a bases de datos [15].

3 Estructura del proyecto

El proyecto en Visual Studio es un proyecto MVC vacío en un principio, luego se agregaron las carpetas *App_Config*, *Content*, *fonts*, *scripts* y *SummitEvent*. La estructura se muestra a continuación en la Figura 1.

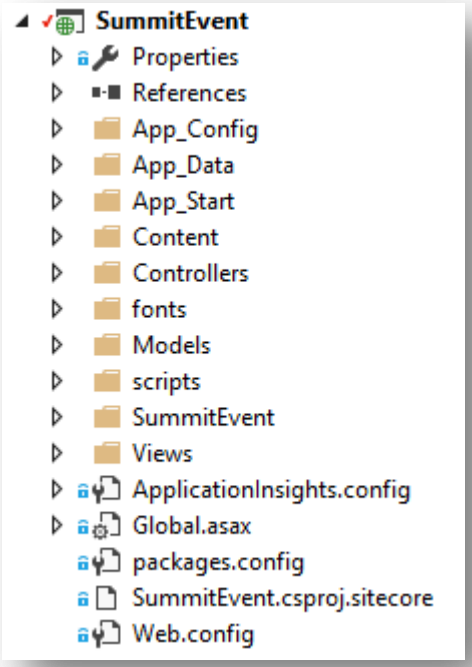


Figura 1 – Captura de pantalla de la estructura de proyecto MVC en Visual Studio. Fuente: Autoría Propia

Las carpetas *Controllers*, *Models* y *Views* tienen una estructura común, la cual consta de dos carpetas llamadas Content y *Structure* y a su vez cada una de ellas contiene una carpeta llamada *Componentes* que contiene, dependiendo de la carpeta padre, controladores, modelos o vistas. La subcarpeta *Content* contiene lo relacionado a los componentes que no son ni el *TopNav* ni el *Footer*, que se encuentran en la carpeta *Structure*.
En la Figura 2 se muestra lo descrito anteriormente para el caso de los modelos.

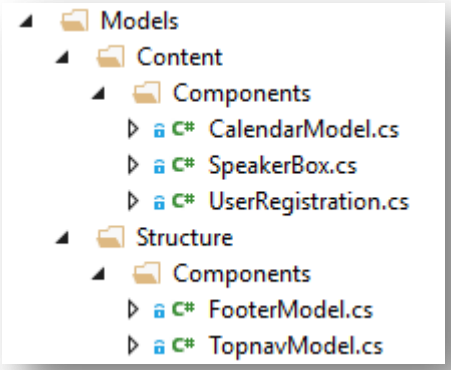


Figura 2 – Captura de pantalla de la estructura del proyecto en Visual Studio. Fuente: Autoría Propia

La misma estructura recién descrita se aplica para el directorio en donde se encuentran los ítems de Sitecore que son sincronizados a través de *Unicorn* (Figura 3).

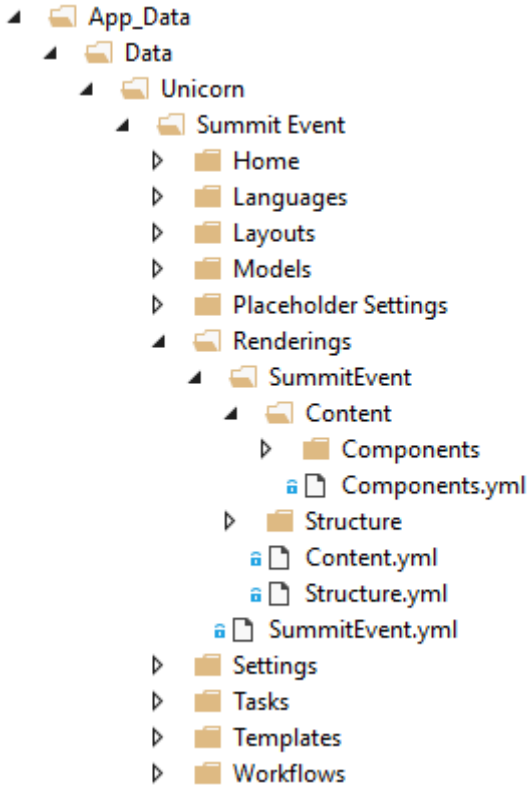


Figura 3 – Captura de pantalla de la estructura de ítems sincronizados a través de Unicorn en Visual Studio. Fuente: Autoría Propia

Se realizó esta organización para mantener un mapeo más sencillo entre el proyecto de *Visual Studio* y la sincronización de la instancia Sitecore a través de Unicorn.

Del lado de la instancia de Sitecore, se realiza una organización similar para las carpetas *Renderings*, *Layouts* (a su vez dentro de *Layout*), *Templates* y Models. *Renderings*, *Layouts* y *Templates* ya vienen por defecto en el editor de contenido, sin embargo la carpeta *Models* fue creada manualmente.

Dentro de las citadas carpetas, se crea una carpeta con el nombre del proyecto, en este caso *SummitEvent*, para separar de forma clara el contenido referido a cada proyecto (puede haber varios en una misma instancia). La estructura interna a la carpeta citada es la misma que se utilizó en Visual Studio. Se ejemplifica en la siguiente figura (Figura 4), para el caso de los *renderings*:

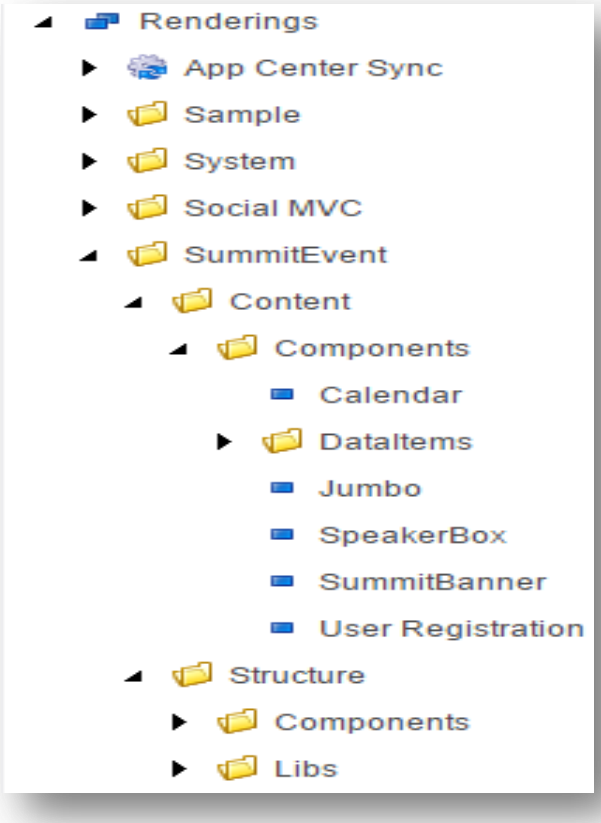


Figura 4 – Captura de pantalla de la estructura del proyecto en *Sitecore*. Fuente: Autoría Propia

4 Componentes

4.1 Estructura del sitio web

De manera similar a AEM y utilizando los mismos criterios, decidimos dividir los componente en estructurales y de contenido.

En primer lugar se construyó el *template BasePage* que representa a las páginas del sitio web. Luego se desarrollaron los componentes de la estructura que mantienen el mismo contenido a través del sitio. Dichos componentes son: *TopNav* y *Footer*.

4.2 Componentes estructurales

4.2.1 BasePage

Se creó un *template* llamado BasePage así como un Layout con el mismo nombre, los cuales están relacionados y juntos representan a todas las páginas que integran el sitio web. El *template* cuenta con un campo booleano del tipo *checkbox* llamado *Hide in Topnav* que va a ser utilizado por los autores para mantener las páginas que prefiera ocultas del navegador, a su vez, el *layout* tiene asociado un archivo BasePage.cshtml que contiene la estructura HTML principal de la página del sitio web e incluye tres *placeholders*. Los mismos son *topnav, main y footer,* a su vez dicho archivo tiene asociado de manera estática dos *view rendering -Footlibs* y *Headlibs*- los cuales son encargados de importar las diferentes librerías de estilos y *scripts* de todo el sitio.

Con el objetivo que todos los ítems creados a partir del *template BasePage* contengan una configuración pre definida, se utiliza la funcionalidad de *standard values* y mediante la misma se asocia al *layout* antes mencionado. A su vez se incluyen inicialmente los *renderings TopNav* y *Footer* en los *placeholders topnav* y *footer* respectivamente. De esta forma todos los ítems creados a partir del *template BasePage* van a contar con un *layout*, tres *placeholders* y cuatro *renderings* encargados de las librerías, navegación y *footer* (Figura 5)

