



Instituto de Computación - Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Observatorio Tecnológico para el Desarrollo de Aplicaciones Web

Informe de Proyecto de Grado

Diciembre 2018

Docentes:

MSc. Ing. Laura González
Ing. Gustavo Guimerans

Autores:

Fabiana Andrade Acosta
Nicolás Martínez Bastón

Resumen

Las tecnologías que dan soporte al desarrollo de aplicaciones web han evolucionado notoriamente en las últimas décadas. Esto por un lado facilita en muchos aspectos las actividades de desarrollo, pero también genera desafíos al momento de seleccionar las tecnologías más adecuadas para un determinado contexto y al momento de decidir cómo utilizarlas para aprovechar su máximo potencial. En este contexto, es de interés contar con una plataforma que brinde asistencia tanto en la selección como en la utilización de tecnologías de acuerdo a los requerimientos para un proyecto específico.

Este proyecto propone una solución del tipo Observatorio Tecnológico que permite, para el área de desarrollo web, categorizar las tecnologías existentes, monitorear su evolución, brindar buenas prácticas y una guía para el uso de las mismas.

En primer lugar se analiza la problemática planteada y los conceptos que se encuentran relacionados a ella, tales como conceptualización de arquitecturas de aplicaciones web y observatorios tecnológicos. Se analizan las diferentes características que deben cumplir los observatorios tecnológicos y se relevan los requerimientos funcionales y no funcionales a ser considerados en la solución.

En segundo lugar, se propone una solución de Observatorio Tecnológico orientado al desarrollo de aplicaciones web. Se presentan las características principales y sus funcionalidades que surgen del análisis y el relevamiento de requerimientos. Además, se realiza una instanciación de la solución propuesta a modo de realizar una primer prueba de concepto, proporcionando diferentes escenarios de arquitecturas de aplicaciones web, y cómo la solución propuesta provee soporte a las mismas.

Luego, se realiza una implementación de un prototipo de Observatorio Tecnológico en base a la solución propuesta que cubre las funcionalidades que se entendieron más relevantes (por ejemplo acceso a la información de las diferentes tecnologías pudiendo obtener una guía para el uso de las mismas). El prototipo se implementó mediante la plataforma Java Empresarial y la herramienta React, mientras que para el despliegue se utilizaron los servicios de Amazon Web Services.

Finalmente se desarrolla un caso de estudio para una empresa de desarrollo de software que, ante un nuevo requerimiento de un cliente, decide registrarse en el Observatorio. Se detallan los principales requerimientos y cómo el Observatorio los fue acompañando a lo largo del ciclo de vida del proyecto.

Palabras clave: Observatorio Tecnológico, Aplicaciones Web, Integración Continua.

Tabla de contenido

1. Introducción	7
1.1. Contexto y motivación	7
1.2. Objetivos	7
1.3. Aportes del Proyecto	8
1.4. Organización del Documento	8
2. Marco conceptual	10
2.1. Arquitectura cliente-servidor	10
2.2. Aplicaciones web	10
2.2.1. Características de una Aplicación Web	10
2.2.2. Arquitectura de aplicaciones web	11
2.2.3. Evolución histórica de las aplicaciones web	12
2.2.4. Capa de presentación de aplicaciones web	13
2.3. Desarrollo de aplicaciones web modernas	15
2.4. Soporte tecnológico para el desarrollo de aplicaciones web	17
2.4.1. Framework de desarrollo	17
2.4.2. Repositorio de código	17
2.4.3. Git	17
2.4.4. GitLab	17
2.4.5. Integración Continua	17
2.4.6. Dependencia de software	18
2.4.7. Versionado de software	18
2.4.8. Gestor de proyectos	18
2.4.9. Docker	19
2.5. Recuperación de Información	19
2.5.1. XML, XSD, XSLT	19
2.5.2. Canales RSS	20
3. Análisis de la problemática	21
3.1. Arquitectura de aplicaciones web	21
3.1.1. Especificación de niveles	21
3.1.2. Componentes de cada capa	23
3.1.3. Capa Transversal	25
3.1.4. Herramientas de desarrollo y administración	26
3.2. Observatorios Tecnológicos	27
3.2.1. Funciones principales	27
3.2.2. Criterios de selección de observatorios tecnológicos existentes	28
3.2.3. Observatorios Tecnológicos seleccionados	29
3.2.4. Comparación de Observatorios Tecnológicos	32
3.3. Relevamiento de requerimientos	32
3.3.1. Descripción del problema	32

3.3.2.	Actores principales	33
3.3.3.	Requerimientos funcionales	33
3.3.4.	Requerimientos no funcionales	34
3.4.	Resumen y conclusiones	34
4.	Solución propuesta	36
4.1.	Descripción general	36
4.2.	Características principales	38
4.2.1.	Espacios de trabajo	38
4.2.2.	Ciclo de vida de las principales entidades	38
4.2.3.	Compatibilidades entre tecnologías	40
4.2.4.	Integración con sistemas externos	41
4.2.5.	Reportes	41
4.2.6.	Usuario, roles y organizaciones	41
4.2.7.	Sistema de puntos	43
4.3.	Modelo conceptual	43
4.3.1.	Gestión de tecnologías	45
4.3.2.	Gestión de AGPDAW y <i>stacks</i>	46
4.3.3.	Gestión de recursos	47
4.3.4.	Definición de formatos y versiones de tecnologías	48
4.3.5.	Gestión de usuarios, roles y permisos	49
4.4.	Funcionalidades	51
4.4.1.	Funcionalidades exclusivas de <i>BackOffice</i>	51
4.4.2.	Funcionalidades de <i>FrontOffice</i> y <i>BackOffice</i>	51
4.4.3.	Resumen de Funcionalidades	54
4.5.	Arquitectura Lógica del Observatorio Tecnológico	55
4.5.1.	Vista Lógica	55
4.5.2.	Componentes de la vista lógica del Observatorio Tecnológico	56
5.	Instanciación	60
5.1.	Ecosistema Java	60
5.1.1.	Web Project Template	60
5.1.2.	Tecnologías	61
5.2.	Conclusiones	63
6.	Implementación	64
6.1.	Descripción general	64
6.2.	Servidor backend Java EE	68
6.2.1.	Capa de negocio	68
6.2.2.	Capa de servicios externos	69
6.2.3.	Capa de acceso a datos	72
6.3.	Capa de persistencia	73
6.4.	Capa de presentación frontend	73

6.5.	Implementación de funcionalidades	73
6.5.1.	Alta de tecnologías principales	73
6.5.2.	Alta de versiones de tecnologías	73
6.5.3.	Recursos	74
6.5.4.	Alta de tipos de proyecto	74
6.5.5.	Ejecución de pruebas automáticas sobre recursos	74
6.5.6.	Alta de componentes y Web Project Templates	74
6.5.7.	Wizard	75
6.5.8.	Reportes	76
6.5.9.	Panel de noticias	76
6.6.	Despliegue	76
6.6.1.	Configuración del despliegue	77
6.6.2.	Puesta en producción	78
6.7.	Decisiones tomadas	79
6.7.1.	Tecnologías	79
6.7.2.	Servicios de repositorios de código	80
6.7.3.	Servicios de integración continua	80
6.7.4.	Tipos de proyecto soportados	81
6.7.5.	Recuperación de noticias	81
7.	Caso de estudio	82
7.1.	Descripción del problema	82
7.2.	Funcionalidades relevantes para la empresa	84
7.3.	Primeros pasos con el Observatorio	87
7.4.	Primeras conclusiones	88
7.5.	Ejecución del Wizard	89
7.6.	Aporte a la comunidad	93
7.7.	Nueva problemática	93
7.8.	Nuevo entorno de ejecución	97
7.9.	Conclusión del caso de estudio	98
8.	Conclusiones y trabajo a futuro	99
8.1.	Resumen y conclusiones	99
8.2.	Trabajo a futuro	101
8.2.1.	Funcionalidades del prototipo	101
8.2.2.	Nuevas integraciones con plataformas y servicios	101
8.2.3.	Mejoras y complementos	102
8.2.4.	Aspectos técnicos	104
	Referencias	105

Apéndice A. Instanciación	111
A.1. Ecosistema Python	111
A.1.1. Web Project Template y Stack	111
A.1.2. Tecnologías	113
A.2. Ecosistema .NET	115
A.2.1. Web Project Template	115
A.2.2. Tecnologías	116
Apéndice B. Integración con canales RSS	118
Apéndice C. Configuración de <i>pipelines</i>	121

1. Introducción

En este documento se presenta el proyecto de grado “Observatorio Tecnológico para el Desarrollo de Aplicaciones Web”. En este capítulo se detalla el objetivo del proyecto, así como también los aportes del mismo. Finalmente se brinda una descripción de la organización del resto del documento.

1.1. Contexto y motivación

Durante los últimos años, el soporte tecnológico orientado al desarrollo de aplicaciones ha experimentado una constante evolución, donde año tras año se desarrollan nuevas soluciones que introducen nuevos paradigmas de implementación de aplicaciones web. Esto ha generado que las distintas organizaciones y personas especializadas en el desarrollo de aplicaciones web requieran de una constante actualización tecnológica, presentando el recurrente problema de selección de tecnologías para nuevos proyectos de desarrollo.

Esta constante evolución y actualización presenta otro tipo de problemas, como puede ser la necesidad de mantener actualizadas las tecnologías que fueron incorporadas en las aplicaciones web implementadas hace algún tiempo. De esta forma resulta de interés contar con una plataforma capaz de brindar soporte al ciclo de vida de las aplicaciones web, desde la selección de tecnologías, implementación, puesta en producción y mantenimiento.

1.2. Objetivos

El objetivo general de este proyecto consiste en el diseño y construcción de un Observatorio Tecnológico para el desarrollo de aplicaciones web, que brinde soluciones para facilitar la selección de tecnologías y promueva una utilización adecuada de las mismas. Para cumplir con el objetivo general del proyecto, se plantean los siguientes objetivos específicos:

- Analizar requerimientos y realizar un análisis del contexto donde se sitúa el proyecto y la realidad planteada, en particular, observatorios tecnológicos orientados al desarrollo de aplicaciones web.
- Realizar un relevamiento de soluciones ya existentes con características similares.
- Proponer una solución que permita gestionar, categorizar, observar la evolución y brindar buenas prácticas para el uso de diferentes tecnologías aplicadas al desarrollo de aplicaciones web.
- Implementar un prototipo de Observatorio Tecnológico que brinde solución a la propuesta planteada para evaluar la factibilidad técnica.
- Desarrollar un caso de estudio que permita mostrar la aplicabilidad de la solución propuesta a la realidad planteada.

1.3. Aportes del Proyecto

Los principales aportes del proyecto son los que se describen a continuación:

- **Análisis de requerimientos para un Observatorio Tecnológico orientado al desarrollo de aplicaciones web**

Se realizó un relevamiento de requerimientos en base al análisis del contexto de la problemática planteada.

- **Relevamiento de soluciones ya existentes con características similares**

Se analizaron soluciones existentes, seleccionándose las más relevantes en base a un criterio de selección establecido.

- **Propuesta de solución que aborda la realidad planteada**

Se propuso una solución tecnológica que permite gestionar, categorizar, observar la evolución y brindar buenas prácticas para el uso de diferentes tecnologías aplicadas al desarrollo de aplicaciones web, acompañando el ciclo de vida de las diferentes tecnologías y proveyendo de herramientas que sirvan como guía a la hora de comenzar nuevos proyectos.

- **Implementación de un prototipo de Observatorio Tecnológico**

Se implementó un prototipo de Observatorio Tecnológico que permitió validar la factibilidad técnica de un sistema de estas características.

- **Desarrollo de un caso de estudio**

Se presentó un caso de estudio que permitió mostrar la aplicabilidad del Observatorio a la realidad de una empresa dedicada al desarrollo de software.

1.4. Organización del Documento

El documento se organiza en siete capítulos:

- En el Capítulo 2 se presenta el marco conceptual donde se describen los conceptos que son relevantes en el contexto del proyecto.
- En el Capítulo 3 se realiza un análisis de la problemática, presentando las características principales de las aplicaciones web, junto con un análisis de soluciones de observatorios tecnológicos existentes. Finalmente se presenta un relevamiento de requerimientos.
- En el Capítulo 4 se presenta la solución propuesta. En particular, se detalla la arquitectura y se describen los aspectos más relevantes de la solución.

- En el Capítulo 5 se definen escenarios de arquitecturas de aplicaciones web y se instancian los componentes más relevantes de la solución propuesta, obteniendo una primera validación práctica de la solución.
- En el Capítulo 6 se presenta el prototipo de Observatorio Tecnológico implementado, detallando las tecnologías, herramientas utilizadas, junto con las integraciones a otros sistemas que fueron implementadas.
- En el Capítulo 7 se desarrolla un caso de estudio, presentando una realidad donde se ejemplifica el uso del Observatorio Tecnológico.
- En el Capítulo 8 se presentan la conclusiones del proyecto y el trabajo identificado para desarrollar en un futuro.

2. Marco conceptual

En este capítulo se presentan los conceptos más relevantes para el proyecto que serán utilizados a lo largo este documento.

2.1. Arquitectura cliente-servidor

La arquitectura cliente-servidor es una arquitectura de software en la que muchos clientes (procesadores remotos) solicitan y reciben servicios de un servidor centralizado. Las computadoras cliente proporcionan una interfaz para permitir que los usuarios soliciten servicios del servidor y muestre los resultados, mientras que los servidores reciben las solicitudes entrantes de los clientes y brindan una respuesta. Idealmente, un servidor proporciona una interfaz transparente estandarizada a los clientes para que no deban conocer las características específicas del sistema, tales como el hardware y el software que proporciona el servicio. Los clientes suelen estar ubicados en computadoras personales o teléfonos móviles, mientras que los servidores están ubicados en otros lugares de la red, generalmente en máquinas más potentes [1].

2.2. Aplicaciones web

Una aplicación es un software diseñado con el fin de cumplir tareas específicas para los usuarios. La idea principal es que facilite tareas a los diferentes usuarios, ya sea dando apoyo a nivel de negocio como puede ser en una empresa, hasta apoyar a actividades cotidianas, como podrían ser de educación o personales [2].

Una aplicación web es un software basado en la arquitectura cliente-servidor, tal que utilizando el protocolo HTTP [3], usualmente a través de un navegador web, se accede a los recursos expuestos por un servidor web [2].

2.2.1. Características de una Aplicación Web

Si bien las aplicaciones web se pueden clasificar de distintas formas, existen un conjunto de características principales comunes a la mayoría [4] [5]:

- Administración de usuarios, autenticación, autorización, control de acceso, manejo de roles
- Manejo de sesiones
- Navegación entre páginas
- Listados de información y contenidos, típicamente extraídos de una base de datos
- Actualización, extracción y almacenamiento de datos en bases de datos

- Administración de archivos, ya sea para almacenamiento o consumo
- Integraciones con sistemas externos

2.2.2. Arquitectura de aplicaciones web

El objetivo de una arquitectura al momento de diseñar una aplicación web es minimizar la complejidad tanto de la construcción como del futuro mantenimiento, logrando separar las tareas en diferentes áreas a modo de delegar responsabilidades. Las principales consideraciones a tomar en cuenta al momento de diseñar una arquitectura se resumen en [6]:

- **Particionar lógicamente la aplicación**

Sin una correcta separación de capas lógicas, se puede alcanzar una aplicación de gran porte con dificultades en términos de mantenimiento. Dividiendo la aplicación en capas de presentación, negocio y acceso a datos se obtiene una primera división por capas, simplificando tanto el desarrollo como el mantenimiento de la aplicación. De esta forma, se permite desarrollar y depurar cada capa por separado, ofreciendo mayor escalabilidad de la aplicación.

- **Implementar mecanismos de abstracción para desacoplar las capas**

Un alto acoplamiento entre capas puede provocar que cambios mínimos en una de ellas repercuta en las subyacentes, lo cual afecta el mantenimiento de la aplicación. Al definir interfaces que se encarguen de traducir solicitudes entre capas, se facilita el mantenimiento de la misma al lograr desacoplar las capas de la lógica que consumen de las subyacentes.

- **Considerar el almacenamiento en *caché* para minimizar los tiempos de respuesta**

Un servidor web debe poder escalar fácilmente en situaciones de carga crítica sin experimentar pérdida de solicitudes, ofreciendo a los usuarios tiempos de respuesta adecuados. Se debe considerar utilizar técnicas para el almacenamiento en *caché* [7] y colas de salida a modo de optimizar la comunicación tanto entre el navegador y el servidor web, como entre el servidor web y la base de datos.

- **Considerar técnicas de *logging***

Las aplicaciones web alojadas en la nube son vulnerables a ataques maliciosos. Es recomendable registrar todas las peticiones entrantes a la aplicación para poder detectar actividades sospechosas de posibles ataques al sistema.

- **Considerar la posibilidad de autenticar usuarios a través de límites de confianza:**

La mayoría de las aplicaciones tienen secciones más críticas que otras en términos de privacidad y seguridad de la información. Es deseable diseñar la aplicación con soporte de autenticación de usuarios con limitaciones de privilegios al cruzar límites de confianza, tales como alguna función de capa de negocio.

- **No transmitir datos confidenciales en texto plano a través de la red:**

Al transmitir datos confidenciales como contraseñas o claves de autenticación a través de la red en texto plano sin cifrar, se es vulnerable a que dicha información sea interceptada por usuarios maliciosos. Es imprescindible considerar la utilización de mecanismos de encriptación y firmas digitales, entre otros.

2.2.3. Evolución histórica de las aplicaciones web

Inicialmente, las aplicaciones web consistían básicamente en un conjunto de documentos estáticos accesibles desde Internet o una intranet, navegables entre sí a través de enlaces (hipervínculos). A medida que los lenguajes de programación fueron integrando el concepto de aplicaciones web a sus plataformas de desarrollo, los servidores web se dotaron con capacidades de ejecución de código, consultas a bases de datos y generación de páginas dinámicas. De esta forma, fue posible acceder a páginas web dinámicas, generadas en un servidor en base a alguna solicitud generada por el usuario [8].

Por otro lado, con los años y gracias al avance de los navegadores web, los mismos se fueron dotando de nuevas herramientas y tecnologías, alcanzando, por ejemplo, la posibilidad de ejecutar código JavaScript [9]. Esto introdujo a los programadores la posibilidad de agregar elementos dinámicos a la interfaz de usuario en el cliente mediante *scripts* programados. De esta forma, en lugar de enviar todos los datos al servidor para generar una página web completa, estos *scripts* embebidos en la página web podían realizar ciertas tareas en el cliente sin intervención del servidor: validación de entradas, mostrar u ocultar secciones de una página web, etc. [8]

Posteriormente se fueron introduciendo los conceptos de *Asynchronous JavaScript And XML* (AJAX) [10] y nuevas versiones de HTML [11] [12], dotando a las aplicaciones web de posibilidades de interacción con la información remota sin necesidad de recargar la página, mecanismos de renderización de gráficos 3D, bases de datos del lado del cliente para proveer soporte *offline* a las aplicaciones. Este tipo de herramientas dieron posibilidad a la construcción de las capas de presentación de aplicaciones web de gran porte que se suelen consumir en el día a día [8].

2.2.4. Capa de presentación de aplicaciones web

La capa de presentación de las aplicaciones web pueden clasificarse principalmente en dos patrones de arquitecturas generales [13]:

- Clientes finos
- Clientes gruesos

El patrón de arquitectura de cliente web fino es útil cuando sólo se puede garantizar una configuración de cliente mínima. Toda la lógica de extracción de datos y generación de interfaces gráficas se ejecuta en el servidor durante el cumplimiento de las solicitudes del usuario. De esta forma, se obtienen clientes que únicamente se limitan a desplegar documentos HTML y ejecutar *scripts* con simples validaciones de entradas y cambios en los componentes visuales. Este tipo de implementaciones de capa de presentación se denominan *Multiple Page Application* (MPA), ya que cada interacción del usuario requiere de enviar una nueva petición al servidor para que el mismo genere una nueva página a ser desplegada por el navegador, como se muestra en la Figura 1.

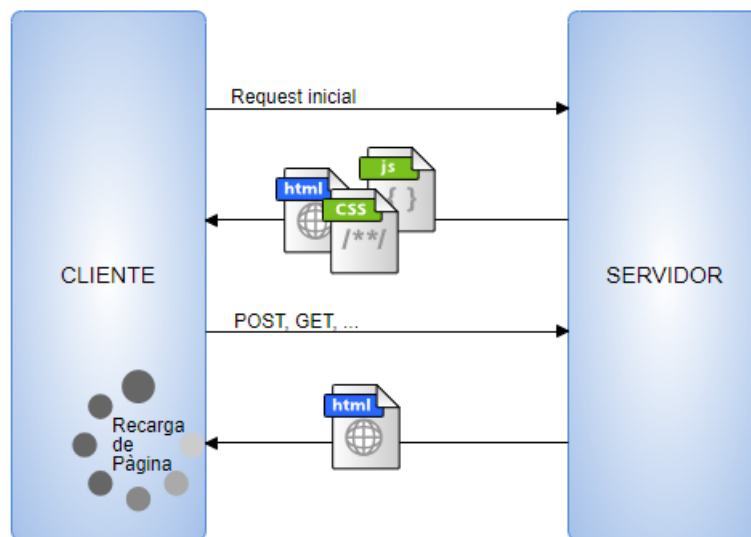


Figura 1: Multiple Page Application

Las ventajas de las MPA residen principalmente en que cualquier cliente, por poca capacidad de procesamiento que disponga, podrá ejecutar la aplicación, ya que todo el procesamiento pesado lo realiza un servidor, delegando al navegador únicamente la tarea de despliegue de documentos HTML. Otra ventaja es el manejo del *Search Engine Optimization* (SEO) [14] que proveen, mejorando los resultados de búsqueda de servicios como

Google [15] [16].

El patrón de arquitectura de cliente web grueso permite implementar capas de presentación que ofrecen experiencias comparables a las ofrecidas por las tradicionales aplicaciones de escritorio. Este tipo de cliente adquiere fuerza luego de la aparición del concepto de AJAX, el cual permite solicitar recursos desde un navegador a un servidor de forma asíncrona. De esta forma, es posible solicitar tanto documentos HTML como extraer datos de un servidor sin la necesidad de recargar la página, tal como se ilustra en la Figura 2. A partir de este concepto comenzaron a surgir las implementaciones de capa de presentación que se conocen como *Single Page Applications* (SPA).

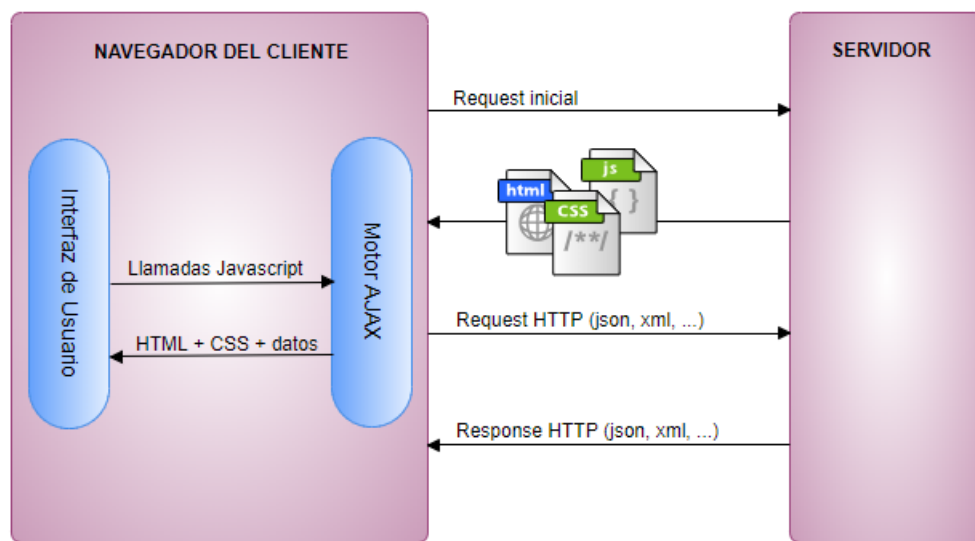


Figura 2: Simple Page Application

Las SPA tienen la particularidad de que, una vez obtenidas del servidor, se ejecutan por completo en el navegador del cliente y no requieren de recargar la página ante las interacciones del usuario, ya que, como su nombre lo indica, se componen de una única página web y código JavaScript encargado de solicitar todos los recursos necesarios (código HTML, CSS, datos) de forma asíncrona para el usuario mediante la técnica AJAX. Una vez obtenido el código JavaScript, el navegador es el encargado de ejecutarlo y desplegar nuevas secciones de la capa de presentación según lo requiera.

En la Figura 2 se muestra el funcionamiento de una SPA, pudiéndose observar cómo, luego de una solicitud inicial donde el servidor provee los recursos estáticos (HTML, CSS, código JavaScript), el mismo se desliga de la capa de presentación, delegando el trabajo de lógica de presentación y captura de entradas al navegador del cliente, limitándose única-

mente a atender peticiones entrantes a través de una capa de servicios [15].

Las ventajas de las SPA respecto a las tradicionales MPA son [15]:

- Las SPA proveen una interfaz más rápida para el usuario, ya que la mayoría de los recursos estáticos se descargan una sola vez en todo el ciclo de vida de la sesión.
- Los datos del servidor se extraen asíncronamente, de forma transparente para el usuario, y los cambios son reflejados en la aplicación de forma natural, sin la necesidad de recargar la página.
- Ofrecen posibilidades de trabajo *offline*, ya que los navegadores pueden almacenar localmente tanto los datos como los recursos estáticos necesarios.

Las desventajas que presentan son [15]:

- Pobre manejo del SEO, ya que las interfaces, al ser generadas del lado del cliente, dificulta a buscadores como Google en la tarea de extracción de la información.
- Tiempos de carga inicial de la aplicación superiores a las MPA, ya que se requiere cargar toda la aplicación al comienzo de la sesión. Es posible mitigar este problema mediante la utilización de técnicas de *lazy loading* [17], donde los recursos de la aplicación son extraídos asíncronamente de un servidor bajo demanda a medida que son requeridos.

2.3. Desarrollo de aplicaciones web modernas

El proceso de desarrollo de aplicaciones web modernas se compone principalmente de dos grandes áreas:

- Desarrollo *frontend*
- Desarrollo *backend*

El desarrollo *frontend* se refiere al proceso de análisis, diseño e implementación de las interfaces gráficas a través de las cuales los usuarios interactúan con la aplicación web. Esto incluye la estructuración de las secciones de la interfaz, tamaños, márgenes entre estructuras, tipos de letra, colores, adaptación para distintas pantallas, captura de entradas del usuario y efectos visuales, entre otros. [8]

El desarrollo *backend* abarca el análisis, diseño e implementación de la lógica de negocio que da soporte a las aplicaciones web. Se centra principalmente en la lógica operacional de una aplicación, con el fin de asegurar que las peticiones o servicios solicitados por una capa de presentación o aplicación externa sean atendidos correctamente con tiempos de respuesta adecuados. Esta tarea puede incluir desde consultas o inserciones en base de datos, hasta

el consumo de servicios externos, entre otros [18] [8].

Otra gran área que abarca el desarrollo *backend* consiste en todo lo que se refiere a integraciones con sistemas externos, como se muestra en la Figura 3. Las *Application Programming Interface* (API) [19] desempeñan un papel fundamental en la forma en que se construyen la mayoría de las arquitecturas de software del lado del servidor, a menudo consumiendo servicios externos para la realización de tareas específicas. De esta forma, un desarrollo *backend* se debe encargar tanto del consumo de APIs de sistemas externos, como de la creación de APIs para que sistemas externos logren comunicarse de forma eficiente con el sistema propio. [18]

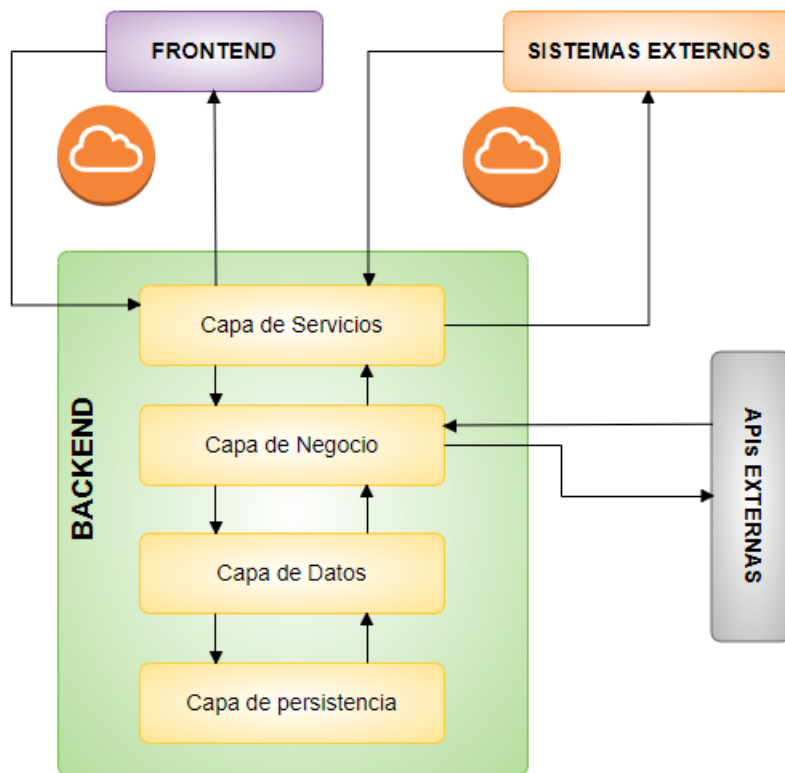


Figura 3: Arquitectura *Backend* de Aplicaciones Web

2.4. Soporte tecnológico para el desarrollo de aplicaciones web

En esta sección se describen las principales herramientas involucradas en los procesos de desarrollo de aplicaciones y que resultan de interés a los efectos de este proyecto.

2.4.1. Framework de desarrollo

Un *framework*, o plataforma de desarrollo, es una colección de paquetes y módulos desarrollados en un lenguaje determinado que permiten a los desarrolladores implementar aplicaciones sin la necesidad de lidiar con detalles de bajo nivel, tales como protocolos, *sockets*, hilos, procesos, entre otros [20].

2.4.2. Repositorio de código

Un repositorio de código es un servicio de almacenamiento de código fuente mediante un sistema de versionado. Un sistema de versionado es un software que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, a modo de posibilitar la recuperación de versiones anteriores en el futuro [21].

2.4.3. Git

Git es un software de código abierto destinado al control de versiones de archivos. Es ampliamente utilizado en la industria del software para el versionado de códigos fuente de aplicaciones. La principal característica de Git reside en la posibilidad de mantener un historial de los archivos mediante la realización de *commits*, lo que representa un punto de restauración. También provee la posibilidad de mantener distintas versiones de archivos de forma paralela mediante *branches* o ramas creadas a partir de un determinado *commit* [21].

2.4.4. GitLab

Gitlab es un servicio web de control de versiones y desarrollo de software colaborativo basado en Git. Provee tanto mecanismos de gestión de repositorios de código en la nube, así como administración de documentación, seguimiento de errores e incidencias, servicios de integración continua, entre otros. [22]

2.4.5. Integración Continua

La integración continua es una práctica de desarrollo de software mediante la cual los desarrolladores envían cambios de forma periódica a un repositorio de código compartido mediante un sistema de control de versiones como Git. Antes de cada envío, es posible ejecutar automáticamente pruebas automatizadas sobre los nuevos cambios realizados en el código. De esta forma, es posible identificar inmediatamente cualquier error como medida

de verificación adicional antes de la integración [23].

Un *pipeline* es un conjunto de etapas que definen las acciones a realizar durante el proceso de integración continua (descargar código, compilar, ejecutar pruebas, desplegar, etc.). Se especifican mediante una sintaxis específica dependiendo del servicio [24].

2.4.6. Dependencia de software

Una dependencia de software es una aplicación o librería externa requerida por otro software para su funcionamiento. [25]

2.4.7. Versionado de software

El versionado de software se refiere al proceso de asignación de nombres o números a estados del software, permitiendo a los desarrolladores saber cuándo se han realizado cambios para así realizar un seguimiento de los cambios implementados [26].

A modo de ejemplo, Semantic Versioning (SemVer) es un estándar de versionado de software cuyo formato se conforma de tres números enteros separados por un punto (*MAJOR.MINOR.PATCH*), los cuales se incrementan de la siguiente manera [27]:

- *MAJOR* cuando se realizan cambios no retrocompatibles con versiones anteriores
- *MINOR* cuando agrega funcionalidad de una manera compatible con versiones anteriores
- *PATCH* cuando se realizan correcciones de errores compatibles con versiones anteriores.

2.4.8. Gestor de proyectos

Un gestor de proyectos es una aplicación destinada a descargar e instalar desde repositorios de librerías las dependencias de un proyecto. Usualmente se basan en uno o más archivos de dependencias que incluyen, para cada una, la versión requerida [25]. También pueden definir mecanismos de compilación de proyectos y ejecución de pruebas mediante comandos predefinidos, junto con estándares de configuración [25].

A continuación se presentan ejemplos de gestor de proyectos:

- **Maven**
Maven es un gestor de proyectos que provee mecanismos de mantenimiento e instalación de dependencias desde un repositorio público basándose en archivos de dependencias *pom.xml*. También provee mecanismos de compilación y ejecución de pruebas de forma estándar para proyectos construidos bajo la plataforma Java EE mediante la utilización de comandos predefinidos [28].

- **NPM**

Node Package Manager (NPM), al igual que Maven, es un gestor de proyectos que provee mecanismos de gestión e instalación de dependencias desde un repositorio público, junto con mecanismos de compilación y ejecución de pruebas mediante comandos predefinidos sobre proyectos de desarrollo ejecutados sobre la plataforma Node.js [29].

2.4.9. Docker

Docker es un software de código abierto que permite automatizar el despliegue de aplicaciones dentro de contenedores de software ligeros y portables, permitiendo que las aplicaciones puedan ejecutarse en cualquier máquina con Docker instalado. De esta forma, se proporciona una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos [30].

Un contenedor es una unidad estándar de software que empaqueta el código y todas sus dependencias para que la aplicación se ejecute de forma rápida y confiable en cualquier entorno. Una imagen de contenedor Docker es un paquete de software liviano, independiente y ejecutable que incluye todo lo necesario para ejecutar una aplicación dentro de un contenedor: código, herramientas del sistema, bibliotecas y configuraciones. En otras palabras, las imágenes de los contenedores se convierten en contenedores en tiempo de ejecución. [31].

Docker Compose es una herramienta que permite definir y ejecutar aplicaciones Docker de múltiples contenedores utilizando un archivo de configuración YAML [32] donde se configuran los servicios de la aplicación [33].

2.5. Recuperación de Información

2.5.1. XML, XSD, XSLT

eXtensible Markup Language (XML) es un estándar de representación que provee un formato universal para datos y documentos estructurados, interpretable por computadoras [34] [35].

XML Schema Definition (XSD) o esquema XML es un estándar que permite describir la estructura de un documento XML. El propósito de un esquema XML es definir los bloques de construcción esperados de un documento XML: elementos y atributos que pueden aparecer en un documento, número y orden de elementos secundarios, tipos de datos para elementos y atributos, etc. [36].

eXtensible Stylesheet Language for Transformations (XSLT) es un estándar de transformación de documentos XML. Se utiliza para transformar un documento XML fuente,

utilizando una o varias reglas, obteniéndose documentos XML y documentos HTML, entre otros [37].

2.5.2. Canales RSS

Really Simple Syndication (RSS) es un formato de distribución de información, avisos de actualización y contenido, usualmente utilizado por canales de noticias y blogs. Es consumido por programas de computadora que organizan los titulares y avisos para una fácil lectura, usualmente mediante código XML interpretable por una computadora [38].

3. Análisis de la problemática

Este capítulo tiene como objetivo analizar la problemática planteada en el proyecto. Para esto, se analizan las aplicaciones web desde un punto de vista arquitectónico y funcional. Además, se estudian observatorios tecnológicos existentes, analizando funcionalidades y objetivos. Por último, se presenta un relevamiento de los requerimientos deseados en un Observatorio Tecnológico orientado al desarrollo de aplicaciones web.

3.1. Arquitectura de aplicaciones web

Tomando en cuenta la arquitectura cliente-servidor para el desarrollo de aplicaciones web, se destaca el modelo arquitectónico separado en niveles. En él, se presentan tres niveles bien diferenciados como se muestran en la Figura 4.