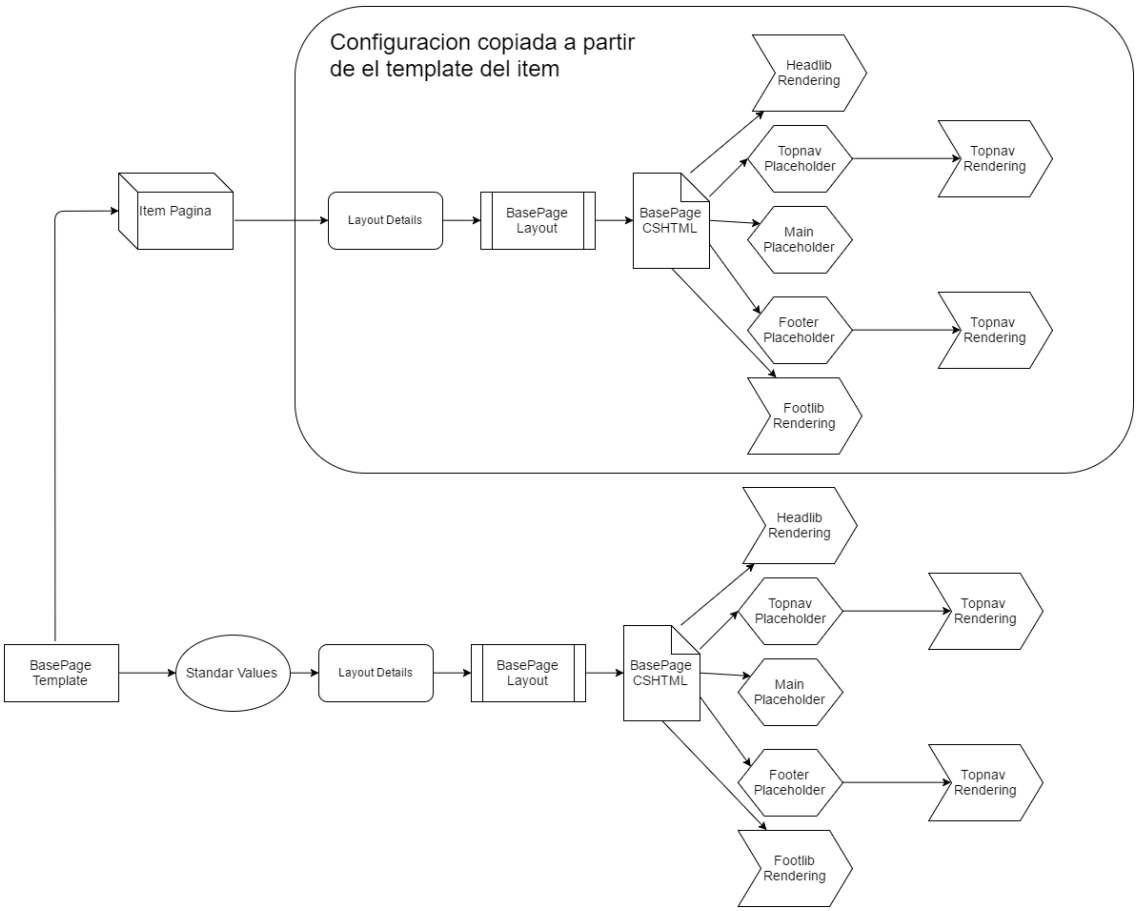


Figura 5 – Esquema de conceptos de *Templates*, *Standard Values* e Items. Fuente: Autoría Propia

4.2.1.1 Headlibs y footlibs

Es importante notar que tanto *Headlibs* como *Footlibs* se definieron como *view renderings*, ya que al no aplicar ningún tipo de lógica ni de manejo de datos (solo se realizan importaciones), no son necesarios ni un modelo ni un controlador.

En el cabezal (sección *head*) del archivo BasePage.cshtml se realizan las importaciones de archivos de estilos CSS mediante la asociación estática del *view rendering Headlibs*. El mismo, en su campo *path* hace referencia a otro archivo cshtml que es el que contiene las importaciones propiamente dichas.

De forma análoga pero dentro del cuerpo del HTML del archivo BasePage.cshtml se realiza la importación de *scripts*, esta vez mediante el *view rendering Footlibs*.

En la siguiente figura (Figura 6) se muestra la relación entre el archivo correspondiente a *BasePage*, los *view renderings Headlibs* y *Footlibs* y sus correspondientes archivos cshtml:

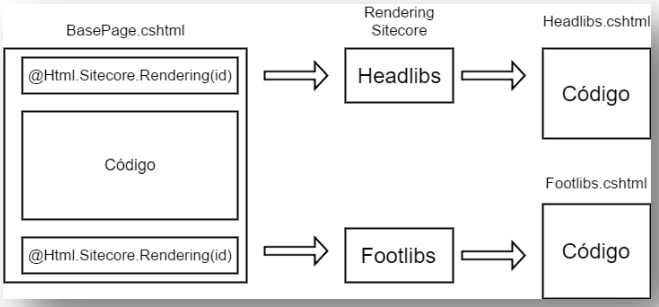


Figura 6 – Relación de archivos que involucran a los *rendering Headlibs* y *Footlibs*. Fuente: Autoría Propia

4.2.2 Topnav

Como se había descrito al final del desarrollo del *Basepage*, *TopNav* muestra enlaces (nombres) a todas las páginas hijas de primer nivel que no tengan el *checkbox* activado de mantener oculto en el *TopNav* (Figura 7). Por defecto dicho *checkbox* está desactivado. El procesamiento de las páginas hijas que serán visibles en el *TopNav* requiere de algunas operaciones lógicas, por lo tanto es necesario el uso de un controlador que las realice.

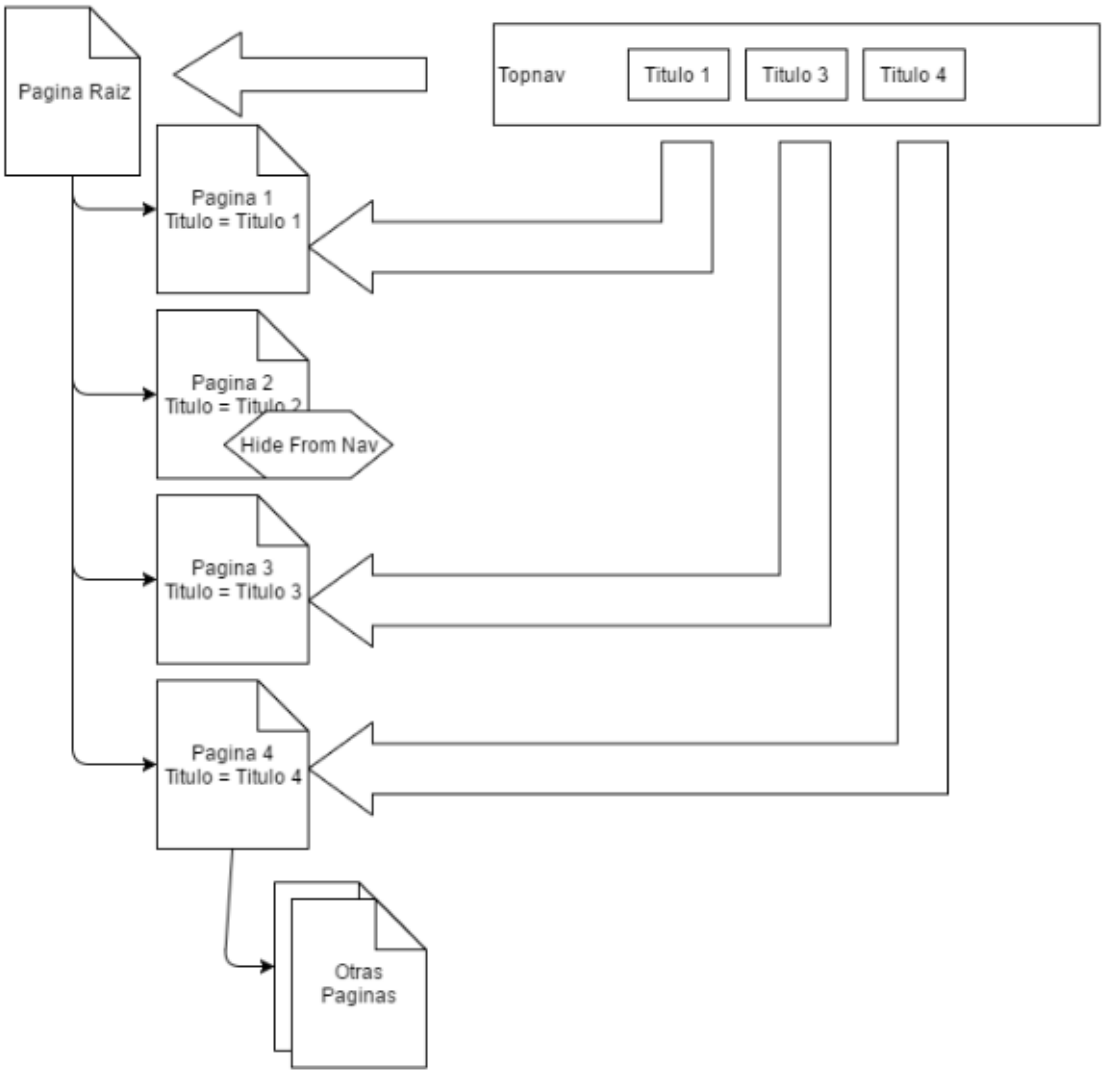


Figura 7 – Representación de páginas en TopNav dependiendo de activación de *checkbox*. Fuente: Autoría Propia

En consecuencia, el desarrollo se hace a partir de un *controller rendering*, referenciando al controlador y método ambos de nombre *TopNav* en el proyecto en *Visual Studio*. Este tipo de *rendering* no permite la asociación de un modelo en Sitecore, por lo tanto el mismo será instanciado a través del controlador.
Para la parametrización se utilizó un ítem como fuente de datos. El ítem en cuestión es el llamado *TopNavDS*. El mismo es el que contiene los campos de información del *TopNav* y fue creado a partir del *template* llamado *TopNavData* (Figura 8).
La utilización de un ítem como fuente de datos busca mantener una misma configuración para todas las instancias del *Topnav* que se creen a través del sitio web.
Se obtiene como resultado la centralización de toda la configuración en el ítem *TopNavDS*.