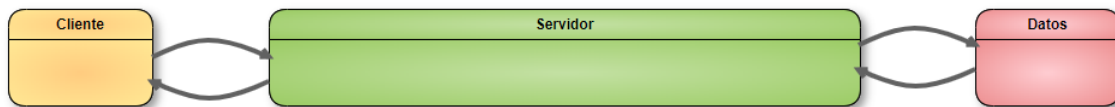


Figura 4: Cliente - servidor [39]

El primer nivel es el del cliente, el cual se compone de navegadores web que interactúan y se coordinan con los servidores web utilizando protocolos estandarizados [13].

El segundo nivel se compone de la lógica de la aplicación y es la encargada de reunir los tres niveles de la arquitectura, procesando las diversas entradas recibidas por los clientes, y encargándose de la interacción con la base de datos [13].

El tercer nivel está compuesto por el sistema de gestión de la base de datos (DBMS). Participa en la lógica de la aplicación al proporcionar métodos de acceso, inserción y actualización de los datos [13].

3.1.1. Especificación de niveles

En esta sección se detallan los niveles de la arquitectura cliente-servidor.

1. Primer nivel - Cliente

Los navegadores de los clientes son los encargados de consumir y ejecutar la capa de presentación de una aplicación web [5].

2. Segundo nivel - Servidor

Como se muestra en la Figura 5, el segundo nivel se compone de cuatro capas: presentación, servicios, negocio y datos [5].

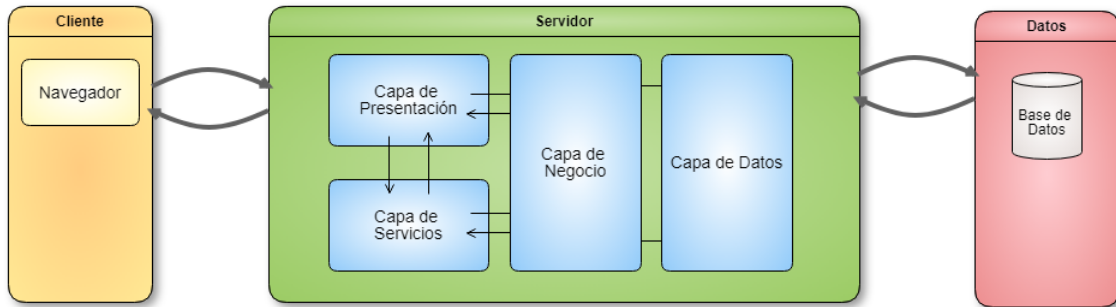


Figura 5: Capas de cliente - servidor [39]

La capa de presentación proporciona a los clientes la interfaz de usuario de interacción con la aplicación. Esta capa de presentación, que puede consistir en una MPA o SPA, será ejecutada por el navegador del cliente [5].

La capa de servicios es la encargada de proveer interfaces de acceso a la capa de negocio de forma de no exponer la lógica de la misma. Este servicio puede ser consumido tanto por la capa de presentación de la aplicación, como por sistemas externos que requieran de integración con la aplicación [5].

La capa de negocio implementa la lógica de la aplicación. Recibe peticiones de las capas superiores, ya sea capa de presentación o capa de servicios, y consume servicios de capas inferiores para retornar su respuesta[5].

La capa de datos proporciona acceso a sistemas de almacenamiento externos como bases de datos. Los componentes de acceso a datos en esta capa son responsables de exponer los datos almacenados a la capa de negocio, así como de proveer métodos uniformes de inserción y actualización de los mismos [5].

3. Tercer nivel - Datos

El tercer nivel se compone de las bases de datos encargadas de implementar la capa de persistencia de datos de la aplicación.

3.1.2. Componentes de cada capa

En esta sección se describen los componentes de cada capa como se muestra en la Figura 6.

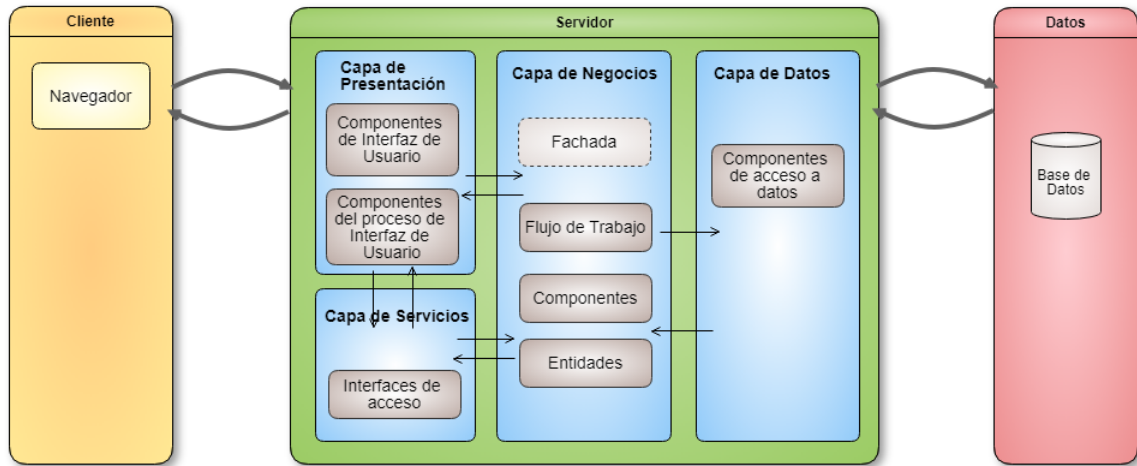


Figura 6: Componentes de cada capa [39]

1. Componentes de capa de presentación

- **Componentes de la Interfaz de Usuario**

Se corresponde a los elementos visuales que se utilizan para mostrar información, así como también capturar, validar y aceptar las entradas del usuario. La función de los Componentes de la Interfaz de Usuario es representar la lógica y los datos subyacentes de la aplicación en una interfaz [39].

- **Componentes del Proceso de Interfaz de Usuario**

Define el comportamiento lógico y la estructura de la aplicación independientemente de la interfaz de usuario. Estos componentes se encargan de implementar los casos de uso de la aplicación y gestionar las interacciones del usuario con la lógica y estado de la misma [39].

2. Componentes de capa de servicios

Una capa de servicios debe contener un conjunto de componentes que implementen tareas especificadas en interfaces de acceso estándar. Las Interfaces de acceso definen contratos de cada servicio provisto mediante una especificación estándar como REST y SOAP [40].

3. Componentes de capa de negocio

- **Fachada**

Proporciona una interfaz de acceso a la lógica de negocio. [39].

- **Flujo de Trabajo y Entidades**

Componen la lógica de aplicación que se relaciona con la recuperación, procesamiento, transformación y la administración de los datos de las aplicaciones. Incluye la aplicación de reglas de negocio garantizando la consistencia y validez de los datos. Por otro lado, las entidades del negocio contienen la lógica y aquellos datos que son necesarios para la representación de la realidad dentro de la aplicación [39].

4. Componentes de capa de datos

- **Componentes de acceso a datos**

Representan los componentes que permiten el acceso a las diversas bases de datos, implementando tareas de acceso a los datos de forma genérica permitiendo encapsular, reutilizar y centralizar la lógica de acceso de forma independiente al DBMS utilizado [39].

3.1.3. Capa Transversal

Los componentes transversales son los encargados de implementar tipos de funcionalidades que involucran a más de una capa, como se muestra en la Figura 7.

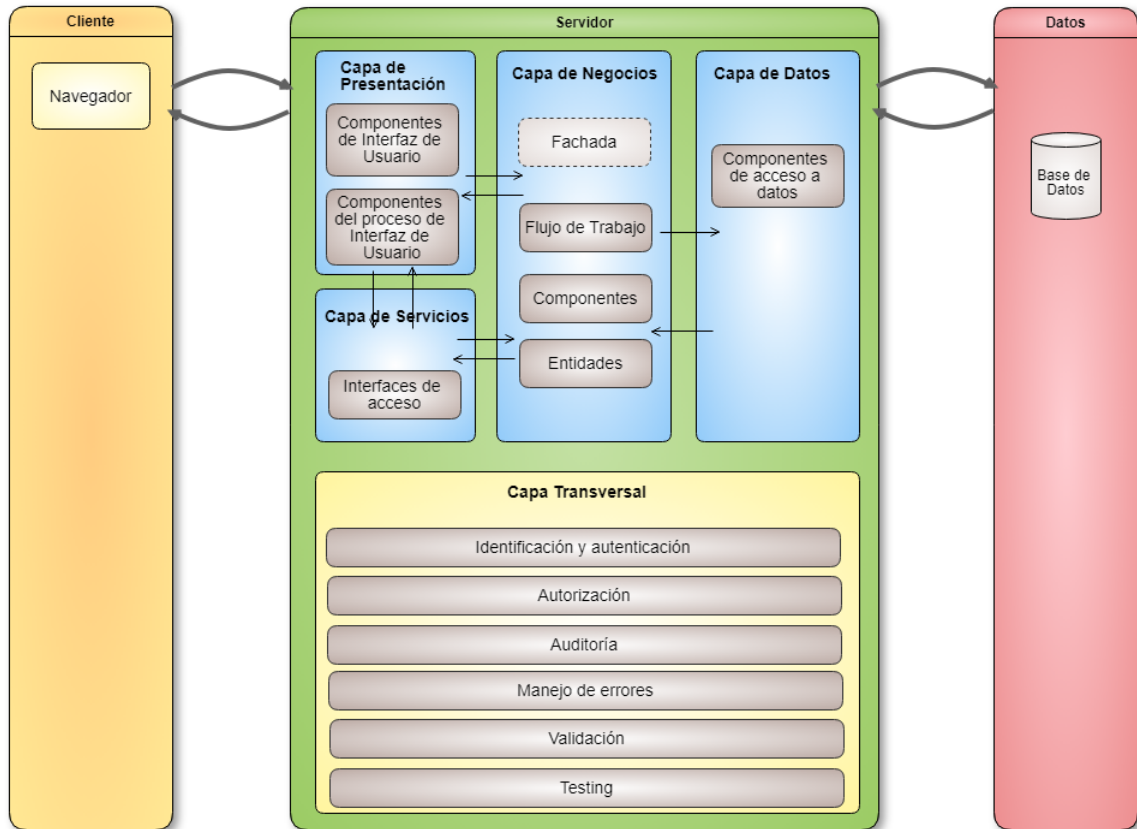


Figura 7: Capa transversal [39]

Ejemplos de estos tipos de funcionalidades son [39] [41] [42]:

- **Identificación y autenticación**

La identificación provee mecanismos para identificar usuarios en el sistema, mientras que la autenticación permite la verificación de dichas identidades.

- **Autorización**

Determina los recursos del sistema a los que se tiene acceso dependiendo de los privilegios con los que disponga el usuario autenticado.

- **Auditoría**

Recopila información sobre el uso de los recursos del sistema. A partir del procesamiento de dicha información, es posible detectar desde usos incorrectos por parte de los usuarios hasta actividades maliciosas como podrían ser intentos de elevación de privilegios, entre otros.

- **Manejo de errores**

Se debe definir un manejo de errores que permita tanto capturarlos como retornar mensajes adecuados al usuario sin comprometer información sensible del sistema.

- **Validación**

Protege las capas inferiores validando todas las peticiones que se reciben. Una validación incorrecta de las entradas del usuario podría llevar a vulnerabilidades que comprometan la capa de negocio y datos del sistema.

- **Testing**

Consiste de implementación de mecanismos de pruebas funcionales y de integración sobre la aplicación en general.

3.1.4. Herramientas de desarrollo y administración

Existen una serie de componentes que no forman parte de una arquitectura de aplicación web, pero resultan relevantes en el desarrollo y gestión de las mismas. Estos componentes pueden proveer soporte para la gestión de aplicaciones web (software de gestión de proyectos, servicios de integración continua), como del desarrollo (entornos de desarrollo, repositorios de código) y documentación. En la Figura 8 se ilustran estos componentes.

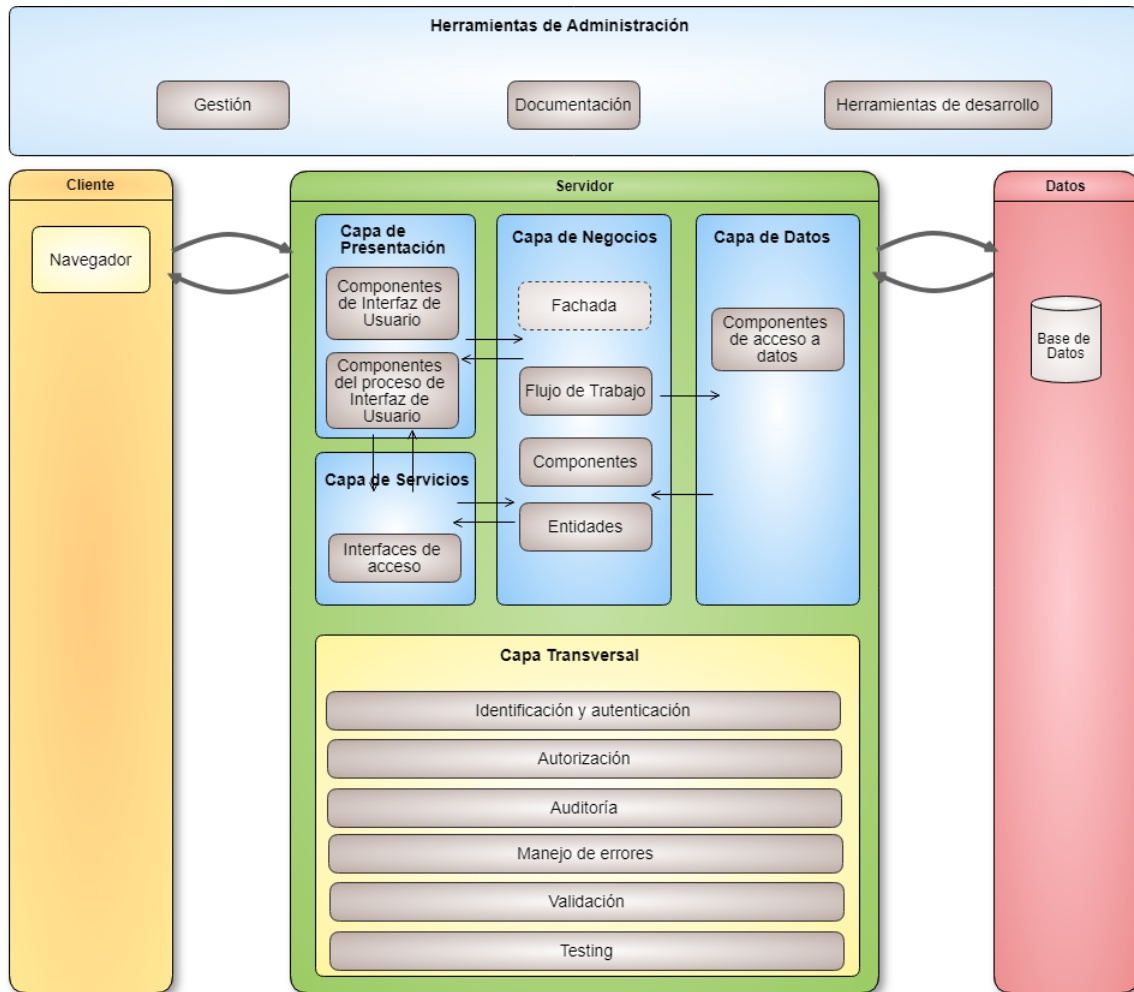


Figura 8: Herramientas de administración [39]

3.2. Observatorios Tecnológicos

En esta sección se detallan algunos de los observatorios tecnológicos existentes planteando para qué son utilizados, qué funcionalidades brindan y datos sobre su utilización.

3.2.1. Funciones principales

Un Observatorio Tecnológico es un espacio que permite gestionar información en diferentes áreas, transformar los datos en información útil y ayudar a la toma de decisiones estratégicas, tácticas y operacionales de una empresa [43].

El trabajo realizado por Rodrigo Díaz Ayala [44] para el Instituto Politécnico Nacional de Ciudad de México presenta un estudio respecto al estado actual y a los retos que demandan los observatorios tecnológicos en general. Presenta a los observatorios tecnológicos como herramientas de enfoque diverso, que varían según el área temática al que apuntan. Se pueden tener observatorios tecnológicos que apunten al área de la Salud, otros que apunten a la Política y derechos Humanos, otros dirigidos a las Tecnologías de la Información, etc. En base a un relevamiento propone una serie de funcionalidades generales que debe cumplir un Observatorio Tecnológico:

- F1: Gestión de los datos e información del Observatorio Tecnológico.
- F2: Generación de reportes para obtener indicadores, como podrían ser tendencias, entre otros.
- F3: Vigilancia tecnológica mediante mecanismos de retroalimentación del Observatorio Tecnológico.
- F4: Inteligencia competitiva a modo de brindar funcionalidades para que los usuarios sean capaces de evaluar riesgos y obtener oportunidades ante la experiencia de terceros.
- F5: Difusión de la información mediante notificaciones, registro, alertas, paneles de noticias, etc.
- F6: Suministro de la información considerando la presentación de la información centrada en la experiencia de usuario.

3.2.2. Criterios de selección de observatorios tecnológicos existentes

Esta sección tiene por objetivo determinar criterios para la selección de Observatorios Tecnológicos que existan actualmente.

En la sección 3.2.1, se presentó un conjunto de funcionalidades que deben cumplir los observatorios tecnológicos sin importar el área al que estén destinados. De esta forma, se propone un relevamiento de observatorios tecnológicos existentes en base a estos criterios para evaluar dichas funcionalidades.

En primera instancia se toma como preselección aquellos que pertenezcan a diferentes partes geográficas del mundo. Además, se espera que dichos observatorios se encuentren activos, pudiéndose comprobar actividad reciente.

En segunda instancia, se definen los siguientes criterios de selección no excluyentes:

- Cr.1: Observatorios Tecnológicos orientados a las Tecnologías de la Información, preferentemente orientados a desarrollo de aplicaciones web.

Cr.2: Disponibles en la nube: Se entiende que es de interés que se encuentren disponibles en la nube, a modo de facilitar el acceso.

Cr.3: Gratuitos: De esta manera se logra que un mayor número de usuarios puedan acceder al Observatorio pudiendo enriquecerse con la información que provee.

3.2.3. Observatorios Tecnológicos seleccionados

En esta sección se analizan los observatorios tecnológicos seleccionados mediante los criterios de selección establecidos en la sección 3.2.2, para luego comparar sus características verificando las funcionalidades especificadas en la sección 3.2.1.