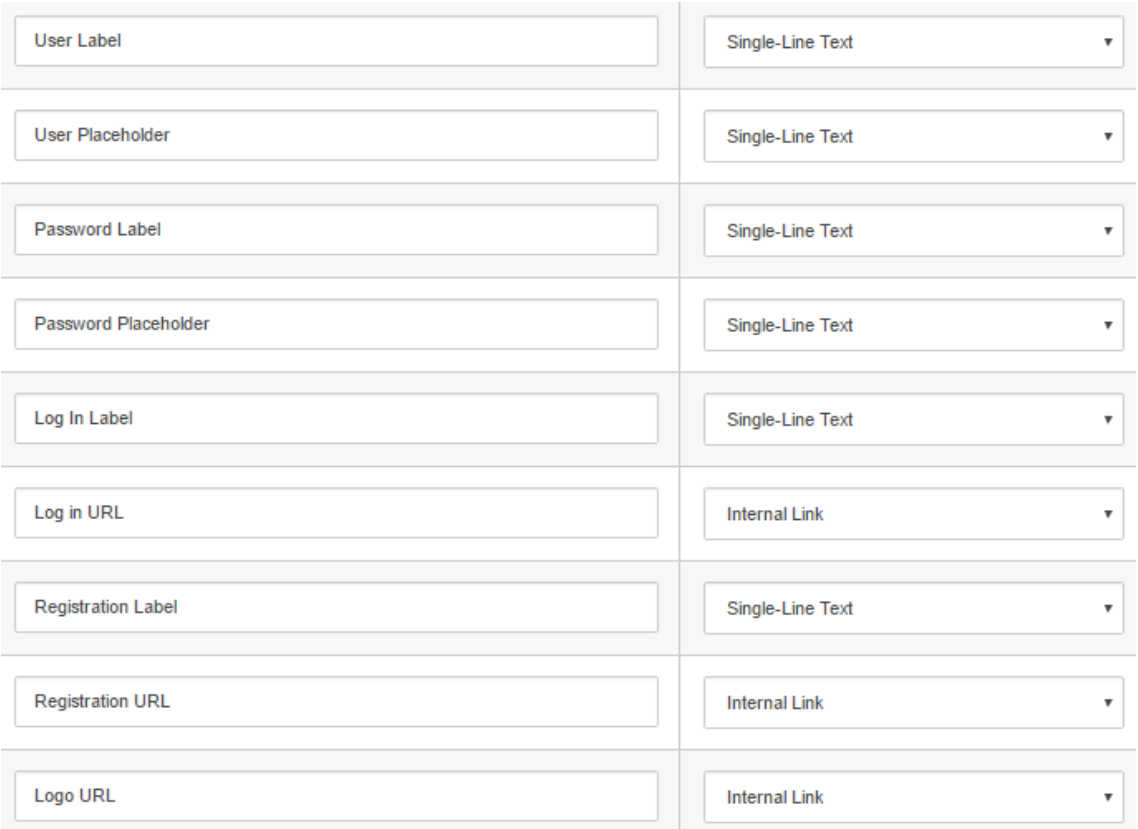


Figura 8 – Campos de información de *template* *TopNavData*. Fuente: Autoría Propia

Los campos de este *template* corresponden a los visibles en el *TopNav* en primera instancia, así como los que se muestran al desplegar el cuadro de *login*, que se describirá más adelante. Este componente es desarrollado de esta forma para no tener que configurar parámetros para cada nueva instancia del *TopNav*, que aparece en la parte superior de todas las páginas del sitio web y debe mostrar siempre la misma información. De esta forma con modificar y establecer los campos del ítem *TopNavDS*, para cada Instancia del *TopNav* se cargarán los datos del mismo.

El componente también contiene un diálogo desplegable que es el encargado de realizar el inicio de sesión, llamado *Login*.

4.2.2.1 Log in

Brinda la posibilidad a los usuarios del sitio web de poder identificarse o también ser re dirigidos a la página para registrarse (componente *Register*). Como vemos en la Figura 9, el *log-in box* permite al usuario ingresar su nombre de usuario y contraseña, al hacer clic en el botón de *Log-In* se comunica utilizando AJAX con una clase especifica que creamos para el manejo de usuarios.

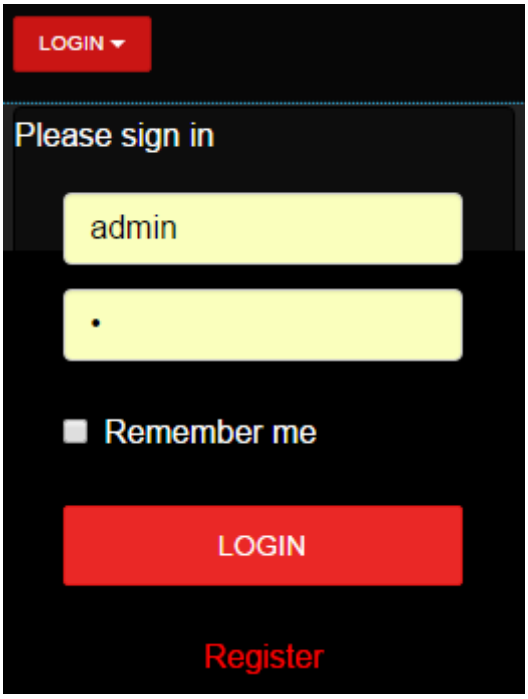


Figura 9 – Log-In box. Fuente: Autoría Propia

4.2.3 Footer

Este componente no requiere de la implementación de lógica de negocio, por lo tanto es implementado por un *View Rendering,* el mismo utiliza una vista y un modelo.
En las propiedades del *rendering* debe hacerse referencia al modelo utilizado, como se muestra en la siguiente figura (Figura 10) en la propiedad *Model*:

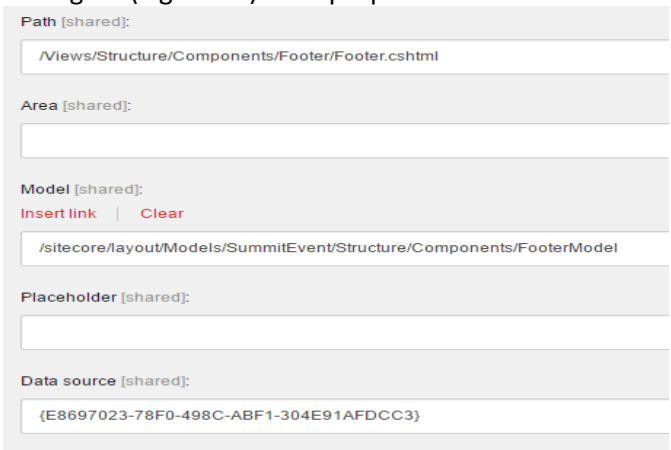


Figura 10 – Referencia a modelo en Sitecore. Fuente: Autoría Propia

El modelo en Sitecore solamente hace referencia a la clase que va a ser utilizada como modelo dentro del proyecto de *Visual Studio*.
Para la parametrización se utilizó un ítem como fuente de datos. El ítem en cuestión es el llamado *FooterDS*. El mismo es el que contiene los campos de información del *Footer* y fue creado a partir del *template* llamada *FooterData*.
La única particularidad de este componente es que uno de sus campos de información es de tipo *multilist* y su ruta corresponde a la carpeta que contiene los ítems *Static Links*. De esta manera se pueden agregar, quitar, editar los *links* disponibles desde el editor de contenido y

así modificar la configuración del ítem *FooterDS* y por tanto la presentación de todas las instancias del *rendering Footer*.

4.2.3.1 Static Links

Estos ítems no tienen ningún archivo u elemento relacionado con código.
Solamente es necesario usar el editor de contenido de Sitecore para crearlos. Son ítems basados en el *template Static Link* el cual define dos campos:

- URL de tipo *Path*.
- *Text* de tipo texto.

Como se muestra en la figura (Figura 11), los ítems son almacenados en *sitecore/content/Summit/GenericData/StaticLinks.*

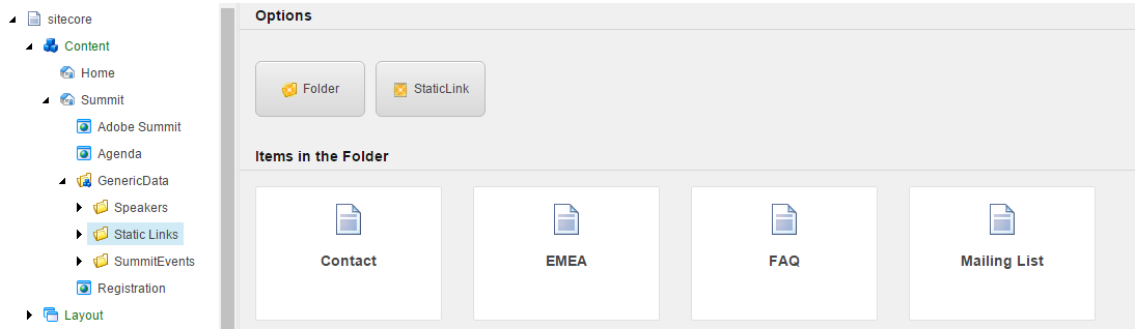


Figura 11 – Carpeta conteniendo los ítems static links. Fuente: Autoría Propia

Para su uso se debe proporcionar en Sitecore su ubicación, mediante un campo de tipo *multilist*, como fue explicado previamente en el desarrollo del *Footer*.

4.3 Componentes de contenido

A partir de tener la estructura del sitio web se pasa a construir los componentes directamente relacionados al contenido.

4.4 Banner

Este componente solamente requiere un campo correspondiente a una imagen. Sin embargo, el procesamiento de la imagen requiere de ciertas operaciones que provee la API de Sitecore, por lo que amerita el uso de un controlador.
Como resultado se utiliza un *controller rendering* el cual es parametrizado con un *parameter template*, de esta forma las instancias son independientes en lo que respecta al valor del campo imagen.
No se utiliza un modelo ya este componente maneja una cantidad reducida de campos.

4.5 Jumbo

El componente Jumbo fue implementado de dos maneras distintas.
En primera instancia se desarrolló utilizando un *Controller Rendering* el cual utiliza una vista, un modelo y un controlador, dado que aunque el *rendering* no implementa lógica de negocio si debe procesar un conjunto considerable de campos.
Para la parametrización se utiliza un *parameter template*, que es referenciado en las propiedades del *rendering*. El mismo tiene los siguientes campos (Figura 12):

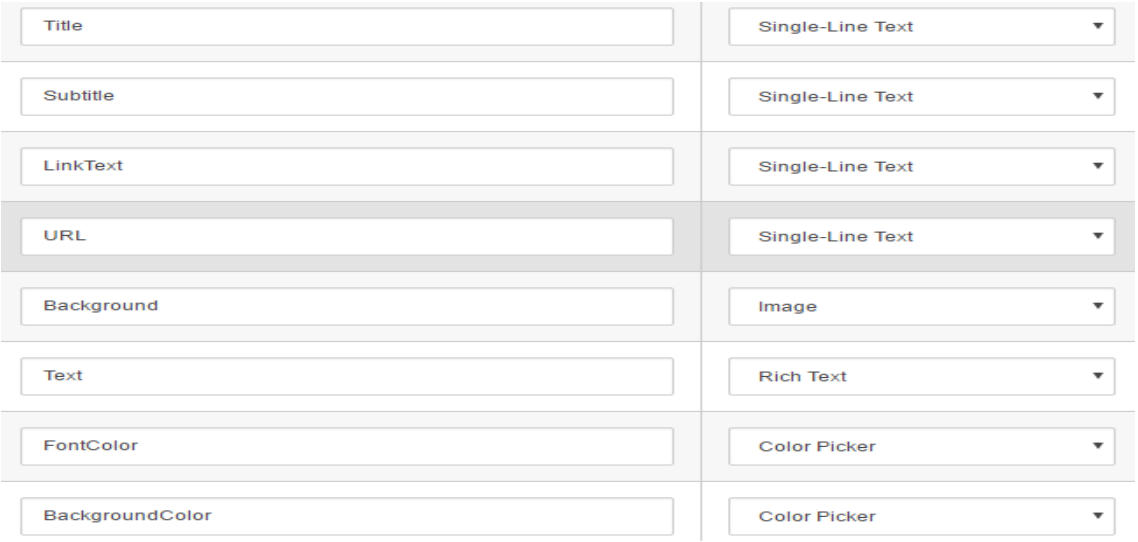


Figura 12 – Campos de componente *Jumbo*. Fuente: Autoría Propia

La segunda alternativa surgió a partir de la interacción con un arquitecto de software con experiencia en Sitecore. El mismo nos sugirió investigar el *framework* Glass.Mapper [5]. Éste permite mapear los valores de los campos asociados a los ítems de Sitecore a una clase C# mediante notaciones sencillas y sin tener que crear código para obtener los valores de los campos. A partir de esto decidimos incluir al menos un *rendering* utilizando dicho *framework* a pesar de que la implementación del caso de estudio en Sitecore ya había sido finalizada.

La utilización del *framework* permite disminuir la cantidad de archivos que deben ser desarrollados eliminando así el controlador, por este motivo esta segunda implementación del componente fue realizada con un *View Rendering*, el cual ni siquiera fue necesario asociarle un modelo, ya que el *framework* se encarga de inicializarlo y mapearlo. De esta forma se reduce de manera significativa la complejidad y cantidad de código que debe ser generado para desarrollar un *rendering* con el mismo comportamiento.

4.6 Registration

En su parte lógica, utiliza la API de seguridad proporcionada por Sitecore, la cual permite entre otras funciones, la creación de usuarios. Por lo tanto es necesario disponer de un controlador que realice dichas operaciones y por ende se realiza a través de un *controller rendering* desde Sitecore.
Para este componente la información mostrada debe ser siempre la misma a través de todo el sitio. Entonces se utiliza como estrategia de parametrización la de utilizar un ítem como fuente de datos. El ítem en cuestión es el llamado *RegistrationDS*, qué está creado a partir del *template RegistrationItem* que se muestra en la figura siguiente (Figura 13):

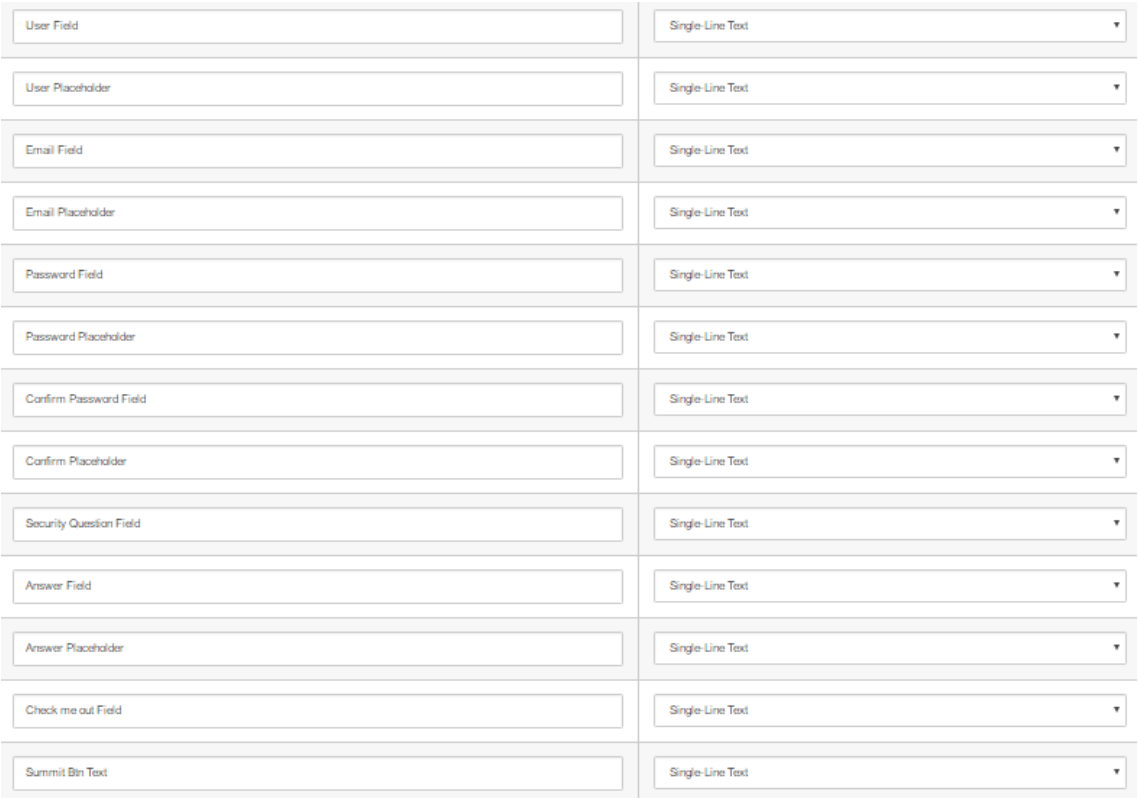


Figura 13 – Campos de *template RegistrationItem*. Fuente: Autoría Propia

El ítem *RegistrationDS*, es referenciado a través de su ID en las propiedades del *rendering*.
El controlador asociado a este componente realiza dos operaciones, el inicio de sesión y el registro de usuario. En caso de registro, se redirige a otra página que se encarga de realizar dicha tarea. Ambas funcionalidades están disponibles a través de un cuadro desplegable ubicado en el *TopNav* (componente *Login*).
Para realizar cambios en el ítem en cuestión, los mismos deben hacerse desde el editor de contenido.

4.7 Speaker

Similar al ítem *Static Link* con el *rendering Footer*, los ítems creados a partir del *template Speaker* son utilizados por los *renderings Speaker Box* y *Calendar*. Sitecore en lo que respecta al manejo de lista de datos asociada a los *renderings*, no ofrece otra opción que no sea obtener los datos –ítems- de la lista ya creados por los autores, esto quiere decir que no es posible para los autores crear un nuevo valor desde un dialogo (como si lo ofrece AEM), sino que deben ser creados de forma anticipada. Esto nos obliga a crear todos los *speakers* que deben ser usados en una carpeta y referenciarlos desde un campo *multilist*.

Los campos asociados al *template* son una imagen que representa la foto del *speaker* y dos textos que refieren a su nombre y profesión.

4.8 Speaker box

El *rendering* utiliza un *template parameter* para manejar los datos de sus campos ya que el contenido de las instancias debe ser independiente entre sí. El *template parameter* define dos campos, uno de tipo *color picker* para manejar el color de fondo y otro de tipo *multilist* el cual hace referencia a la carpeta donde los ítems *Speaker* son creados.

Debido a que la lógica involucrada en este componente es menor, es implementado mediante un v*iew rendering*. A su vez utiliza un modelo el cual es el encargado de obtener los datos de sus campos.

Este componente tiene la particularidad de poder agregar la cantidad (hasta cuatro) y en el orden deseado de *speakers* disponibles y se pueden realizar cambios a los mismos en línea (editar información y cambiarle la imagen) por parte del autor, quedando estos cambios de forma permanente, esto es posible gracias a que los *speakers* son representados por ítems. En cada lugar que se utilice (haga referencia a un ítem speaker), los cambios se verán reflejados ya que todas hacen referencia a un mismo ítem.
Este componente aplica de forma combinada las dos formas de realizar parametrización, ya que usa como uno de sus campos parametrizables a los ítems, referenciándolos a través de una ruta que es la fuente de datos y también otro campo correspondiente al color de fondo.

4.9 Calendar

Para este componente, el más complejo que se realizó, se utilizan variadas tecnologías, como jQuery, AJAX, Javascript.
Nos basamos en un *framework* que nos fue brindado por nuestro cliente Conexio con ciertas funcionalidades de calendario, construido a partir del *framework* disponible en https://fullcalendar.io/ [16]. Nuestro trabajo se basó en adaptar su funcionamiento en Sitecore y desarrollar nuevas funcionalidades para poder dar de alta, baja o modificar eventos y reservas.
Se utiliza también la API proporcionada por Sitecore tanto para acceso a ítems, como para la creación, borrado y modificación de los mismos.

4.9.1 Parametrización de rendering

Cada instancia de calendario es única y funciona de forma independiente. Por lo tanto para la parametrización, se utiliza un *parameter template*.

Además dado la complejidad de este componente y el manejo de ítems que hay que realizar, es evidente la utilización de un controlador y por ende la de un *controller rendering* del lado de Sitecore.
También se utiliza un modelo, el cual es referenciado en las propiedades del *rendering*.