▪ Observatorio Tecnológico del ITI

El Observatorio Tecnológico del ITI tiene origen a finales del año 2008 y ha sido desarrollado por el Instituto Tecnológico de Informática (ITI) que se encuentra en el Parque Científico de la Universidad Politécnica de Valencia, España. La motivación del observatorio es estar a la vanguardia respecto a las temáticas que se ven involucradas en las TIC. Para ello, clasifica las áreas temáticas con el propósito de agrupar las líneas tecnológicas que las TIC abordan [45].

Este observatorio está orientado a empresas e instituciones científicas. Busca facilitar a los usuarios en la tarea de planeación de estrategias tecnológicas, potenciando la innovación y el desarrollo. Para esto, implementa una serie de acciones como recopilación de información, almacenamiento, categorización, ofreciéndola de manera intuitiva mediante distintos reportes [45].

El observatorio es una herramienta disponible en la nube que contribuye en el descubrimiento de las tecnologías existentes y el análisis de su impacto, para posteriormente poder aplicar dichas tecnologías en los procesos de las empresas y organizaciones [45].

Dentro de las principales funcionalidades que posee el observatorio del ITI se encuentran:

- Acceso a la información del Observatorio TIC desde el portal público.
- Suscripción a boletines de novedades y alertas.
- Reportes de tendencias tecnológicas, tanto sobre tecnologías de última generación como de tecnologías en investigación.

Este observatorio fue seleccionado ya que es una plataforma gratuita disponible en la nube, que se mantiene con una comunidad activa bajo el respaldo de una universidad. [45].

■ **Observatorio de Ciencias y de Tecnología (OCT)**

El Observatorio de Ciencias y de Tecnología (OCT), o *Observatoire des Sciences et des Technologies* por su nombre en francés, es un Observatorio Tecnológico desarrollado por la Universidad de Quebec en Canadá en el año 1997, con la finalidad de estudiar la actividad científica, tecnológica, social y política del país [46].

El Observatorio se caracteriza por crear indicadores relativos a las ciencias de las tecnologías, y por contribuir a la creación y mantenimiento de bases de datos sobre investigación, desarrollo tecnológico, publicaciones, entre otros [46].

El OCT es utilizado en universidades, ministerios y otras organizaciones públicas de Canadá, ofreciéndoles asistencia para evaluar sus actividades científicas y tecnológicas, junto con el acceso a una plataforma con bases de datos de todo tipo, tales como revistas científicas internacionales [46].

De los servicios ofrecidos por el OCT, se destacan lo siguientes:

- Producción de Indicadores de Ciencias de la Tecnología.
- Acceso a bases de datos de información
- Paneles de noticias
- Asistencia en la evaluación de las actividades de Ciencia y Tecnología.
- Acceso a una red de miembros del Observatorio para fomentar la transferencia de conocimiento.
- Publicación de artículos científicos e informes de investigación.

El OCT fue seleccionado ya que provee acceso gratuito desde la nube a un gran repositorio de información de distintas temáticas relacionadas no sólo con las tecnologías de la información, sino con el avance tecnológico de distintas áreas tales como la medicina, educación y política [46].

■ **European Information Technology Observatory (EITO)**

El *European Information Technology Observatory* (EITO) es un Observatorio Tecnológico orientado a las tecnologías de la información creado en 1993 con fines comerciales, mantenido por la Asociación Federal de Tecnología de la Información, Telecomunicaciones y Nuevos Medios en Alemania [47].

EITO atiende un amplio espectro de clientes, desde empresas de informática y telecomunicaciones, proveedores y usuarios de tecnología, analistas de mercado, consultores informáticos, prensa y organizaciones involucradas en la investigación y

desarrollo de las tecnologías de la información [47].

Las principales temáticas abarcadas por el EITO se resumen en:

- Tecnología de la Información
- Equipos y servicios de telecomunicaciones
- Software
- Electrónica
- Internet de las Cosas (IoT)
- *Big Data*
- Inteligencia Artificial

EITO provee acceso a un conjunto de reportes pagos orientados a las tecnologías de la información. Entre los que resultan de mayor relevancia para el proyecto, se encuentra un reporte denominado *Key Technology Trends in Europe 2017/18*, el cual ofrece una visión general de las principales tendencias tecnológicas que actualmente están dominando el entorno y la industria de las TIC, centrándose en las temáticas de la nube, el *big data* y el Internet de las Cosas (IoT) en Europa. Por otro lado, provee otra serie de reportes más generales acerca del desarrollo y las tendencias del mercado de las TIC para cada uno de los principales países [47].

Este Observatorio fue seleccionado ya que, a pesar de no ser gratuito, recolecta información no solo de países de Europa, sino que de los principales mercados de las tecnologías de la información como Estados Unidos y Japón, lo que resulta útil para poder relevar e inferir tendencias de distintos partes del mundo [47].

Sin embargo, dentro de las temáticas abarcadas dentro de las tecnologías de la información, no especifica ninguna orientada específicamente al desarrollo de aplicaciones web [47].

3.2.4. Comparación de Observatorios Tecnológicos

A continuación se presentan cuadros comparativos, donde en el cuadro 1 se comparan los Observatorios Tecnológicos seleccionados en la sección 3.2.3 desde el enfoque del criterio de selección.

Observatorio Tecnológico	Criterios de Selección		
	Cr.1	Cr.2	Cr.3
ITI	X	X	X
OCT	X	X	X
EITO	X	X	

Cuadro 1: Comparación de Criterios de Selección

Mientras que en el cuadro 2, se comparan los Observatorios Tecnológicos teniendo en cuenta las funcionalidades planteadas en la sección 3.2.1. Todos los observatorios considerados cumplen con las diferentes funcionalidades mediante alguna estrategia. Es marcado con el identificador *Si* para indicar que el observatorio cumple con la funcionalidad, mientras que con el identificador *NC* se especifica que no se pudo comprobar dicha funcionalidad.

Observatorio Tecnológico	Funcionalidades					
	F1	F2	F3	F4	F5	F6
ITI	Si	Si	Si	Si	Si	Si
OCT	Si	NC	Si	Si	Si	Si
EITO	Si	Si	Si	Si	Si	Si

Cuadro 2: Comparación de Funcionalidades

3.3. Relevamiento de requerimientos

Esta sección tiene por finalidad especificar los requerimientos de un Observatorio Tecnológico orientado al desarrollo de aplicaciones web. Para esto, se brinda una descripción del problema y se definen actores principales. Además, se especifican requerimientos funcionales y no funcionales para la plataforma.

3.3.1. Descripción del problema

Existe la necesidad de contar con una herramienta que sea capaz de ayudar tanto a empresas como a emprendedores que desean incursionar en el mundo de desarrollo de aplicaciones web, con el fin de mitigar la problemática presentada por la constante evolución que experimenta esta rama del desarrollo de software.

Esta herramienta debe ser capaz de generar información que permita facilitar el trabajo tanto en toma de decisiones como en diseño de estrategias, brindando facilidades a la hora de comenzar un nuevo desarrollo, ya sea proporcionando un entorno con reportes que proporcionen una guía para el usuario, junto con la posibilidad de descarga de proyectos prototipo completos.

Debe proveer mecanismos de ayuda en la elección entre las diversas tecnologías existentes, considerando compatibilidades y recomendaciones a la hora de sugerirlas.

A su vez, debe ofrecer un análisis de nuevas tendencias, sistemas de puntuación, acceso a documentación fidedigna, buenas prácticas, experiencias de proyectos realizados por otros usuarios y un sistema de notificaciones ante cambios de versiones de las diferentes tecnologías utilizadas.

La herramienta debe brindar funcionalidades que permitan gestionar la información de manera eficiente convirtiéndola en conocimiento a disposición de los usuarios.

3.3.2. Actores principales

Los actores principales son organizaciones dedicadas al desarrollo de software que deseen contar con una herramienta que les permita generar una metodología de trabajo, unificar estándares de desarrollo y acortar los tiempos en la curva de aprendizaje. También está destinada a usuarios particulares entusiastas del mundo del desarrollo de aplicaciones web que deseen mantenerse al día en cuanto a últimas metodologías, tendencias y tecnologías.

3.3.3. Requerimientos funcionales

- **Gestión de tecnologías, *frameworks* y herramientas para el desarrollo de aplicaciones web**

El sistema debe permitir gestionar tecnologías, frameworks y herramientas manteniendo un registro de versiones, recomendaciones, compatibilidades e incompatibilidades entre las mismas.

- **Gestión de documentación y buenas prácticas**

El sistema debe permitir mantener una biblioteca de documentación y buenas prácticas asociadas a las distintas tecnologías y herramientas del Observatorio.

- **Guía de selección de tecnologías para la creación de aplicaciones web**

El sistema debe proveer una guía de selección de tecnologías y herramientas que puedan operar en conjunto en un proyecto de aplicación web.

- **Integración con servicios de repositorios de código e integración continua para almacenamiento de proyectos de ejemplo**

El sistema debe poder integrarse con repositorios de código donde se encuentren almacenados distintos proyectos de ejemplo de uso de tecnologías. También debe proveer mecanismos de integración con distintos servicios de integración continua a modo de ejecutar pruebas automáticas sobre los proyectos para verificar compatibilidades ante nuevas liberaciones de tecnologías.

- **Paneles de noticias y reportes**

El sistema debe ofrecer un panel de noticias relacionadas al mundo de desarrollo de aplicaciones web, así como también distintos reportes generales sobre el uso de las tecnologías del Observatorio.

- **Gestión de usuarios, roles y permisos**

El sistema debe permitir el manejo de distintos usuarios y roles dentro de las organizaciones mediante la definición de distintos permisos.

3.3.4. Requerimientos no funcionales

- **Plataforma**

Implementación de las funcionalidades y servicios con la utilización de la plataforma Java Empresarial (Java EE). En cuanto a la interfaz de usuario, no hay restricciones en la plataforma a utilizar.

- **Extensibilidad**

Se debe poder extender la arquitectura de la aplicación y soportar nuevas integraciones de forma configurable por administradores.

3.4. Resumen y conclusiones

En base al relevamiento de observatorios tecnológicos existentes y requerimientos planteados, se observan varias propuestas pero ninguna que se enfoque en el desarrollo de aplicaciones web de forma explícita.

Los enfoques de los diferentes observatorios aplican a un nivel más general abarcando temas de las diferentes tecnologías de la información y comunicación, sin profundizar en ramas específicas.

Las funcionalidades y objetivos principales se basan en la centralización de la información, proveyendo un acceso simplificado a la misma mediante suscripciones a boletines y alertas.

Si bien la base de un observatorio orientado al mundo de las aplicaciones web tiene varios aspectos en común con los presentados, resulta de interés profundizar y enriquecer dichas funcionalidades con aspectos más relevantes a la implementación.

4. Solución propuesta

En este capítulo se propone una solución de Observatorio Tecnológico orientado al desarrollo de aplicaciones web. Para ello, se elabora un modelo conceptual y se listan sus funcionalidades.

4.1. Descripción general

Se propone una solución de Observatorio Tecnológico orientado al desarrollo de aplicaciones web, con el fin de atacar la problemática planteada ofreciendo soporte a las necesidades tanto de las organizaciones que requieren de una innovación constante, como de los usuarios que deseen incursionar en el desarrollo web.

Se define un modelo que representa la arquitectura de un proceso de desarrollo de aplicaciones web, considerando aspectos que abarcan desde el diseño, implementación y puesta en producción, partiendo de la base que las mismas pueden descomponerse en varios componentes. Para esto, se introduce el concepto de *arquitectura general de proceso de desarrollo de aplicación web* (AGPDAW), como el conjunto de componentes que dan forma a una aplicación web y al proceso de desarrollo en su totalidad. Cada uno de dichos componentes puede ser implementado mediante el uso de una o más herramientas o tecnologías, formando así lo que se conoce como un *Stack Web*: conjunto de herramientas y tecnologías utilizadas para la implementación de una AGPDAW.

La Figura 9 muestra conceptualmente cómo distintos tipos de usuarios acceden al Observatorio Tecnológico.

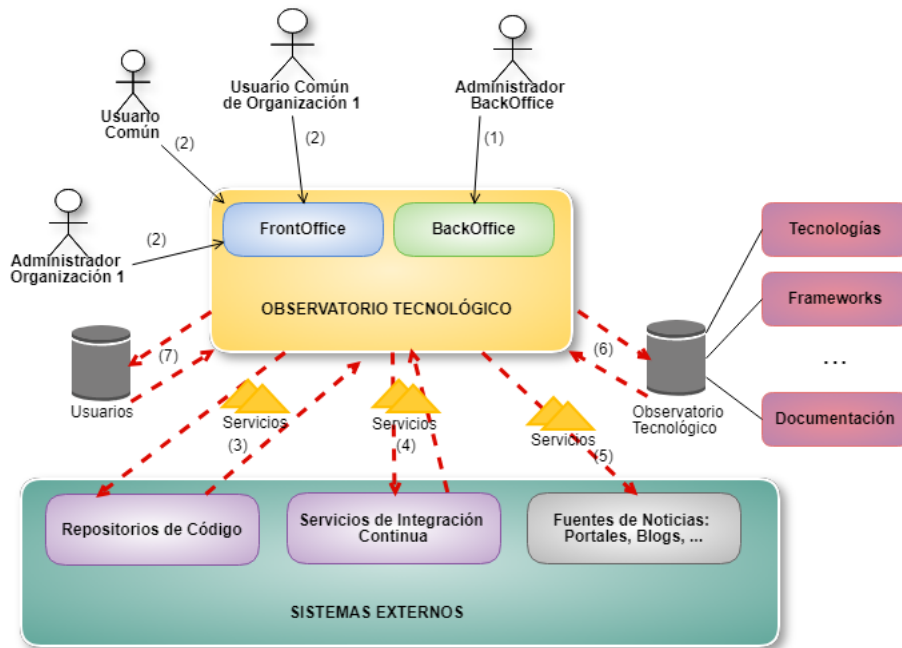


Figura 9: Solución propuesta - Observatorio Tecnológico

Se proponen dos interfaces, una de administración (*BackOffice*) y otra de acceso al Observatorio (*FrontOffice*). El *BackOffice* será utilizado por usuarios administradores (1) del Observatorio Tecnológico y está destinado al mantenimiento general del mismo (alta, baja y edición de entidades globales del sistema).

Por otro lado, el *FrontOffice* será utilizado tanto por organizaciones como por usuarios particulares (2) con el fin de acceder al Observatorio Tecnológico.

Contará con una capa de servicios que brinde diferentes funcionalidades tanto para las aplicaciones de *FrontOffice*, *BackOffice* como funcionalidades del Observatorio en general. Para ello, será necesaria la integración con diferentes repositorios de código (3), servicios de integración continua (4) y portales de noticias (5).

También será necesario contar con bases de datos con la capacidad necesaria para almacenar la información que genere y recolecte el Observatorio Tecnológico (6), como por ejemplo, tecnologías, *frameworks*, *stacks web*, documentación, entre otros; y para la administración de los datos de usuarios (7).

A modo de ejemplificar los conceptos presentados anteriormente, se plantea la situación donde un usuario lleva adelante el desarrollo de una aplicación web que incluye los siguientes componentes, donde cada uno se encarga de resolver problemas específicos:

- Capa de presentación
- Capa de servicios
- Capa de negocio
- Capa de acceso a datos
- Capa de persistencia
- Capa de ejecución
- Herramientas de desarrollo

Dado que cada componente tiene asociado tecnologías que dan solución al mismo, el usuario puede generar un *stack web*, seleccionando una o más tecnologías o herramientas de dichos componentes.

Dada la cantidad de arquitecturas y de combinaciones de tecnologías posibles, resulta difícil encontrar una AGPDAW y *stack web* general que satisfaga los requerimientos de todas las aplicaciones web, por lo que los usuarios pueden generar sus propios AGPDAW y *stack web* a medida, a modo de satisfacer sus necesidades.

4.2. Características principales

En esta sección se brindan los aspectos principales respecto a la solución brindada para el Observatorio Tecnológico orientado al desarrollo de aplicaciones web.

4.2.1. Espacios de trabajo

Los usuarios disponen de diversos espacios de trabajo dentro de la aplicación. Por defecto, trabajan en el espacio personal del usuario, donde todas las acciones que se realicen, se harán bajo su nombre. En caso de pertenecer a alguna organización, el usuario puede cambiar el espacio de trabajo para que todas sus acciones se ejecuten en nombre de la organización.

4.2.2. Ciclo de vida de las principales entidades

Cuando un usuario cree alguna entidad (tecnologías, AGPDAW, *stacks web*, etc.) bajo algún espacio de trabajo, tiene la opción de elegir para crearlos de forma privada o pública para la comunidad de usuarios. En caso de crearla en el espacio de trabajo personal en modo privado, la misma se mantiene como un proyecto personal privado del usuario, donde solamente él tiene acceso. Si decide crearla pública, la misma es visible por todos los usuarios del Observatorio Tecnológico.

Si en cambio decide crearla bajo el espacio de trabajo de una organización en modo privado, se mantiene accesible a los usuarios de la organización teniendo en cuenta los permisos otorgados a los diferentes grupos. Si decide crearla pública para la comunidad, la misma permanece visible para todos los usuarios del Observatorio Tecnológico, manteniendo a la organización como responsable, y al usuario como creador. Luego de creadas las entidades, es posible cambiar la configuración de acceso de las mismas.

En la Figura 10 se muestra el flujo para la visibilidad de las diferentes entidades: AGP-DAW, stacks web, tecnologías, entre otros.

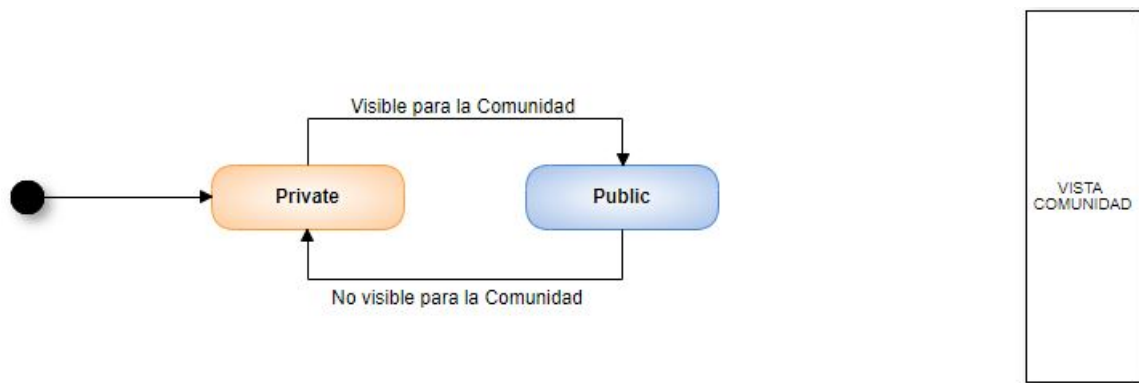


Figura 10: Visibilidad de Entidades

Al momento de crear las entidades, se inicializan en un estado *WORK IN PROGRESS*, para luego cambiarse a estado *PUBLISHED* y/o *DEPRECATED*.

El estado *WORK IN PROGRESS* se utiliza mientras se está trabajando sobre alguna entidad. Cuando la entidad se encuentra en un estado estable, es posible cambiarla al estado *PUBLISHED*.

El estado *DEPRECATED* es posible alcanzarlo a partir de los estados *PUBLISHED* o *WORK IN PROGRESS*, y se utiliza para representar las entidades que fueron discontinuadas o que no son más válidas dentro del Observatorio Tecnológico. De esta forma se advierte a los usuarios que el material no es fiable.

En la Figura 11 se muestra el flujo de estados por los que pueden pasar las diferentes entidades.

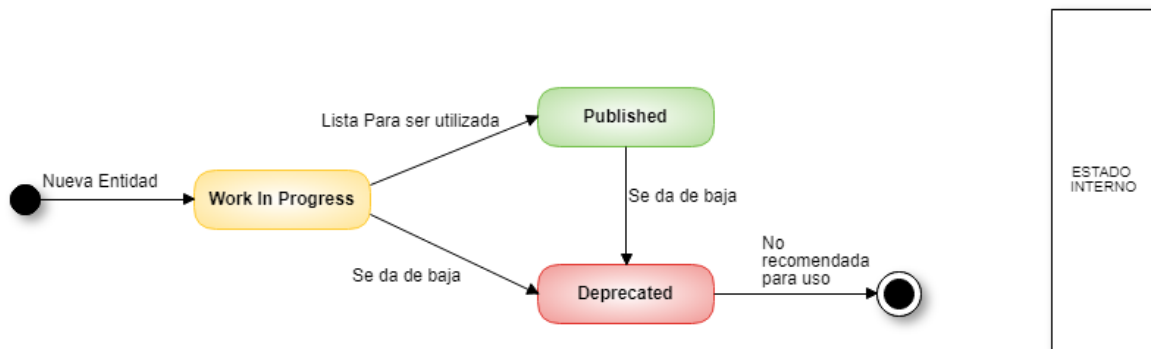


Figura 11: Estados de Entidades

A modo de ejemplo, al crearse un *stack web* dentro de una organización, se selecciona si se desea crearlo público o privado. Se puede trabajar sobre la entidad, salvar cambios y que quede visible para la comunidad. Una vez listo se publica, cambiando el estado de la entidad a *PUBLISHED*. Si luego de un tiempo, se decide que una tecnología utilizada en el *stack web* no es válida, basta con cambiar el estado a *DEPRECATED*.

4.2.3. Compatibilidades entre tecnologías

Cada tecnología se encarga de implementar uno o más componentes. A su vez, se relacionan con otras tecnologías a modo de determinar compatibilidad e incompatibilidad entre las mismas. Cabe destacar que muchas veces, dos tecnologías son compatibles entre sí a través de la utilización de una o más tecnologías como intermediarias.

Se cuenta con tecnologías principales que agrupan conjuntos de tecnologías del mismo tipo, independiente de sus versiones. A su vez, las distintas tecnologías principales se agrupan según el ecosistema al que pertenecen.

De esta forma, se determinan los siguientes criterios de compatibilidad entre tecnologías:

- Por defecto, todas las tecnologías que comparten un mismo ecosistema son compatibles entre sí.
- En caso de tener dos tecnologías pertenecientes a diferentes ecosistemas las cuales son compatibles entre sí, se debe establecer la relación de compatibilidad de forma explícita.
- En caso de tener dos tecnologías incompatibles pertenecientes al mismo ecosistema, se debe establecer la relación de incompatibilidad de forma explícita.

- En caso de tener dos tecnologías incompatibles entre sí pero que a través de un conjunto de otras tecnologías se vuelven compatibles, se debe establecer la relación de compatibilidad a través de dichas tecnologías.
- En caso de tener dos tecnologías que además de ser compatibles entre sí, el uso de una sugiere el uso de la otra, se debe establecer una relación de recomendación, con el objetivo de ponderar el uso de la misma.

Dado que cada tecnología incluye una versión específica, es necesario que todas las tecnologías asociadas a una tecnología principal mantengan una coherencia en los formatos. De esta forma, se mantienen distintos estándares de versionado específicos asociados a las tecnologías principales. A su vez, cada estándar de versionado mantiene distintos formatos que especifican el estado de liberación de la versión (en desarrollo, estable, versión final, etc.).

4.2.4. Integración con sistemas externos

El sistema es capaz de detectar y actuar ante cambios de versiones de tecnologías. Las tecnologías y los *stacks web* pueden estar acompañados de recursos que incluyan ejemplos de código, los cuales a su vez pueden incluir casos de prueba automáticos. De esta forma, el sistema es capaz de, ante la liberación de una nueva versión de una tecnología específica, ejecutar los casos de prueba de recursos asociados a la misma, con el fin de detectar y alertar a los usuarios de aquellos casos que fallaron.

Para esto, el Observatorio cuenta con integraciones con servicios de repositorios de código para el almacenamiento y acceso a código fuente de los recursos asociados a tecnologías y *stacks web* incorporados en el Observatorio Tecnológico. Junto a esto, también cuenta con integraciones con distintos servicios de integración continua a modo de ejecutar los casos de pruebas de los recursos y notificar los resultados al Observatorio.

Por otro lado, el Observatorio cuenta con paneles de noticias relacionadas al mundo del desarrollo web. Para esto, se extrae la información de diferentes portales de noticias configurables por los administradores.

4.2.5. Reportes

El Observatorio ofrece un conjunto de reportes que cuantifican las tendencias de uso de las diferentes tecnologías a modo de asistir a los usuarios en el proceso de toma de decisiones a la hora de diseñar una solución tecnológica.

4.2.6. Usuario, roles y organizaciones

Se identifican dos tipos de usuarios:

- Usuarios administradores del sistema.

- Usuarios del Observatorio Tecnológico.

Los usuarios administradores del sistema son los únicos con acceso a la aplicación de *BackOffice*, y están encargados de la creación de componentes que dan forma a los AGP-DAW, así como del manejo y administración de usuarios y organizaciones. También son los encargados del mantenimiento de las tecnologías y recursos asociados a las mismas, con el fin de mantener un estándar de calidad. Para esto, reciben denuncias de usuarios sobre recursos de mala calidad, y tienen la potestad tanto de definirlos como obsoletos como darlos de baja por completo.

Los usuarios del Observatorio Tecnológico representan a los usuarios finales del sistema, los cuales pueden o no pertenecer a organizaciones. Además, son capaces de registrar organizaciones al Observatorio Tecnológico, pudiendo invitar a otros usuarios a unirse a la misma.

Se pueden definir distintos roles sobre los usuarios dentro de la organización, donde cada rol goza de ciertos privilegios. Por defecto, cada organización cuenta con un rol “Administrador de la Organización” el cual es asignado automáticamente al usuario que la creó, otorgándole la potestad de asignar dicho rol a otros usuarios pertenecientes a la misma organización. A modo resumen, el rol “Administrador de la Organización”, cuenta con los siguientes privilegios:

- Creación de roles y definición de permisos sobre las distintas entidades.
- Asignar/quitar roles a usuarios de la organización.
- Asignar/quitar usuarios como administradores de la organización.
- Enviar invitaciones de pertenencia a la organización.

Los usuarios administradores de una organización pueden crear diferentes roles, tales como Arquitectos, Desarrolladores I, Desarrolladores II, etc.

A cada rol se le define, para las AGPDAW, *stack web* y recursos, los permisos que puede tener sobre ellos, los cuales pueden ser “Crear”, “Editar” o “Ver”. A modo ejemplo, es posible configurar el grupo “Arquitectos” con permisos para crear, editar y ver AGPDAW, *stack web* y recursos; junto con un grupo “Desarrolladores I” con permisos solamente para ver AGPDAW y con permisos de crear, editar y ver *stack web* y recursos. De esta forma, se mantiene un control de acceso basado en roles orientado a modelar la realidad de las organizaciones, donde los arquitectos son los encargados de definir la arquitectura de los sistemas, mientras que los desarrolladores, más experimentados en el área práctica del desarrollo, pueden ayudar en la elección de tecnologías apropiadas para el problema.

4.2.7. Sistema de puntos

Se maneja un sistema de puntos sobre las entidades (AGPDAW, *stacks web*, tecnologías, etc) que son públicas para los usuarios. Los administradores reciben denuncias sobre recursos de mala calidad y puede darlos de baja y/o penalizar usuarios u organizaciones.

4.3. Modelo conceptual

La Figura 12 presenta el modelo conceptual que surge de la realidad planteada en la descripción general y de las características principales.

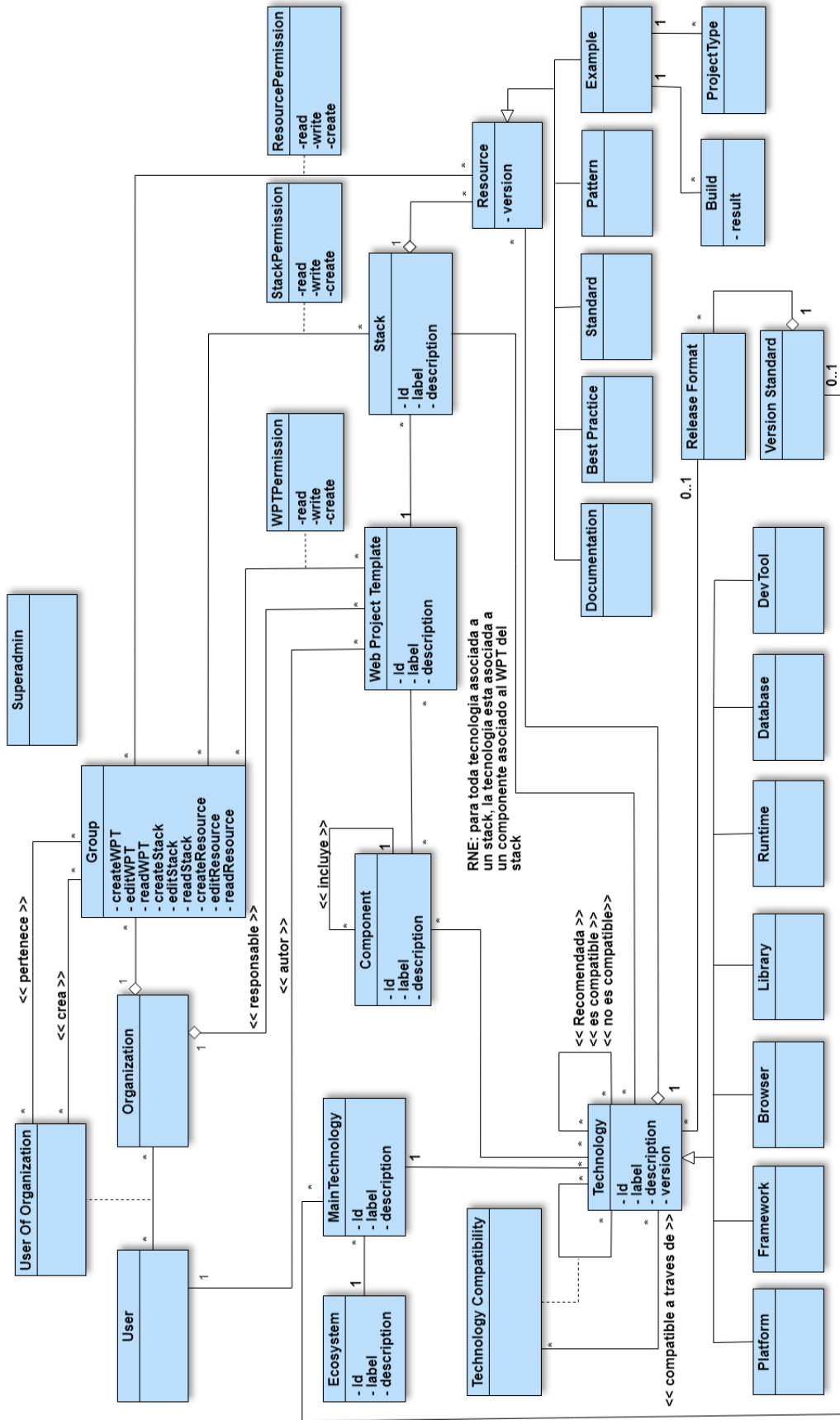


Figura 12: Modelo conceptual

4.3.1. Gestión de tecnologías

En la Figura 13 se presentan resaltados los elementos del modelo conceptual que son relevantes para la gestión de tecnologías.

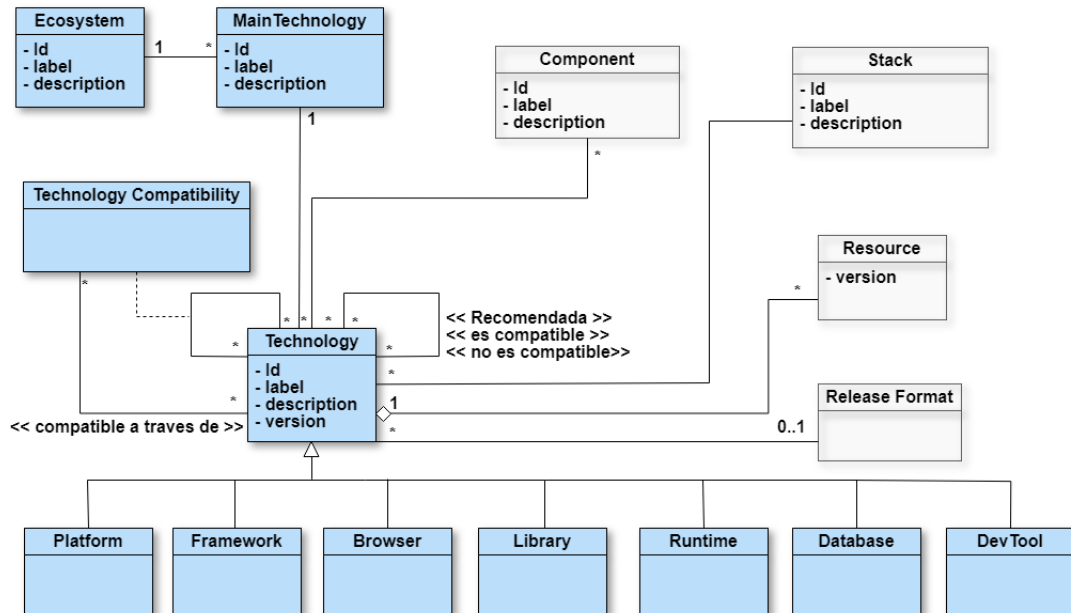


Figura 13: Gestión de tecnologías

A continuación se describen los elementos presentados:

- **Technology**

Representa las diferentes tecnologías y herramientas que se utilizan en el proceso de desarrollo de las aplicaciones web, las cuales podrían ser plataformas, librerías, navegadores, base de datos, entornos de ejecución, herramientas de desarrollo, entre otros. Las relaciones “es compatible”, “no es compatible”, “recomendada” y “compatible a través de” representa los criterios de compatibilidad que se definieron en la sección 4.2.3.

- **MainTechnology**

Representa la tecnología principal. Para cada tecnología se podrían tener diferentes versiones asociadas. Esta entidad refiere a la tecnología sin versionar. A modo de ejemplo, si se define la tecnología principal “Java”, se podría tener las tecnologías “Java 1.7” y “Java 1.8”.

- **Ecosystem**

Representa una agrupación de MainTechnology que comparten un mismo ecosistema. A modo de ejemplo, el ecosystem Java agrupa las MainTechnology JSF, JSP, JPA, etc.

- **TechnologyCompatibility**

Representa la relación de compatibilidad entre dos entidades Technology a través de un conjunto de entidades Technology.

4.3.2. Gestión de AGPDAW y *stacks*

En la Figura 14 se presentan resaltados los elementos del modelo conceptual que son relevantes para la gestión de los AGPDAW y *stacks web*.

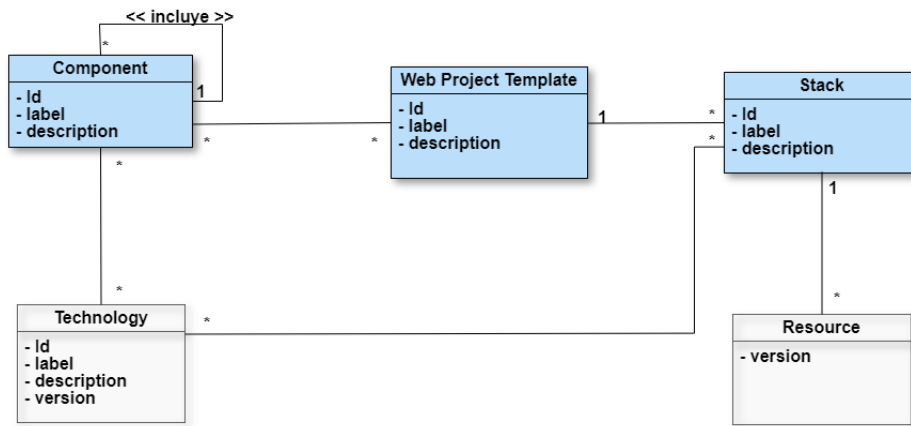


Figura 14: Gestión de proceso de desarrollo

A continuación se describen los elementos presentados:

- **Component**

Se refiere a los componentes encargados de resolver tareas específicas dentro del proceso de desarrollo de una aplicación web. Representa la forma en que pueden ser clasificadas las diferentes tecnologías con las que son implementadas las aplicaciones web. La clasificación puede ser arborescente, a modo de representar componentes más específicos para las diversas tecnologías. A modo ejemplo, se podría tener un componente “Capa de Presentación”, que a su vez esté compuesta por otros dos componentes, “SPA” y “MPA”. De esta manera, es posible tener tecnologías vinculadas al componente “SPA” y otras a “MPA”.

- **WebProjectTemplate**

Modela el concepto de AGPDAW, el cual agrupa un conjunto de entidades Component.

- **Stack**

Modela un *stack web*. Representa una implementación de un *web project template* mediante la utilización de una o más tecnologías por cada componente que da forma al *web project template*.

4.3.3. Gestión de recursos

En la Figura 15 se presentan resaltados los elementos del modelo conceptual que son relevantes para la gestión de recursos asociados a tecnologías y *stack*.