4.9.2 Items Utilizados por el Rendering

4.9.2.1 Eventos

4.9.2.1.1 Item Evento

Es definido por un *template* el cual contiene los campos mostrados en la imagen a continuación:

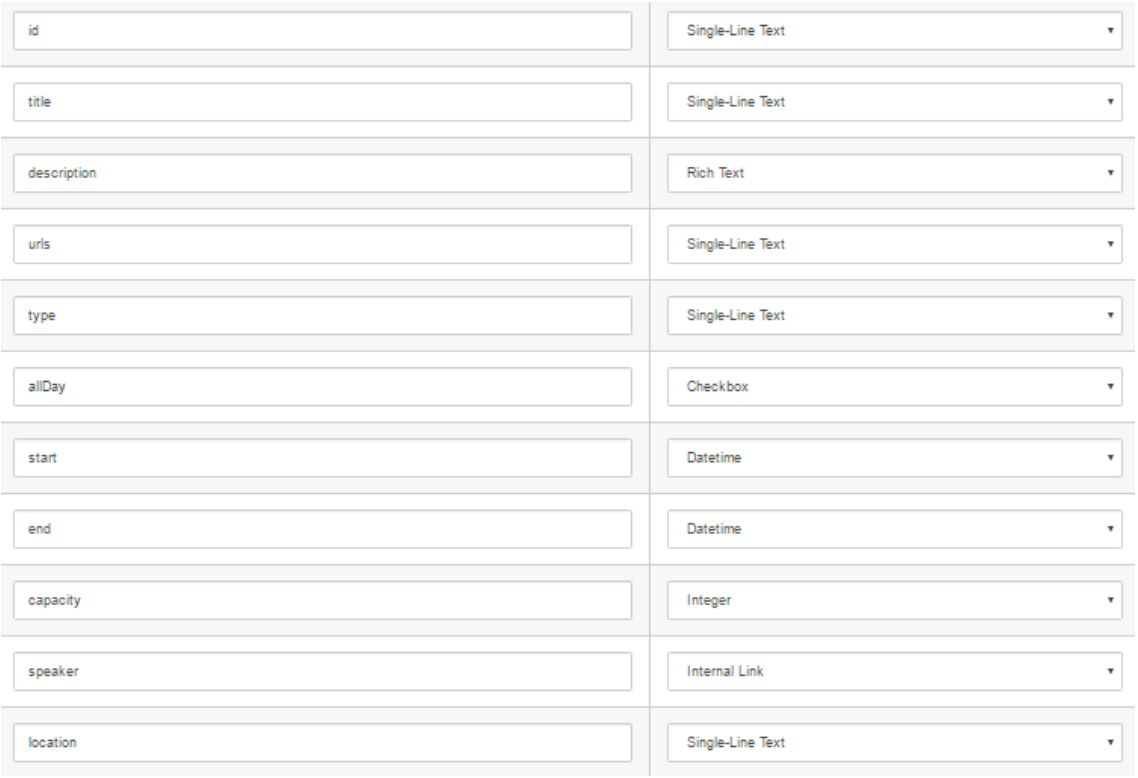


Figura 14 – Campos del *template* utilizado por ítem Evento. Fuente: Autoría Propia

4.9.2.1.2 Persistencia de eventos

La estrategia definida para persistir un evento, está basado en una estructura jerárquica de ítems. De esta forma se facilita la búsqueda de eventos o reservas por la ruta de los mismos.

Dadas las restricciones definidas a la hora de realizar la búsqueda de un ítem en Sitecore, la estrategia para generar la ruta completa de un evento va a estar dada por su fecha y su ID de la siguiente forma:

*sitecore/*carpeta definida por autor*/año del evento/mes del evento/día del evento/eventoID*

Donde el eventoID es un nombre único auto numerado por la lógica del controlador del *rendering*.

Esta estrategia nos permite saber exactamente donde está guardado un evento dado su fecha y su identificador, también nos permite obtener todos los eventos creados en una fecha determinada mediante la búsqueda por ruta que brinda Sitecore.

Para el caso de dar de baja un evento, se realiza un borrado del ítem correspondiente, a través de la API de Sitecore y mediante la ruta del mismo.

4.9.2.2 Reservas

4.9.2.2.1 Ítem Reserva

El ítem de reserva, llamado *Reservation*, solo contiene dos campos de texto correspondientes al nombre de usuario y su *mail*.

4.9.2.2.2 Persistencia de reservas

Se guardan como hijos del ítem correspondiente al evento de registro, con el nombre correspondiente al usuario que realiza la reserva. Estas decisiones:

- Facilitan las búsquedas, en especial para el caso de una baja, y en general para las búsquedas por ruta.
- Evitan el agregado de parámetros para la identificación del evento al cual pertenece la reserva.
- Evitan el duplicado de reservas para un mismo usuario.

Para el caso de dar de baja una reserva a un evento, se realiza el borrado del ítem correspondiente, a través de la API de Sitecore y mediante la ruta del ítem.
En la figura a continuación (Figura 15) se muestra donde hay un evento creado sin usuarios registrados al mismo, y otro que tiene un usuario registrado.

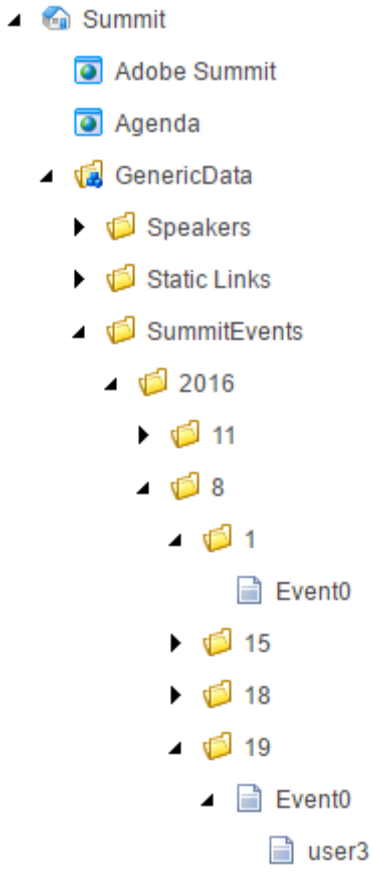


Figura 15 – Ítems representando eventos y reservas a los mismos. Fuente: Autoría Propia

4.9.2.3 Speaker

El autor puede configurar la ruta de donde el *rendering* busca a los ítems *Speaker* (Figura 16). Se decide que no se puedan editar en línea, para limitar que solo puedan ser modificados en un único *rendering* (*Speaker Box*).

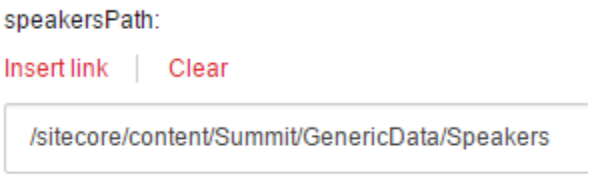


Figura 16 – Ruta referenciando la ubicación de ítems *speakers*. Fuente: Autoría Propia

Los *speakers* se asocian con los eventos mediante su ID. Para esto los eventos tienen un parámetro de tipo *internal link* el cual es posible asignarle el ID del *speaker*.

4.9.3 Renderizado

Similar a lo que sucede con AEM, uno de los principales problemas a la hora de implementar el componente *Calendar* surge a partir de la posibilidad de ofrecer un manejo simple a los autores de los eventos que contiene el mismo.

Si bien Sitecore ofrece la posibilidad de crear ítems por fuera de un diálogo y asignar sus valores de una forma simple y a diferencia de AEM sin la necesidad que pertenezcan a una página, creemos que la mejor forma de administrar los eventos es mediante la utilización del propio calendario. Por ese motivo, se utilizaron las tres vistas descritas anteriormente en la implementación de AEM para ofrecer la administración de los eventos a los autores.

5 Creación de contenido

Por último y habiendo implementado los componente definidos anteriormente, pasamos a crear las páginas del caso de estudio e incluir los componentes requeridos en cada una. La misma va a estar constituida por seis páginas, como vemos en la Figura 17, primero tenemos el nodo referente a nuestro sitio –*Summit*-. Cuenta con tres páginas hijas, *Adobe Summit, Agenda* y *Registration.* A su vez existe una carpeta llamada *GenericData* donde son persistidos los ítems de los *speakers*, *static links* y eventos de calendario.
Para la página *Registration* se define *Hide in Navigation* en *true,* para que no esté disponible desde el *TopNav*.

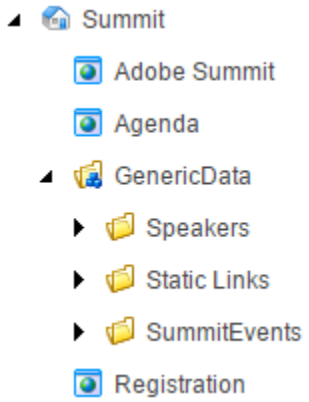


Figura 17 – Estructura de sitio web *Summit*. Fuente: Autoría Propia

6 Solución móvil

6.1.1.1 Xamarin

Xamarin [17] es una *cross platform* basada en .NET, lo cual significa que programando en C# se pueden implementar aplicaciones móviles nativas tanto para *Android*, *iOS* y *Windows Phone*.

En abril del año 2015 Sitecore anuncia la asociación con Xamarin con el objetivo de expandir su oferta en desarrollo móvil, de esta forma Sitecore ofrece una API -*Sitecore MobileSDK Xamarin*- para poder comunicarse con sus servidores desde aplicaciones creadas utilizando *Xamarin*.

Para la solución móvil no fue necesario el desarrollaro de nuevos componentes, ya que la API ofrecida para trabajar con *Xamarin* no maneja *renderings,* sino que es utilizada para obtener ítems y sus datos. Se desarrolló un proyecto para la familia de sistemas operativos *Android*. No se realizó para otros sistemas operativos (*iOS*, *Windows Phone*) ya que no se disponía de dispositivos con esos sistemas operativos (necesario para el caso de *iOS*).

Se hizo uso de la API antes mencionada en su versión 1.3.0, la cual permite una gran variedad de funciones en lo que respecta a ítems. Las más importantes para llevar a cabo este proyecto fueron en particular la lectura, creación y borrado de ítems (también imágenes para el caso de la lectura). La aplicación está compuesta por tres actividades de *Android* que son representadas mediantes clases de C# (ver Figura 3):

6.1.1.2 Funcionalidades

6.1.1.2.1 Inicio de sesión

Es la actividad inicial de la aplicación móvil -Figura 1-. Se permite el acceso, búsqueda y registro/baja de eventos a los usuarios registrados en el sitio web. La misma se comunica con el servidor para validar la combinación de usuario y contraseña ingresada.

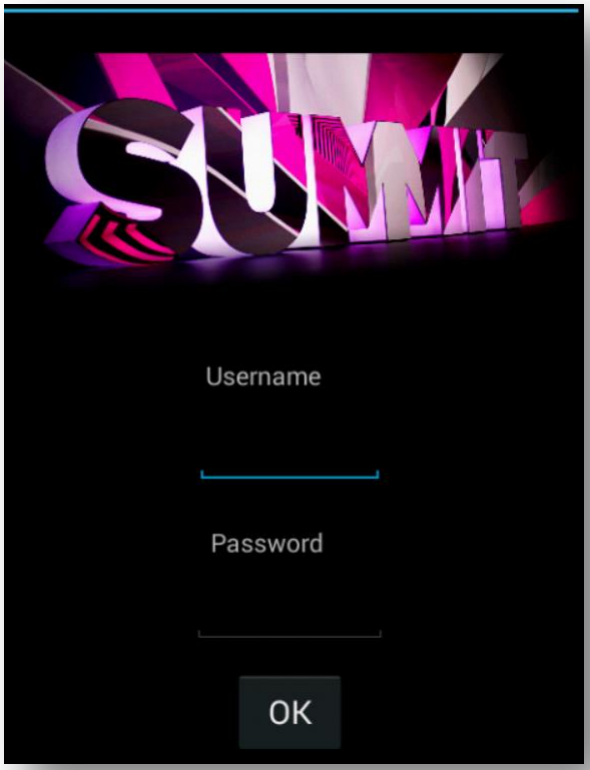


Figura 1 - Captura de pantalla de la aplicación móvil desarrollada. Fuente: Autoría propia.

6.1.1.2.2 Búsqueda de eventos por fecha

Una vez logueado el usuario es enviado a la actividad de búsqueda de eventos -Figura -, la cual provee de un calendario que filtra por fecha los eventos y los despliega por nombre en orden alfabético.

6.1.1.2.3 Inscripción/baja de evento

Esta actividad se despliega al seleccionar un evento en el calendario descrito anteriormente y muestra la información más relevante del mismo -Figura 2-. Se ofrece la opción de darse de baja o registrarse, dependiendo del caso. Utilizando la API *Sitecore MobileSDK Xamarin* creamos un ítem de reserva desde la aplicación móvil en la instancia de Sitecore o eliminando la reserva en caso que el usuario quiera darse de baja.

En la Figura 3 se puede observar un resumen de la interacción de los usuarios con la aplicación y el servidor de Sitecore. Para que el usuario pueda llegar a realizar una reserva en un evento debe pasar por un conjunto de tres pasos secuenciales. El usuario comienza identificándose con un usuario y contraseña, el siguiente paso consiste en seleccionar una fecha para que la aplicación despliegue los eventos disponibles. Finalmente el usuario selecciona uno y realiza o elimina una reserva.

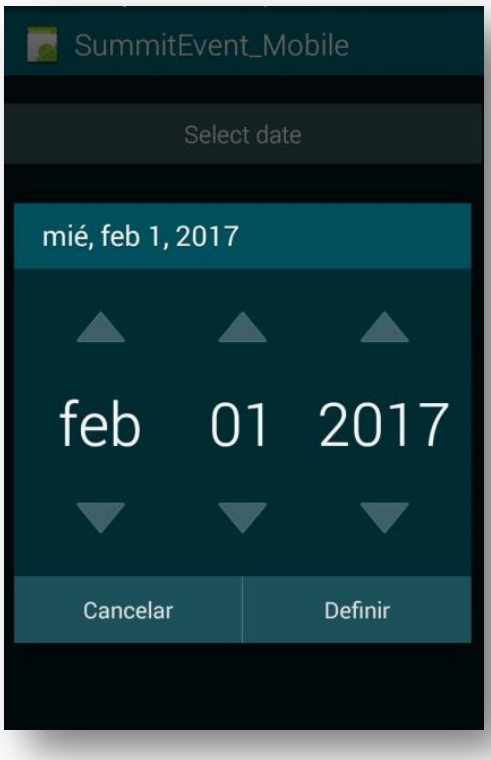


Figura 19 - Captura de pantalla de la aplicación móvil desarrollada, calendario para búsqueda de ventos. Fuente: Autoría propia.

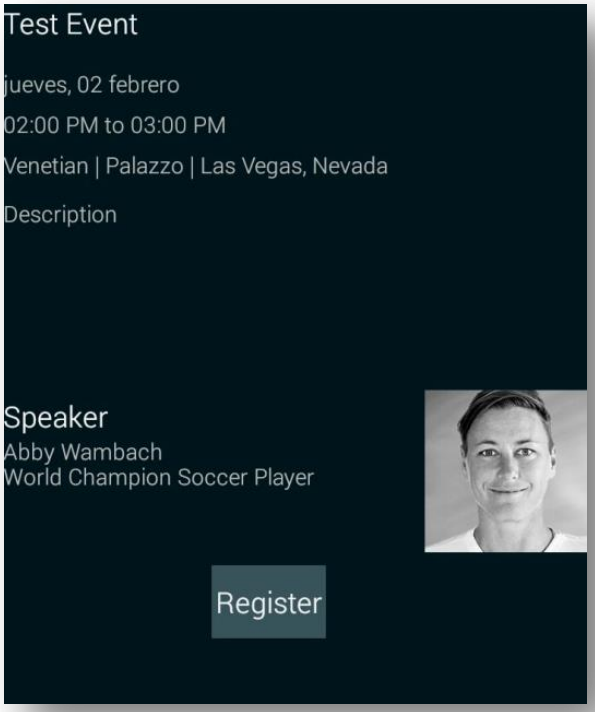


Figura 2 - Captura de pantalla de la aplicación móvil desarrollada. Fuente: Autoría propia.

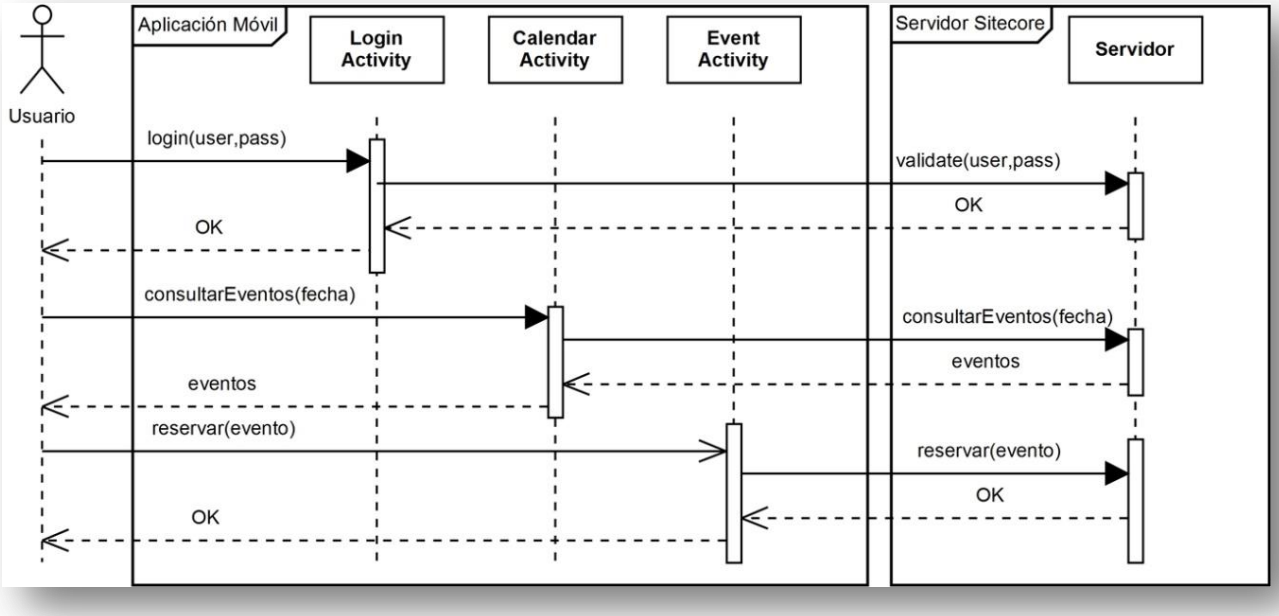


Figura 3 – Diagrama de secuencia de la comunicación de los usuarios con la aplicación y ésta con el servidor. Fuente: Autoría Propia

7 Referencias

[1] http://logview4net.com/. Último acceso: 17/10/2016.
[2] https://docs.angularjs.org/guide/introduction. Último acceso: 17/10/2016.
[3] http://getbootstrap.com/. Último acceso: 17/10/2016.
[4] https://fullcalendar.io/. Último acceso: 17/10/2016.
[5] http://www.glass.lu/. Último acceso: 17/10/2016.
[6] http://vsplugins.sitecore.net/. Último acceso: 17/10/2016.
[7] https://es.atlassian.com/software/bitbucket. Último acceso: 17/10/2016.
[8] http://www.teamdevelopmentforsitecore.com/. Último acceso: 5/2/2017.
[9] http://docs.sitecoreship.apiary.io/ Último acceso: 01/02/2017.
[10] https://github.com/adoprog/Sitecore-Courier Último acceso: 01/02/2017.
[11] https://github.com/kamsar/Unicorn. Último acceso: 01/02/2017.
[12] David Flanagan. *Javascript. The definite guide.* O'Reilly Media, Inc. 2011.
[13] http://www.w3schools.com/css/. Último acceso: 17/10/2016.
[14] https://jquery.com/. Último acceso: 17/10/2016.
[15] http://www.w3schools.com/asp/razor_intro.asp. Último acceso: 17/10/2016.
[16] https://fullcalendar.io/. Última visita: 23/2/2017.
[17] https://www.xamarin.com/ Último acceso: 01/02/2017.

Anexo 6 Actas de reuniones

Índice

Introducción

A continuación se presentarán las actas correspondientes a todas las reuniones mantenidas en el marco de este proyecto de grado, las cuales incluyen reuniones con nuestro tutor y cliente de Conexio , con un docente del instituto de computación –INCO- de la Facultad de Ingeniería así como una demostración de Sitecore hecha a una empresa estadounidense que ya trabaja con dicha plataforma.