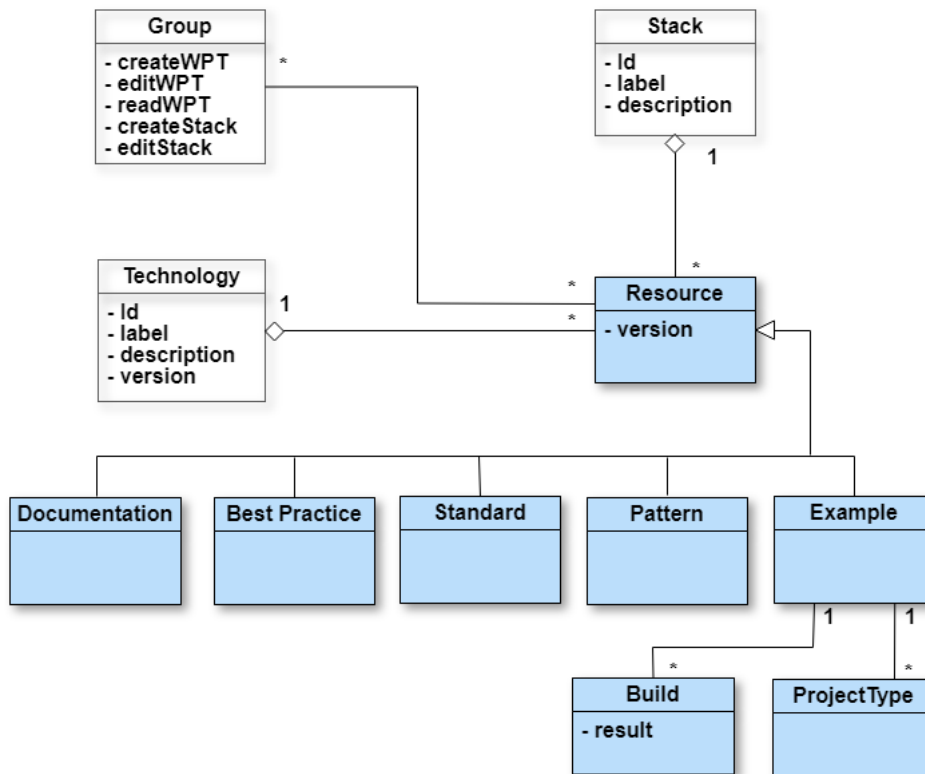


Figura 15: Gestión de recursos

A continuación se describen los elementos presentados:

- **Resource**

Se refiere a los recursos asociados a las tecnologías o *stacks*, los cuales podrían ser de los siguientes tipos:

- **Documentation**

Representan documentación sobre la tecnología o stack asociado.

- **BestPractice**

Representan buenas prácticas de uso de la tecnología o stack.

- **Pattern**

Representan patrones de diseño.

- **Example**

Representan tanto ejemplos en código de proyectos completos de aplicaciones web implementadas utilizando las tecnologías del *stack* asociado, así como simples ejemplos de uso de la tecnología. Estos ejemplos pueden contener casos de prueba automatizados. De esta forma, el sistema será capaz de ejecutar los casos de pruebas para detectar incompatibilidades o problemas en el código a modo de advertir a los usuarios.

- **ProjectType**

El tipo de proyecto determina el mecanismo de ejecución de las pruebas automáticas de los recursos en los servicios de integración continua, junto con la plataforma de ejecución donde ejecutar los casos de prueba. A modo de ejemplo, se pueden contar con un tipo de proyecto *Maven-jdk1.7*, que represente proyectos Maven donde los casos de prueba se deban ejecutar sobre el entorno de ejecución Java 1.7.

- **Build**

El concepto de Build se refiere a las ejecuciones de las pruebas automáticas realizadas sobre los recursos de tipo Example. Permite mantener un histórico de resultados de ejecución de las pruebas a modo de efectuar reportes.

4.3.4. Definición de formatos y versiones de tecnologías

En la Figura 16 se presenta resaltado los elementos del modelo conceptual que son relevantes para la definición de formatos y versiones de las tecnologías.

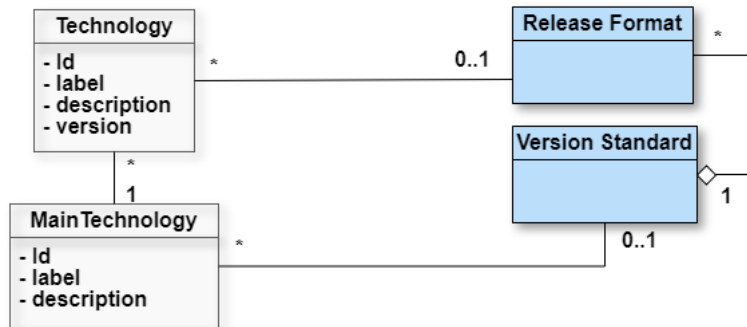


Figura 16: Versiones y formatos de tecnologías

A continuación se describen los elementos presentados:

- **VersionStandard**

Representa un estándar de versionado, el cual se asocia a una o más tecnologías principales. A modo de ejemplo, se puede contar con SemVer como estándar de versionado.

- **ReleaseFormat**

Representa el formato del estado de liberación dentro de un estándar de versionado específico. A modo de ejemplo, se puede contar con un estándar de versionado que incluya los siguientes formatos para los posibles estados, donde X e Y representan números:

- Versión en desarrollo: formato 0.X.Y
- Versión estable: formato 1.X.Y

4.3.5. Gestión de usuarios, roles y permisos

En la Figura 17 se presentan resaltados los elementos del modelo conceptual que son relevantes para la gestión de usuarios, roles y permisos.

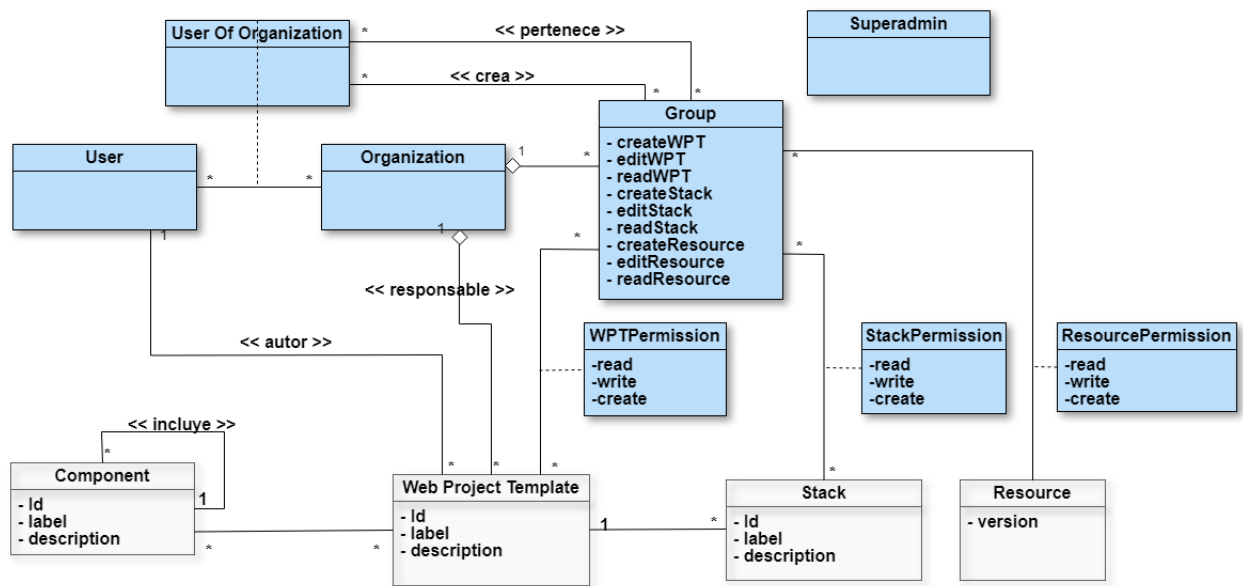


Figura 17: Gestión de usuarios, roles y permisos

A continuación se describen los elementos presentados:

- **SuperAdmin**

Representa a los usuarios administradores del Observatorio Tecnológico, los cuales son los únicos que pueden utilizar la aplicación de *BackOffice*.

- **User**

Representa a los usuarios de la aplicación *FrontOffice* del Observatorio Tecnológico.

- **Organization**

Corresponde a las organizaciones que pueden ser registradas por los usuarios.

- **Group**

Representa los grupos de usuarios que pueden definirse dentro de una organización. Es posible definir grupos a los cuales se le asignan diferentes permisos sobre algunas entidades del sistema dentro de la organización.

- **UserOfOrganization**

Representa la relación de pertenencia entre usuarios y organizaciones, donde los usuarios pueden pertenecer a varias organizaciones. Cada usuario dentro de una organización tiene asociado uno o más grupos de los que se hayan definido para dicha organización.

- **WPTPermission**

Modela el permiso de acceso que existe entre los miembros de un grupo de usuarios y las instancias de *web project template* de una organización.

- **StackPermission**

De forma análoga a WPTPermission, modela el permiso de acceso que existe entre un Stack y un grupo dentro de una organización.

- **ResourcePermission**

De forma análoga a WPTPermission, modela el permiso de acceso que existe entre un Resource y un grupo dentro de una organización.

4.4. Funcionalidades

En esta sección se listan las funcionalidades propuestas para el Observatorio Tecnológico.

4.4.1. Funcionalidades exclusivas de *BackOffice*

En esta sección se listan las funcionalidades exclusivas de la interfaz de *BackOffice*.

- **Login BackOffice**

Esta funcionalidad permite autenticarse en el sistema mediante la interfaz de *BackOffice* ingresando usuario y contraseña.

- **Alta, baja, modificación y listados (ABML) de entidades globales**

Esta funcionalidad permite crear, modificar, eliminar y listar componentes, tecnologías principales y ecosistemas.

- **Baja de contenido**

Esta funcionalidad permite a los administradores recibir denuncias de los usuarios sobre tecnologías, recursos, *web project templates* y *stacks*, teniendo la potestad de cambiarles el estado a *DEPRECATED* o eliminándolos por completo del sistema.

4.4.2. Funcionalidades de *FrontOffice* y *BackOffice*

En esta sección se listan las funcionalidades accesibles tanto desde la interfaz de *FrontOffice* como de *BackOffice*.

- **Visualización de reportes**

Esta funcionalidad permite visualizar los siguientes reportes:

- Puntuación de entidades del sistema. Por ejemplo, puntuación de *stacks*, tecnologías, entre otros.

- Cantidad de contenido en función del tiempo.
 - Cantidad de usuarios activos en el sistema en función del tiempo.
 - Estadísticas de resultados de test automáticos ante nuevas liberaciones de tecnologías.
 - Cantidad de recursos asociados a tecnologías y ecosistemas.
- **Registro de organización**

Esta funcionalidad permite a los usuarios registrar organizaciones. De esta forma, se crea una organización y el usuario que realiza la operación se incluye dentro del grupo de administradores de la misma.
 - **Crear usuario administrador**

Esta funcionalidad permite a los usuarios administradores de una organización, asignar permisos de administrador a otros usuarios. En caso de que el usuario a asignarle el rol de administrador no se encuentre registrado en el Observatorio Tecnológico, se envía una invitación vía *e-mail*. En dicha invitación, se accede a un enlace para completar el registro. Si el usuario ya está registrado y pertenece a la organización, el usuario administrador de la organización lo puede incluir dentro del grupo de administradores.
 - **Agregar colaborador**

Esta funcionalidad permite a usuarios administradores de una organización enviar invitaciones a otros usuarios para unirse a la misma como miembro de determinado grupo.
 - **Gestión de grupos dentro de la organización**

Esta funcionalidad permite a los usuarios administradores de las organizaciones, agregar, quitar o editar grupos de usuarios dentro de las mismas. Cada uno de ellos contiene un nombre único dentro de la organización, así como un conjunto de permisos (*Read, Write, Create*) sobre los *stacks*, recursos, *web project template* de la organización.
 - **Gestión de usuarios dentro de Organización**

Esta funcionalidad permite a los administradores de una organización, agregar, quitar o mover usuarios de grupos definidos para la misma.
 - **Aceptar invitación a unirse a organización**

Esta funcionalidad permite a los usuarios recibir invitaciones para unirse a grupos dentro de las organizaciones. Para esto, reciben un *e-mail* con un enlace para registrarse o autenticarse en el sistema. Una vez completado el proceso, pertenecen a la organización bajo el grupo al que se le haya designado por el administrador. Un

usuario puede pertenecer a varias organizaciones de forma simultánea, con distintos roles para cada una.

- **Login *FrontOffice***

Esta funcionalidad permite a los usuarios autenticarse en el sistema mediante la interfaz de *FrontOffice* ingresando *e-mail* y contraseña.

- **Recuperar contraseña**

Esta funcionalidad permite a los usuarios reiniciar su contraseña. Para esto, acceden a un enlace que se le envía vía *e-mail*.

- **Seleccionar espacio de trabajo**

Esta funcionalidad permite a los usuarios seleccionar un espacio de trabajo.

- **ABML Entidades Principales**

Esta funcionalidad permite crear, modificar, eliminar y listar tecnologías, recursos, etc.

- **Publicar entidad**

Esta funcionalidad permite a los usuarios cambiar el estado de un *stack*, recurso o *web project template* de *WORK IN PROGRESS* a *PUBLISHED*

- **Crear Web Project Template**

Esta funcionalidad permite, una vez seleccionado un espacio de trabajo y si se disponen de los permisos necesarios, crear un *web project template* seleccionando distintos componentes. Es posible elegir entre crear el recurso privado o público para la comunidad. También permite crear un *web project template* a partir de la clonación de otro.

- **Mantenimiento de estados en Entidades Principales**

Esta funcionalidad permite mantener un *web project template*, *stack* o recurso en estado *WORK IN PROGRESS* mientras se encuentra trabajando en el mismo.

Durante el estado *WORK IN PROGRESS*, los usuarios no pueden crear otras entidades a partir de ellas. A modo ejemplo, si se tiene un *web project template* en estado *WORK IN PROGRESS*, no es posible crear un *stack* partiendo de él hasta que se encuentre en estado *PUBLISHED*. Sin embargo, sí es posible visualizarlo y editarlo.

Finalmente, es posible cambiar el estado de un *stack*, recursos o *web project template* al estado *DEPRECATED* en caso de disponer de los privilegios necesarios.

- **Ver mis Entidades**

Esta funcionalidad ofrece al usuario una visualización de todas las entidades (*web project template*, *stack*, recursos) a las que tiene acceso, indicando el tipo de permiso otorgado.

- **Ejecutar pruebas de recursos**

Esta funcionalidad ofrece al usuario la posibilidad de ejecutar las pruebas automáticas configuradas en los recursos indicando la versión específica de las diferentes tecnologías utilizadas.

- **Wizard**

Esta funcionalidad permite a los usuarios crear un *stack* siguiendo un asistente o *wizard*. Partiendo de un *web project template* existente, el usuario puede seleccionar una o más tecnologías por cada componente del *web project template*, restringiéndose la elección de las mismas según las compatibilidades registradas y las preferencias ingresadas por el mismo.

- **Sistema de comentarios y ayuda**

Esta funcionalidad permite mantener, para las entidades *stack*, recursos y *web project template*, un foro de discusiones, reportes de problemas y ayuda, donde los usuarios pueden ingresar preguntas, brindar respuestas y puntuar respuesta de otros usuarios.

- **Calificar entidades**

Esta funcionalidad permite a los usuarios puntuar las entidades creadas por otros usuarios tanto dentro de las organizaciones a las que pertenece, como las creadas de forma pública para la comunidad.

- **Panel de noticias**

Esta funcionalidad permite a los usuarios acceder a un panel de noticias relacionadas al mundo del desarrollo web, tales como liberaciones de nuevas versiones asociadas a tecnologías en el Observatorio Tecnológico, buenas prácticas, tutoriales, entre otros.

- **Alertas sobre cambio de versiones**

Esta funcionalidad permite enviar notificaciones a los usuarios ante cambios de versiones de tecnologías, ejecutando los casos de prueba de recursos de tipo código, detectando aquellos que dejaron de funcionar.

4.4.3. Resumen de Funcionalidades

En base a las funcionalidades generales para los observatorios tecnológicos relevadas en la sección 3.2.1, se verifica que las mismas se satisfacen mediante alguna funcionalidad

propuesta. En el Cuadro 3 se presenta un resumen de las funcionalidades del Observatorio, indicando a través de qué interfaz, *FrontOffice* (FO) ó *BackOffice* (BO), se puede acceder a ella. Además se indica, para las funcionalidades propuestas, cuáles de ellas se corresponden con las funcionalidades generales.

Funcionalidades propuestas	FO	BO	Func. generales
Login <i>BackOffice</i>		X	
ABML Entidades globales		X	F1
Baja de contenido		X	
Visualización de reportes	X	X	F2 F4 F6
Registro de organización	X	X	
Crear usuario administrador	X	X	
Agregar colaborador	X	X	
Gestión de grupos dentro de organización	X	X	
Gestión de usuarios dentro de organización	X	X	
Aceptar invitación a unirse a organización	X	X	
Login <i>FrontOffice</i>	X		
Recuperar contraseña	X	X	
Seleccionar espacio de trabajo	X		
ABML entidades principales	X	X	F1
Publicar entidad	X	X	
Crear Web Project Template	X	X	F1
Mantenimiento de estados en entidades principales	X	X	F3
Ver mis entidades	X		F6
Ejecutar pruebas de recursos	X	X	F3
Wizard	X	X	F1 F4 F6
Sistema de comentarios y ayuda	X	X	F3
Calificar entidades	X	X	F3
Panel de noticias	X	X	F5 F6
Alertas sobre cambio de versiones	X	X	F3 F4 F5

Cuadro 3: Funcionalidades del Observatorio Tecnológico

4.5. Arquitectura Lógica del Observatorio Tecnológico

En esta sección se presenta una visión global del sistema describiendo la arquitectura lógica mediante diferentes vistas.

4.5.1. Vista Lógica

La Figura 18 presenta una arquitectura lógica del Observatorio dividida en capas.