Actas

20/04/2016

20/04/2016
Duración 45 min:

Integrantes de la reunión:
Jorge Corral
Martin Santagata
Federico Mujica
Pablo Larrosa

Descripción de la reunión:

Se habló sobre el punteo tentativo del estado del arte, los avances del estudio técnico de Sitecore. Pablo participó brindando su punto de vista sobre qué aspectos son importantes en el desarrollo de proyecto en plataformas WCMS:

- Mantener la fidelidad del contenido e información para los diferentes canales de publicación.
- Desarrollo simple en diferentes tipos de dispositivo manteniendo la centralización de la información.
- Brindar la información de las empresas de forma fácil, instantánea y en el momento adecuado y al usuario adecuado. Ejemplificar este punto hablando sobre el diferente contenido que muestra una página web si acceden personas de diferente ubicación geográfica.

Objetivos para próxima reunión miércoles 4/5/2016:

1. Tener una primera versión de la planificación general del Proyecto.
2. Tener una versión 0.1 de tabla comparativa de WCMS con diferentes criterios de fuentes secundarias.
3. Creación de un prototipo básico de Sitecore (TopNav, HomePage, navegación básica).
4. Avanzar aproximadamente 10 páginas en el estado del arte.
5. Comenzar a definir un documento de criterios para el diseño del caso de uso para comparar Sitecore y AEM.

04/05/2016

04/05/2016
Duración 60 min:

Integrantes de la reunión:
Jorge Corral
Martin Santagata
Federico Mujica
Pablo Larrosa

Se cumplieron con los primeros 4 objetivos de la reunión del miércoles 20/04/2016, no hubo mucho avance en el punto 5, en el caso del punto 3 no se llegó a crear un TopNav o generar navegación básica:

1. Tener una primera versión de la planificación general del Proyecto.
2. Tener una versión 0.1 de tabla comparativa de WCMS con diferentes criterios de fuentes secundarias.
3. Creación de una prototipo básico de Sitecore (TopNav,HomePage,Navegacion Basica).
4. Avanzar aproximadamente 10 páginas en el estado del arte.
5. Comenzar a definir un documento de criterios para el diseño del caso de uso para comparar Sitecore y AEM.

Descripción de la reunión:
Se realizó una demo sobre los principios básicos de Sitecore y cómo desarrollar componentes de baja complejidad. Se discutió cuáles serían los pasos a futuro sobre la plataforma Sitecore y que se debería comenzar a investigar, se habló de comenzar a documentar cómo se desarrolla en Sitecore.
Se sugirió buscar blogs de empresas *partners* que trabajan con Sitecore para obtener información.

Objetivos para próxima reunión miércoles 18/5/2016:

1. Seguir avanzando con el prototipo básico de Sitecore (TopNav, HomePage, Navegación Básica).
2. Avanzar en el estado del arte.
3. Crear la documentación técnica de cómo utilizar la plataforma Sitecore (Pueden existir comparaciones con AEM dentro del documento).
4. Comenzar a definir un documento de criterios para el diseño del caso de uso para comparar Sitecore y AEM.

18/05/2016

18/05/2016
Duración 45 min:

Integrantes de la reunión:
Jorge Corral
Martin Santagata
Federico Mujica
Pablo Larrosa

Descripción de la reunión:

Se realizó una demo de lo avanzado en Sitecore, básicamente como generar un *template* de página que tome por defecto un *layout*.
También se mostró como generar un componente a partir de un controlador y utilizar MCV.

Objetivos para próxima reunión miércoles 1/6/2016:

1. Poder crear un ambiente de desarrollo con alguna herramienta colaborativa como Git, SVN, etc.
2. Avanzar con la documentación técnica de Sitecore.
3. Hacer un sitio básico con algunos componentes para que el autor pueda modificarlos.
4. Seguir probando *templates* que vienen *out of the box* en Sitecore.
5. Investigar la exportación de componentes a otras instancias de Sitecore.

1/06/2016

1/06/2016
Duración 45 min:

Integrantes de la reunión:
Jorge Corral
Martin Santagata
Federico Mujica

Descripción de la reunión:

- Se comentó a grandes rasgos cómo se logró realizar un ambiente colaborativo para Sitecore, cabe destacar la dificultad enfrentada en este tema, se investigaron diferentes herramientas utilizadas para encapsular un proyecto en Sitecore y podes desplegar en cualquier instancia, básicamente se deben extraer los ítems que se desean exportar de la base de datos y agregarlos como archivos al proyecto.
- Se realizó una demo de lo avanzado en Sitecore, en particular el agregado de componentes y manejo básico a nivel de autor.

Objetivos para próxima reunión miércoles 15/06/2016:

- Realizar una iteración más sobre lo ya realizado en la parte web.
- Avanzar con la documentación técnica de Sitecore.
- Investigar importación de bibliotecas (Bootstrap, jQuery, etc).
- Realizar comparación entre AEM y Sitecore.
- Investigar diálogos en Sitecore.
- Realizar una primera iteración en desarrollo móvil en Sitecore, investigando distintas maneras de poder hacerlo.

15/06/2016

15/06/2016
Duración 45 min:

Integrantes de la reunión:
Pablo Larrosa
Federico Mujica
Martin Santagata

Descripción de la Reunión:

- Se realizó una demo de lo avanzado en Sitecore, en particular sobre diálogos en Sitecore.
- Se realizó una demo sobre lo avanzado en Sitecore móvil, utilizando Xamarin y consumiendo datos de Sitecore y mostrándose en una aplicación móvil.
- Se plantearon lineamientos para el comienzo del desarrollo del caso de estudio, el mismo va a consistir en una aplicación de tipo agenda de eventos con reserva de salones e inscripción de participantes, tanto para web como para móvil.

Objetivos para próxima reunión miércoles 29/06/2016:

- Avanzar sobre lo que va a ser el caso de estudio, de acuerdo a lo hablado.
- Avanzar en la documentación de Sitecore.
- Avanzar en el estado del arte.
- Crear un documento con los requerimientos del caso de estudio para que sean evaluados por Pablo y Jorge.

06/07/2016

06/07/2016
Duración 45 min:

Integrantes de la reunión:
Jorge Corral
Martin Santagata
Federico Mujica
Pablo Larosa

Descripción de la reunión:

Se realizó una puesta a punto de lo avanzado en el proyecto, se definió basar el *look and feel* del caso de estudio en la web http://summit.adobe.com/na/. Se realizó una muestra de los 2 repositorios creados, uno para AEM y otro para Sitecore. Se hizo un repaso de lo avanzado en la documentación y la creación de un anexo para el estudio de Sitecore.

Objetivos para próxima reunión miércoles 1/06/2016:

1. Avanzar en el desarrollo del caso de estudio.
2. Documentar los requerimientos del caso de estudio.
3. Avanzar con la documentación de desarrollo sobre Sitecore.
4. Analizar qué metodologías de desarrollo y de testing se usaría para el desarrollo de proyectos CMS.
5. Estudiar alternativas Mobile para trabajar con AEM y Sitecore.

21/07/2016

21/07/2016
Duración 45 min:

Integrantes de la reunión:
Jorge Corral
Martin Santagata
Federico Mujica

Descripción de la reunión:

Se realizó una puesta a punto de lo avanzado en la implementación en Sitecore del Caso de Estudio, se implementaron 5 componentes, se avanzó en la definición de requerimientos del caso de estudio dividido por componentes. Se definió poder tener una versión casi completa de la implementación del caso de estudio para fines de agosto en Sitecore.

Objetivos para próxima reunión miércoles 03/08/2016:

1. Priorizar el avanzar en la implementación de los componentes del Caso de Estudio más complejos que quedan, como el registro de usuarios y *log in*.
2. Como segunda prioridad avanzar en la documentación técnica de Sitecore.
3. Por último analizar si la división en componentes de los requerimientos del caso de estudio es válido, tanto para AEM como para Sitecore.

03/08/2016

03/08/2016
Duración 45 min:

Integrantes de la reunión:
Jorge Corral
Federico Mujica

Descripción de la reunión:

Se le mostró a Jorge las implementaciones hechas en Sitecore, las 4 diferentes formas de crear un componente (MVC-MV-CV-V) y las diferentes formas que tiene Sitecore de administrar usuarios. Se realizó una demo de la implementación que se hizo para Sitecore para el *log in* y el registro de usuarios en la web. Se acordó realizar una demo a un cliente de EEUU a fines de agosto para obtener un *feedback* de cómo se está utilizando Sitecore.

Objetivos para próxima reunión miércoles 18/08/2016:

1. Priorizar el avanzar en la implementación de los componentes del Caso de Estudio más complejos que quedan, como el registro de eventos.
2. Pulir el código.

17/08/2016

17/08/2016
Duración 45 min:

Integrantes de la reunión:
Jorge Corral
Pablo Larrosa
Federico Mujica
Martín Santagata

Descripción de Reunión:

Se le mostró a Jorge y a Pablo las implementaciones hechas en Sitecore, las 4 diferentes formas de crear un componente (MVC-MV-CV-V) y las diferentes formas que tiene Sitecore de administrar usuarios. Se realizó una demo de la implementación que se hizo para Sitecore para el *log in* y el registro de usuarios en la web. Se acordó realizar una demo a un cliente de EEUU a el día 24 de agosto para obtener un *feedback* de cómo se está utilizando Sitecore y de los avances realizados en el mismo.

Objetivos para próxima Reunión Miércoles 26/08/2016:

1. Terminar de realizar los componentes vistos para la presentación del día 26 de agosto.
2. Realizar una pequeña presentación introductoria para enviar al cliente antes de la realización de la demo.

26/08/2016

26/08/2016
Duración 45 min:

Integrantes de la reunión:
Jorge Corral
Pablo Larrosa
Federico Mujica
Martín Santagata
4 Integrantes de Empresa EEUU

Descripción de Demo:

Se realizó una demo del sitio web creado con Sitecore a 4 integrantes de una empresa con base en EEUU que trabaja con dicha tecnología, durante 25 minutos se mostraron los diferentes componentes desarrollados. Luego el referente técnico realizó diversas preguntas de la implementación, la metodología y las herramientas utilizadas durante 20 minutos aproximadamente. La devolución fue muy satisfactoria, se recomendó el uso de ciertos *frameworks* que generalmente se utilizan en la metodología de desarrollo de Sitecore como Glass.Mapper, hubo reconocimiento del gran avance mostrado y quedaron conformes con el manejo de la plataforma alcanzado, cabe destacar que dicha empresa fue la que nos brindó la licencia de Sitecore para aprender a desarrollar en ella.
Se abrieron las puertas para futuras colaboraciones y proyectos.