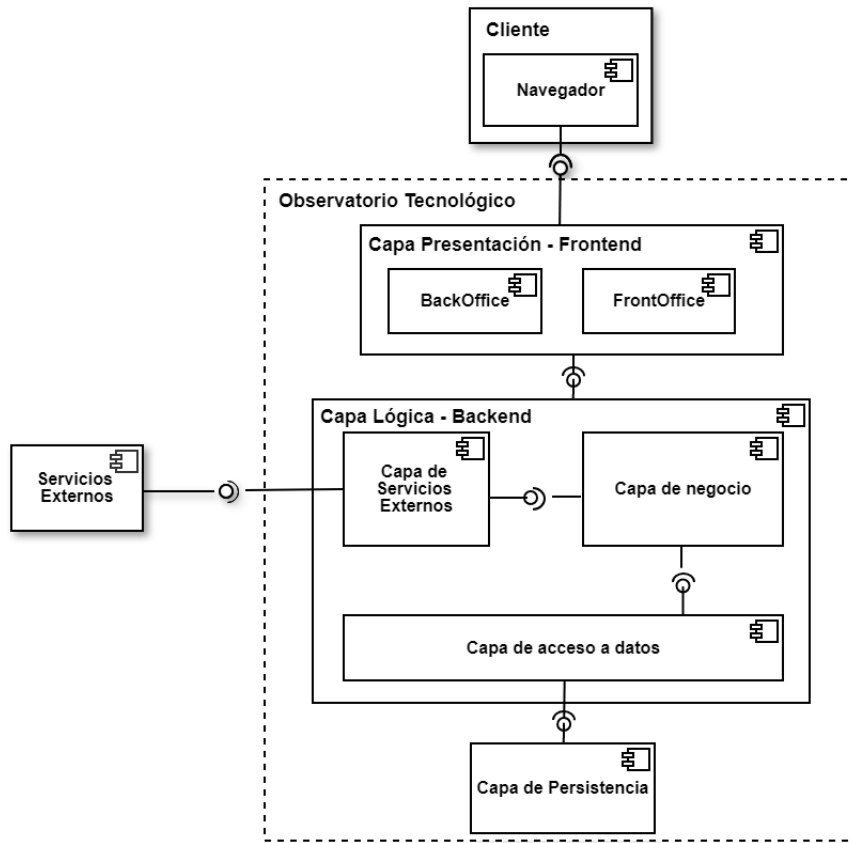


Figura 18: Vista lógica

La capa de presentación está compuesta por los componentes *BackOffice* y *FrontOffice* los cuales consumen una capa de servicios proporcionada por la capa lógica.

La capa lógica está compuesta por tres componentes: la capa de servicios externos, la capa de negocio y la capa de acceso a datos.

Finalmente, la capa de persistencia se encarga de almacenar toda la información recolectada por el Observatorio Tecnológico, tanto de usuarios, como de entidades.

4.5.2. Componentes de la vista lógica del Observatorio Tecnológico

En la Figura 19 se detallan los componentes de las diferentes capas que se presentan en la vista lógica.

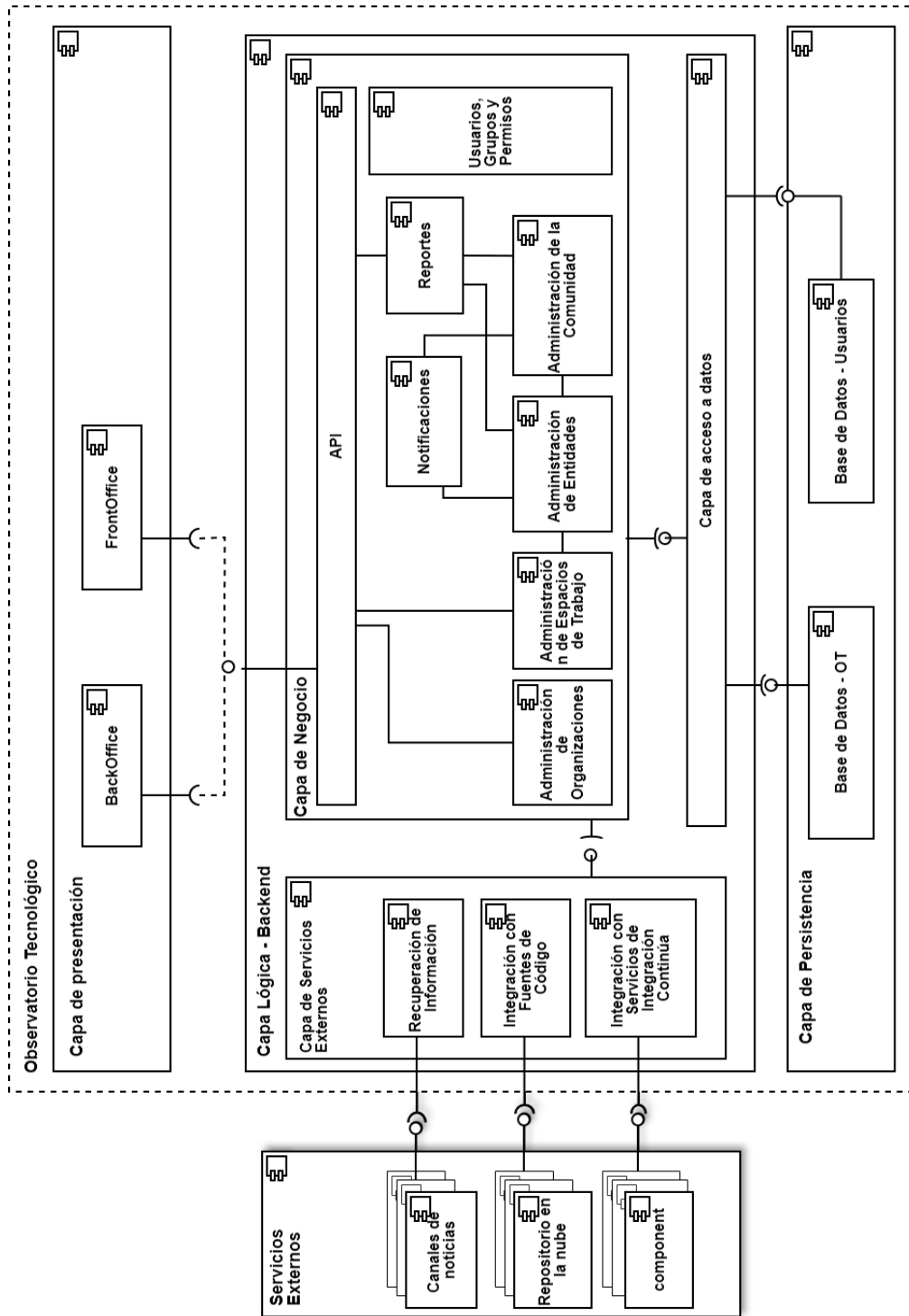


Figura 19: Componentes de la vista lógica

API

El componente API es el encargado de exponer los servicios proporcionados por la capa de negocio a las aplicaciones de *FrontOffice* y *BackOffice*.

Notificaciones

El componente de Notificaciones es el encargado de enviar notificaciones por *e-mail* a los usuarios, a modo de advertirles ante liberaciones de nuevas versiones de tecnologías que utiliza. También se encarga de notificar cuando se detecten incompatibilidades en los proyectos de los usuarios.

Reportes

El componente de Reportes es el encargado de la generación de todos los reportes concernientes al Observatorio Tecnológico. Para esto debe realizar consultas a la base de datos a través de la capa de acceso a datos.

Usuarios, Grupos y Permisos

El componente de usuarios, grupos y permisos se encarga de implementar los mecanismos de control de acceso de los usuarios sobre las entidades del Observatorio. Debe asegurar que un usuario solamente pueda realizar acciones sobre los recursos que tenga permiso en base a los grupos y roles definidos.

Administración de Organizaciones

El componente de Administración de Organizaciones implementa todas las tareas concernientes a la gestión y mantenimiento de las organizaciones, tanto la creación y asignación de usuarios a roles, como de grupos y permisos sobre las entidades.

Administración de Espacios de Trabajos

Es el encargado de proveer mecanismos para la administración del espacio de trabajo de los diferentes usuarios. De esta forma, cada usuario podrá contar con un espacio privado donde trabajar, manipulando las diferentes entidades con las que se cuenta en el Observatorio Tecnológico. A modo ejemplo, el usuario podrá definir nuevas tecnologías o versiones de ellas, almacenarlas en su espacio de trabajo, para en otro momento continuar con su edición.

Administración de Entidades

El componente de administración de entidades implementa toda la lógica de negocio asociada a la creación y mantenimiento de las entidades del Observatorio.

Administración de la Comunidad

El componente de Administración de la Comunidad se encarga de mantener los estados de visualización de las entidades del Observatorio, a modo de definir si un recurso es visible para todos los usuarios o sólo para la organización/usuario que la creó.

Capa de Acceso a Datos

El componente de Acceso a Datos provee mecanismos de acceso a la base de datos de forma transparente para las capas superiores.

Base de Datos

Este componente es el encargado de almacenar en base de datos toda la información recolectada por el Observatorio Tecnológico.

Base de Datos - Usuarios

El componente de Base de Datos de Usuario almacena la información de autenticación de los usuarios y organizaciones. También cuenta con la información necesaria para el manejo y administración de roles y permisos sobre los mismos.

Recuperación de Información

El componente de Recuperación de Información es el encargado de recuperar información de servicios externos, a través de los cuales puede detectar liberaciones de nuevas versiones de las tecnologías. A su vez, es la fuente de información a ser desplegada en el panel de noticias, recolectado información relevante a buenas prácticas, tendencias, nuevas tecnologías y demás.

Integración con Fuentes de Código

El componente de Integración con Fuentes de Código está destinado a manejar la integración con servicios de repositorios de código en la nube para que los usuarios puedan almacenar desde simples ejemplos de código, hasta proyectos completos implementados mediante una o más tecnologías.

Integración con Servicios de Integración Continua (CI)

El componente de Integración con Servicios de CI provee mecanismos de ejecución de pruebas automáticas orquestadas por el Observatorio. A través de este componente, el sistema puede ejecutar casos de prueba que se incluyan en los recursos alojados en los repositorios, a modo de detectar incompatibilidades ante alguna actualización de las tecnologías involucradas.

5. Instanciación

En esta sección se presenta una posible instancia de *web project template* y *stack* de tecnologías. Se parte de un *web project template* de ejemplo y se presenta un *stack* formado por una selección de tecnologías pertenecientes a un ecosistema específico. En el Apéndice A se presentan más ejemplos de instanciación utilizando otros ecosistemas.

5.1. Ecosistema Java

Java Enterprise Edition (Java EE) es una plataforma de desarrollo de aplicaciones empresariales. Provee soporte tanto para el desarrollo, implementación y administración de aplicaciones centradas en el lado del servidor, así como para la construcción de interfaces de usuario basadas en la Web [48].

5.1.1. Web Project Template

En la Figura 20 se presenta un ejemplo de *web project template*, el cual consta de los siguientes componentes:

- Capa de Presentación
- Capa de Servicios
- Enrutamiento
- Capa de Negocio
- Capa de Acceso a Datos
- Manejo de usuarios, autenticación y control de acceso
- Entorno de ejecución
- Herramientas de desarrollo
- Persistencia

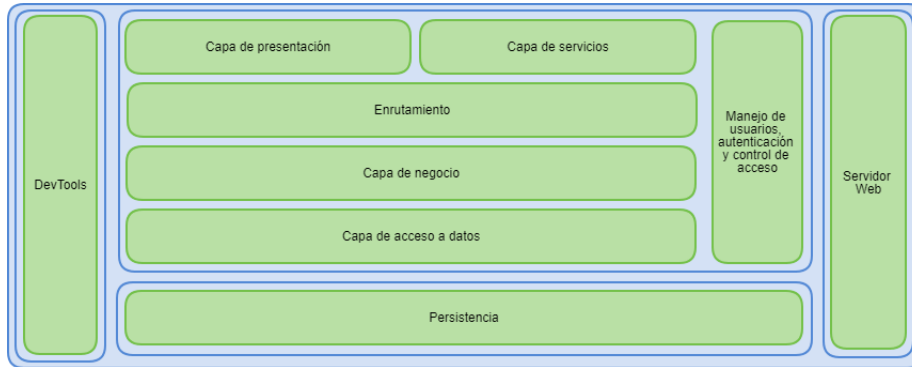


Figura 20: Web Project Template 1

Para este ejemplo se toma de base el ecosistema Java EE. En la Figura 21 se muestra una posible selección de tecnologías vinculadas a dicho ecosistema, sin considerar tecnologías para el componente de Enrutamiento.

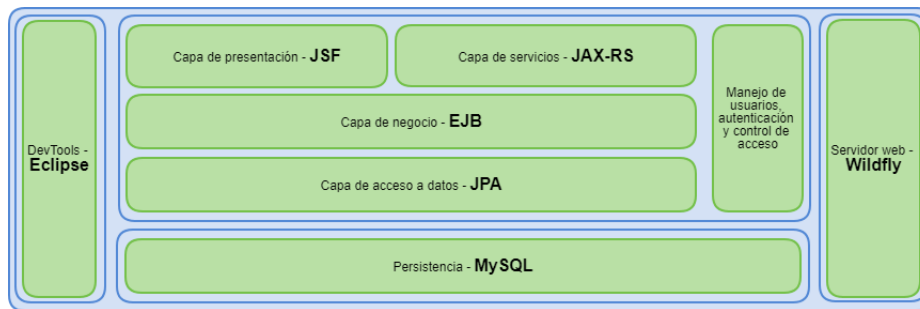


Figura 21: Stack para el ecosistema Java EE

5.1.2. Tecnologías

A continuación se detalla, para cada componente del *stack*, la tecnología asociada.

Capa de presentación

Java Server Faces (JSF) es una especificación incluida dentro de Java EE destinada a la implementación de capas de presentación del tipo MPA. Provee componentes de interfaz de alto nivel (paneles, menús desplegables, etc.) mediante etiquetas similares a las de HTML, las cuales son interpretadas y compiladas por JSF dando como resultado componentes HTML y JavaScript [49] [50].

Capa de servicios

JAX-RS es una especificación incluida en Java EE destinada al desarrollo de servicios web en aplicaciones Java construidas de acuerdo al estilo arquitectónico REST. La API

de JAX-RS ofrece anotaciones declarativas que permiten identificar los componentes de la aplicación, enrutar solicitudes a métodos apropiados en una clase seleccionada, extraer parámetros de una solicitud, entre otros [51].

Capa de negocio

Enterprise JavaBeans (EJB) 3.1 es la especificación de Java EE utilizada para implementar arquitecturas basadas en componentes del lado del servidor. Encapsulan la lógica del negocio de una aplicación implementando los métodos de la lógica del negocio, que pueden ser invocados por clientes remotos para acceder a los servicios importantes proporcionados por la aplicación [52].

Los EJB atacan los principales problemas que se presentan a la hora de desarrollar aplicaciones del lado del servidor, resolviendo aspectos relevantes a seguridad, transacciones, concurrencia, entre otros [52].

Los Sessions EJB son un tipo de EJB destinado a implementar la capa de negocio de las aplicaciones Java EE. Estos Session EJB atienden a un usuario a la vez, y pueden ser del tipo *stateful* (con estado), *stateless* (sin estado) y *singleton* [52].

Los Statefull Session Bean tienen la habilidad de mantener información relativa a la interacción entre un cliente y la capa de servicios entre distintos llamados a la capa de negocio. Los Stateless Session Bean no mantienen ninguna información relativa al cliente entre distintos llamados a la capa de negocio. Los Singleton Session Bean garantizan que solamente una instancia de *bean* será mantenida en la aplicación. De esta forma, se puede compartir una única instancia de *bean* entre todos los componentes de la aplicación [52].

La elección del tipo de EJB a utilizar para implementar la capa de negocio dependerá de los requerimientos de la misma [52].

Capa de acceso a datos

Java Persistence API (JPA) es un *Object Relational Mapping* (ORM)¹ provisto por Java EE para la gestión de bases de datos relacionales desde aplicaciones Java. Provee un lenguaje de consulta de bases de datos de alto nivel, mecanismos de definición de esquemas de bases de datos y demás [53].

Manejo de usuarios, autenticación y control de acceso

Java EE provee por defecto herramientas para el manejo de usuarios, grupos, roles y permisos sobre los recursos del sistema [54].

¹Modelo de programación que consiste en la transformación de las tablas de una base de datos en entidades que simplifican las tareas de acceso.

Servidor Web

Wildfly es un servidor de aplicaciones para aplicaciones Java que provee un ambiente de producción flexible y escalable [55].

Devtools - IDEs

Eclipse es un IDE que da soporte al desarrollo de aplicaciones Java EE. Provee un entorno de desarrollo que facilita el trabajo con tecnologías tales como JPA y JSF [56].

Persistencia

MySQL es un DBMS compatible con JPA [57].

5.2. Conclusiones

En base a la presentación de los ejemplos se logra una primera validación práctica de la solución, generando diferentes *stacks* a partir de distintos *web project template* con varios componentes en común. Además, se valida que los *web project templates* son capaces de soportar distintos *stacks* formados por diferentes conjuntos de tecnologías.

6. Implementación

En esta sección se presenta la implementación de un prototipo del Observatorio Tecnológico que da soporte a un subconjunto de las funcionalidades que se consideraron más relevantes, detallando las decisiones tomadas al momento de implementarlas.

En primer lugar, se presenta la descripción general del sistema que da soporte al prototipo del Observatorio. Luego se profundiza en la implementación del lado del servidor, detallando la arquitectura del sistema así como los principales componentes que dan soporte a las funcionalidades requeridas. Finalmente, se presenta la aplicación *frontend* que da soporte a la interacción de los usuarios con el Observatorio, tanto administradores como usuarios finales.

6.1. Descripción general

El prototipo consiste en una implementación parcial de la solución propuesta. En la Figura 22 se presenta el diagrama de la arquitectura lógica del Observatorio, remarcando los componentes del diagrama que se implementaron en el prototipo.