La próxima reunión está pendiente de confirmar ya que Jorge se va de Viaje:

1. Avanzar en la documentación de Sitecore.
2. Investigar e integrar el framework Glass.Mapper.
3. Crear una aplicación móvil para que se conecte al proyecto realizado.

14/09/2016

14/09/2016
Duración 45 min:

Integrantes de la reunión:
Jorge Corral
Pablo Larrosa
Federico Mujica
Martín Santagata

Descripción de la reunión:

Se le mostró a Jorge y a Pablo el prototipo móvil hecho para Android, que se conecta a la instancia de Sitecore a través de una API que permite crear, actualizar, leer y borrar ítems, autenticar sesiones entre otras funciones.
Se decidió terminar con el desarrollo en Sitecore con lo que respecta a la parte web y móvil, e investigar sobre el tema *Analytics*, así como la documentación a realizar.

Objetivos para próxima Reunión Miércoles 26/08/2016:

1. Investigar *Analytics* tanto en Sitecore como en AEM.
2. Realizar en AEM lo mismo que fue desarrollado en Sitecore (parte web y móvil).
3. Documentar el caso de estudio (diseño, arquitectura, etc.).
4. Realizar análisis de las etapas de desarrollo aprendidas a lo largo de la carrera con la metodología utilizada para trabajar con WCMSs.

28/09/2016

28/09/2016
Duración 45 min:

Integrantes de la reunión:
Jorge Corral
Pablo Larrosa
Federico Mujica
Martín Santagata

Descripción de Reunión:

Se le puso al tanto a Jorge y a Pablo sobre los avances de las pasadas dos semanas. Se está terminando el desarrollo web del caso de estudio en AEM, faltando la parte mobile por comenzar.
Con respecto a *Analytics* en Sitecore, no hubo muchos avances, pero por lo visto para sacarle provecho habría que probar con el sitio web publicado en un servidor.
En cuanto a la documentación, comentamos lo que tenemos hecho hasta el momento en cuanto al diseño y requerimientos del caso de estudio y de la solución en Sitecore. Jorge y Pablo nos hicieron algunas recomendaciones al respecto, en particular sobre en qué puntos enfocarnos en cuanto a la comparación de metodologías tradicionales de desarrollo en comparación con las de desarrollo en CMS (requerimientos, gestión del proyecto, *deploy* y *testing*). También se nos sugirió realizar un esqueleto de lo que sería la estructura del informe final.

Objetivos para próxima Reunión Miércoles 12/10/2016:

1. Investigar un poco más *Analytics* tanto en Sitecore como en AEM.
2. Continuar con el desarrollo en AEM del caso de estudio.
3. Continuar con la documentación y comenzar el análisis de la metodología tradicional de desarrollo y de la de desarrollo en WCMS.
4. Realizar esqueleto de lo que será el informe final.

12/10/2016

12/10/2016
Duración 60 min:

Integrantes de la reunión:
Jorge Corral
Federico Mujica
Martín Santagata

Descripción de Reunión:

Se le puso al tanto a Jorge sobre los avances de las pasadas dos semanas. Se terminó el desarrollo web del caso de estudio en AEM, faltando la parte móvil por comenzar.
Con respecto a *Analytics* en Sitecore, no hubo avances significativos que sean de provecho, y por lo tanto se decidió dejar el tema para trabajo futuro y realizar documentación teórica sobre el tema.
En cuanto a la documentación, comentamos lo que tenemos hecho hasta el momento en cuanto al diseño y requerimientos del caso de estudio y de la solución en Sitecore, también comentamos sobre la estructura de lo que sería el informe final.
También se acordó en tener una entrevista con un docente del departamento de ingeniería de software de la facultad, a fin de que podamos obtener sugerencias y otros puntos de vista sobre el tema de metodologías de desarrollo.
Jorge nos hizo algunas recomendaciones al respecto de la documentación realizada y a realizar, dentro de los puntos más importantes se destacan:

- Para el estudio de metodologías de desarrollo, primero estudiar desarrollo en metodologías ágiles (en particular Scrum), luego sobre desarrollo en WCMS y luego realizar un análisis comparando los puntos anteriores.
- Justificar elección de puntos a comparar.
- Documentar por qué se realizó el caso de estudio que finalmente se llevó a cabo.
- Documentar en capítulos distintos para Sitecore y AEM.
- Documentar el esfuerzo requerido para la comprensión y trabajo en Sitecore.
- Realizar introducción sobre Sitecore y AEM y luego dar paso a las soluciones en ambas plataformas.
- Documentar problemas encontrados, alternativas disponibles y decisiones tomadas.

Objetivos para próxima Reunión Miércoles 27/10/2016:

1. Continuar con el desarrollo móvil en AEM del caso de estudio.
2. Continuar con la documentación, aplicando las sugerencias recibidas.

28/10/2016

28/10/2016
Duración 45 minutos:

Integrantes de la reunión:
Jorge Corral

Federico Mujica
Martín Santagata

Descripción de la reunión:

Se le puso al tanto a Jorge sobre los avances de las pasadas dos semanas. Se terminó el desarrollo web del caso de estudio en AEM, faltando la parte móvil por comenzar.
Jorge nos hizo una sugerencia en cuanto al orden de los capítulos de forma que el análisis de las metodologías vaya luego del caso de estudio y solución del mismo. Además nos aconsejó sobre el largo y el detalle del informe, en particular para la parte de la solución del caso de estudio.
También se sugirió la incorporación al proyecto de algún artefacto que demuestre la planificación del proyecto, así como el cálculo de esfuerzo.

Objetivos para próxima Reunión Miércoles 9/11/2016:

1. Continuar con el desarrollo móvil en AEM del caso de estudio.
2. Continuar con la documentación, aplicando las sugerencias recibidas.

7/11/2016

7/11/2016
Duración 45 minutos:

Integrantes de la reunión:
Sebastián Pizard
Federico Mujica
Martín Santagata

Descripción de Reunión:

Se lo puso al tanto brevemente de qué tratan las tecnologías WCMS, el proyecto que estamos desarrollando y el enfoque del mismo.
Se nos hizo algunas sugerencias, como:

- Buscar más fuentes (publicaciones científicas) en el portal Timbó.
- Realizar algún diagrama de BPM o los utilizados en la materia PSP de la facultad para describir las metodologías.
- Documentar el proceso de facto para CMS, así como el de Conexio y realizar comparación.

9/11/2016

9/11/2016
Duración 45 minutos:

Integrantes de la reunión:
Jorge Corral
Federico Mujica

Martín Santagata

Descripción de la reunión:

Se le puso al tanto a Jorge sobre los avances de las pasadas dos semanas, en particular sobre la parte metodológica que formará parte del informe.
Jorge nos hizo una sugerencia en cuanto a las búsquedas realizadas de bibliografía relacionada con metodologías de desarrollo en WCMSs, que ampliemos las búsquedas realizadas así como explicitar todas las búsquedas realizadas y los pocos resultados obtenidos.
También se nos sugirió que se extraigan las etapas en común de los procesos de desarrollo llevados a cabo en Conexio y concluir a partir de eso cuál sería el mejor según la empresa.
También se sugirió la incorporación al proyecto de algún artefacto que demuestre la planificación del proyecto, así como el cálculo de esfuerzo.
También se sugirió la realización con algún especialista en el tema de QA y requerimientos.

Objetivos para próxima Reunión Miércoles 23/11/2016:

1. Continuar con el desarrollo móvil en AEM del caso de estudio.
2. Continuar con la documentación.
3. Aplicar sugerencias hechas para la parte metodológica de desarrollo con CMS.

23/11/2016

23/11/2016
Duración 45 minutos:

Integrantes de la reunión:
Jorge Corral
Federico Mujica
Martín Santagata

Descripción de Reunión:

Se le puso al tanto a Jorge sobre los avances de las pasadas dos semanas, en particular sobre la parte metodológica que formará parte del informe. Se mostró la estructura básica del informe y anexos, y que contiene cada parte a grandes rasgos.
Se le mostró a Jorge documentación realizada a través de la extracción de las etapas en común de los procesos de desarrollo llevados a cabo en Conexio combinado con la experiencia nuestra.
También se sugirió la incorporación al proyecto de algún artefacto que demuestre la planificación del proyecto, así como el cálculo de esfuerzo.

Objetivos para próxima Reunión Miércoles 7/12/2016:

1. Continuar con el desarrollo móvil en AEM del caso de estudio.
2. Continuar con la documentación.

10/12/2016

10/12/2016
Duración 60 minutos:

Integrantes de la reunión:
Jorge Corral
Federico Mujica
Martín Santagata

Descripción de la reunión:

Se le mostró a Jorge una primera versión del informe final. Se nos hizo numerosas observaciones y correcciones sobre detalles en la escritura y formato del mismo.
Jorge se quedó con la copia del informe entregada y lo seguirá viendo a fin de aportar una nueva devolución sobre el mismo.

Objetivos para próxima Reunión Miércoles 22/12/2016:

1. Corregir y mejorar la documentación de acuerdo a las correcciones y observaciones recibidas.

25/01/2017

25/01/2017

Duración 90 min:

Integrantes de la reunión:
Jorge Corral
Martin Santagata
Federico Mujica

Descripción de Reunión:

- Jorge realizó una devolución del informe realizado, mostrando y explicando todas las correcciones hechas.

Objetivos para próxima (fecha a determinar):

- Realizar las correcciones hechas en el informe para una nueva revisión.

01/03/2017

01/03/2017
Duración 45min:

Integrantes de la reunión:
Jorge Corral
Martin Santagata

Federico Mujica

Descripción de la reunión:

- Jorge realizó una devolución del informe realizado, mostrando y explicando todas las correcciones hechas. Se discutió principalmente los capítulos de conclusiones, trabajo futuro y dificultades afrontadas en el proyecto.

Objetivos para próxima (fecha a determinar):

- Realizar las correcciones hechas en el informe para una nueva revisión.