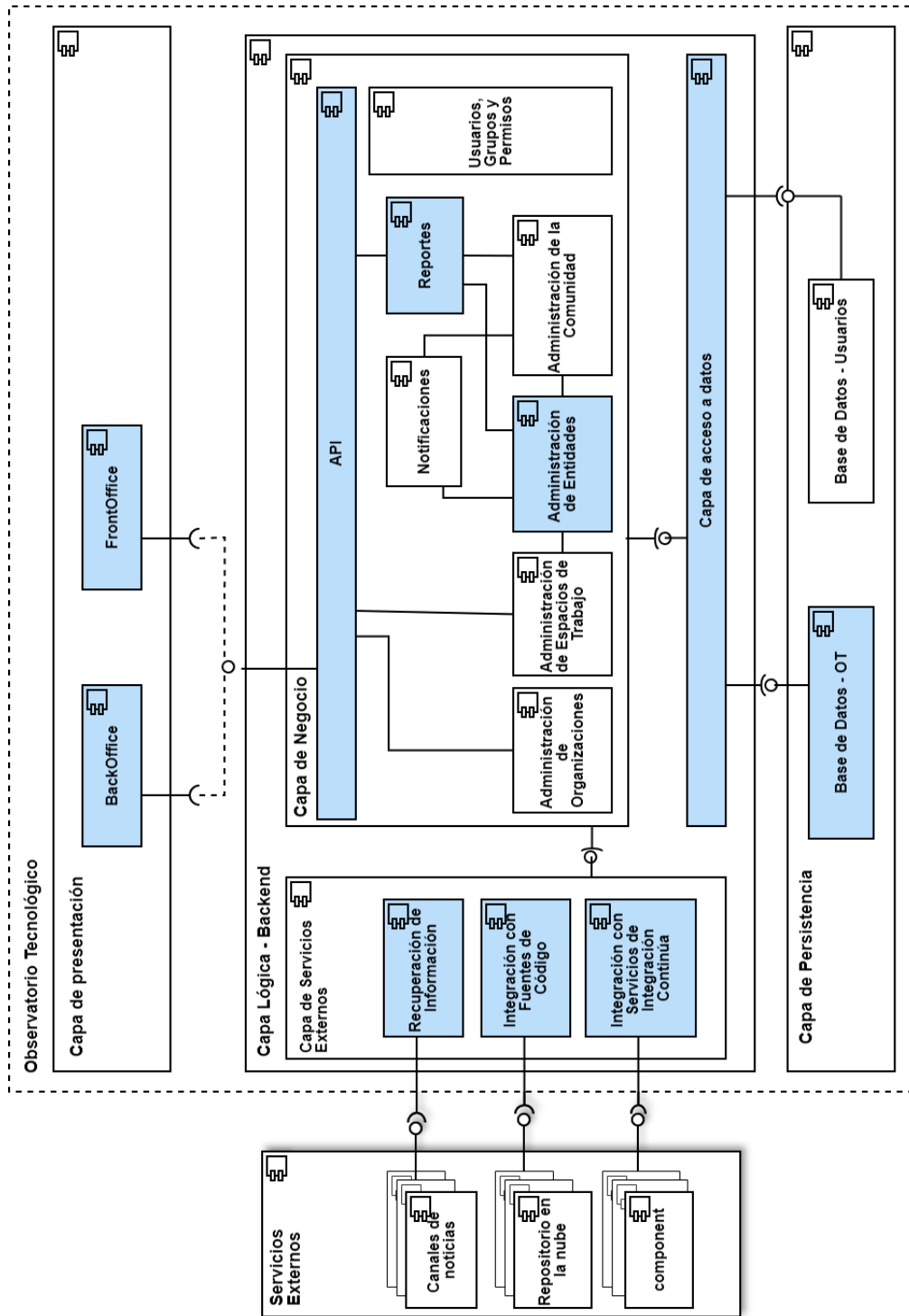


Figura 22: Componentes de capa lógica implementados

Dentro del componente de integración con fuentes de código se encuentran los componentes *GitIntegration* y *GitLabIntegration*, destinados a implementar la integración con servicios de repositorios GitLab.

El componente de integración continua incluye los componentes *JenkinsIntegration* y *GitLabCiIntegration* encargados de interactuar con los servicios de integración continua soportados por el Observatorio.

El componente de recuperación de información es implementado mediante el componente *RSSChannelIntegration* encargado de la extracción de noticias de distintos canales RSS.

Finalmente, el componente de administración de entidades se encuentra formado por un componente *Entities* encargado de proveer mecanismos de alta y mantenimiento de las entidades del Observatorio.

El prototipo de Observatorio tecnológico se conforma de un servidor *backend* compuesto por un proyecto Maven sobre la plataforma Java EE, una base de datos PostgreSQL y una capa de presentación compuesta por un proyecto NPM basado en la herramienta *React.js*, la cual incluye tanto la interfaz de *BackOffice* como *FrontOffice*. En la Figura 23 se presenta un diagrama de la arquitectura de la solución propuesta, resaltando los componentes mencionados anteriormente junto con las tecnologías utilizadas en cada caso.

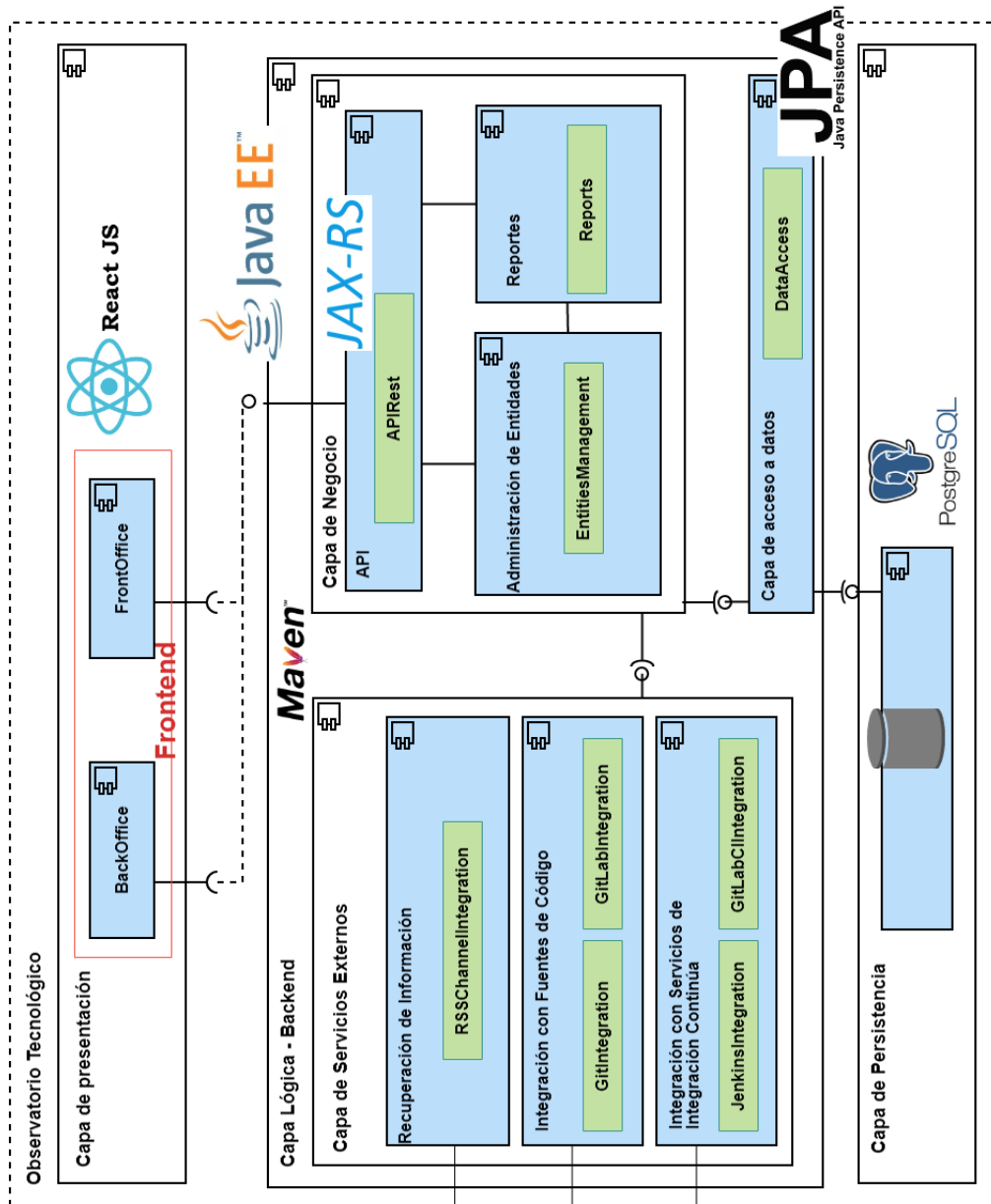


Figura 23: Arquitectura del Observatorio Tecnológico

6.2. Servidor backend Java EE

Como se mencionó anteriormente, el servidor *backend* consiste en una aplicación Java EE que se compone de una capa de negocios, una capa de servicios externos y una capa de acceso a datos.

La capa de negocio se conforma de un conjunto de componentes que dan soporte a distintas funcionalidades del Observatorio, exponiendo sus servicios a la aplicación *frontend* de *BackOffice* y *FrontOffice* mediante una capa de servicios.

La capa de servicios externos provee a la capa de negocio de mecanismos de integración con los servicios externos de repositorios de código en la nube, integración continua y canales de noticias.

Finalmente, en la capa de acceso a datos se encuentran los modelos que representan las entidades del Observatorio.

6.2.1. Capa de negocio

La capa de negocio se compone de los componentes *APIRest*, *EntitiesManagement* y *Reports*.

APIRest

El componente *APIRest* es el encargado de implementar la capa de servicios del Observatorio. Consiste de una API REST implementada mediante JAX-RS encargada de proveer a la aplicación *frontend* de servicios de acceso al Observatorio. Entre los más importantes se encuentran los servicios de creación de tecnologías con compatibilidades, *web project templates*, *stack*, configuración de tipos de proyectos y canales de noticias.

Por otro lado, dispone de un conjunto de servicios de notificación de resultados de casos de prueba destinados a la implementación de las integraciones con los servicios de integración continua.

EntitiesManagement

El componente *EntitiesManagement* es el encargado de implementar la lógica de negocio asociada a la creación y mantenimiento de todas las entidades del Observatorio, proveyendo métodos para crear, listar y actualizar entidades como componentes, tecnologías, *web project template*, *stack*, recursos, etc.

Reports

El componente *Reports* es el encargado de implementar los reportes provistos por el Observatorio.

6.2.2. Capa de servicios externos

La capa de servicios externos consta de cuatro componentes encargados de implementar distintas integraciones con servicios externos de repositorios de código, servicios de integración continua y canales RSS de noticias: *GitLabIntegration*, *GitLabCiIntegration*, *JenkinsIntegration* y *RSSChannelIntegration*.

GitLabIntegration

El componente *GitIntegration* es el encargado de proveer una interfaz de implementación de interacciones con servicios de repositorios Git en la nube. Los métodos provistos por esta interfaz son:

1. Crear una rama a partir de otra rama origen
2. Recuperar el contenido de un archivo de un repositorio
3. Realizar un *commit* de cambios sobre archivos sobre una rama
4. Eliminar una rama

De esta manera, mediante la implementación de dicha interfaz, se permite la integración con repositorios de otros servicios al Observatorio de manera extensible y transparente.

También provee interfaces para actualizar versiones de tecnologías sobre los archivos de dependencia de los proyectos realizando un *commit* sobre los mismos. A grandes rasgos, dicha interfaz recibe el contenido de un archivo de dependencias, un nombre de dependencia y una versión; y retorna el contenido del archivo actualizado. A los efectos del proyecto, se implementaron interfaces de actualización de versiones en archivos de dependencia correspondientes a proyectos Maven y NPM.

El componente *GitLabIntegration* es el encargado de implementar dicha interfaz proveyendo soporte para la integración con repositorios GitLab consumiendo la API provista por el mismo [58].

El Observatorio cuenta con una cuenta propia de GitLab, a la que los usuarios propietarios de los repositorios deben otorgar permisos de escritura de forma explícita. Luego, mediante un *access token* de la cuenta de GitLab, el Observatorio es capaz de acceder a todos los repositorios registrados por los usuarios en el mismo.

GitLabCiIntegration

GitLabCiIntegration es el componente encargado de implementar la lógica de ejecución de *pipelines* de los repositorios de los usuarios en servidores GitLab. Cuando un usuario

da de alta un recurso asociado a una tecnología o *stack web* formado por un proyecto de ejemplo, debe incluir información acerca del tipo de proyecto (NPM, Maven, etc.), rama dentro del repositorio donde se encuentra el código con los casos de prueba a ejecutar y ruta a los archivos de dependencia del proyecto dentro del repositorio.

El Observatorio puede ejecutar casos de pruebas sobre los proyectos de los usuarios, con la posibilidad de actualizar una o más versiones de tecnologías presentes en los archivos de dependencias indicados. La Figura 24 muestra la interacción del Observatorio con el servicio de integración continua de GitLab.

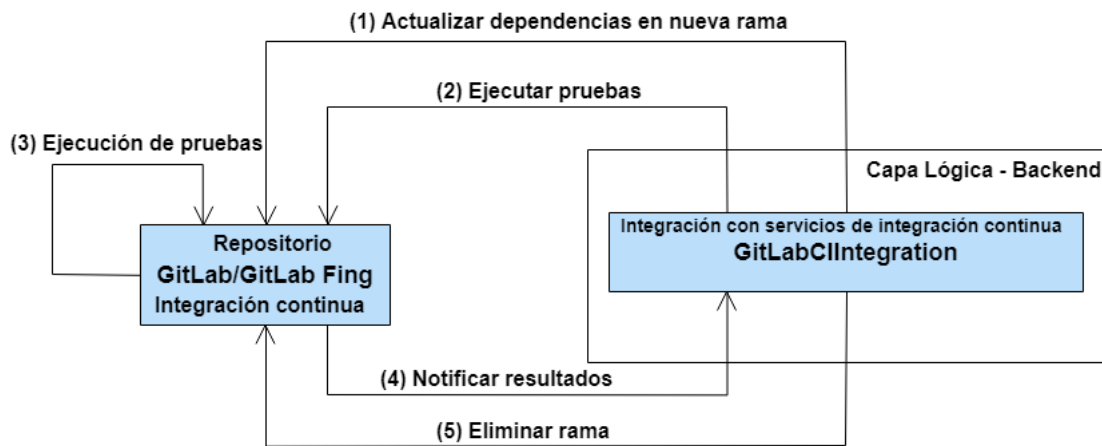


Figura 24: Interacción observatorio - GitLab

Para esto, se crea una rama en el repositorio donde se realiza un *commit* con el archivo de dependencias actualizado (1), para luego iniciar la ejecución de los casos de prueba sobre determinada plataforma de ejecución (2).

Dado que el proceso de ejecución de los casos de prueba puede demorar varios minutos y se realiza de forma asíncrona en los servidores de GitLab (3), los códigos de los *pipeline* deben contar con una lógica específica de notificación de los resultados de las pruebas al Observatorio. Para esto, la capa de servicios del Observatorio cuenta con un método específico donde los servicios de CI externos pueden notificar los resultados de las pruebas (4). De esta forma, el Observatorio puede identificar incompatibilidades de proyectos ante nuevas liberaciones de las tecnologías involucradas en los mismos. Una vez finalizadas las pruebas, se elimina del repositorio la rama creada al inicio del proceso (5).

JenkinsIntegration

Jenkins [59] es un servidor de integración continua de código abierto que provee mecanismos de ejecución de *pipelines* sobre proyectos alojados en repositorios.

El Observatorio cuenta con un servidor Jenkins propio donde puede ejecutar casos de prueba de los proyectos de los usuarios. *JenkinsIntegration* es el componente encargado de implementar la integración con el servidor de Jenkins propio del Observatorio, de forma análoga a como *GitLabCiIntegration* lo hace con GitLab.

En caso que un usuario desee ejecutar los casos de prueba de sus repositorios en el servidor Jenkins del Observatorio, el mismo deberá otorgarle permisos de lectura para poder clonar el repositorio. Para esto, el servidor Jenkins cuenta con una clave pública que el usuario deberá configurar en su repositorio para otorgarle los permisos requeridos.

De forma análoga a *GitLabCIIntegration*, *JenkinsIntegration* actualiza las versiones de las dependencias deseadas antes de ejecutar los casos de prueba. En la Figura 25 se presenta la interacción del Observatorio con el servidor Jenkins.

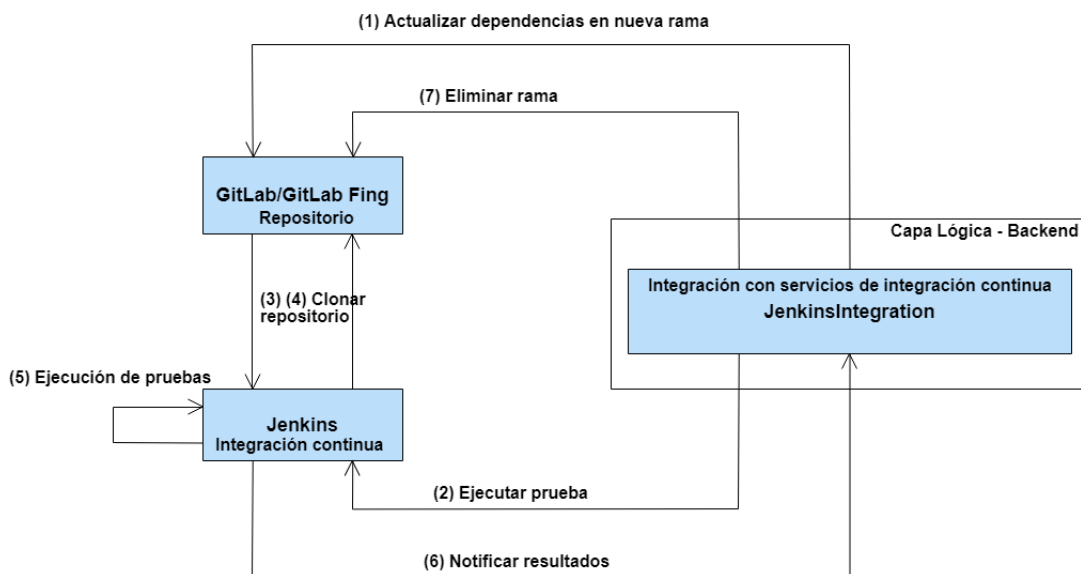


Figura 25: Interacción observatorio - Jenkins

RSSChannelIntegration

El componente *RSSChannelIntegration* es el encargado de implementar la integración con distintos canales RSS de noticias. Provee métodos para dar de alta canales RSS, especificando la URL del canal, junto con un código XSLT encargado de convertir el código

XML del canal al formato definido por el observatorio mediante un esquema XSD.

A modo de resumen, el código XSLT funciona como una capa traductora que establece una correspondencia entre el código XML de los canales RSS al esquema definido por el XSD del Observatorio. De esta forma, mediante un código XSLT adecuado, el Observatorio puede extraer de cualquier canal RSS los títulos de las noticias, fechas de publicación, imágenes, etc. También es posible definir el formato de fecha que utiliza el canal.

En la Figura 26 se muestra la interacción del Observatorio con los distintos canales RSS.

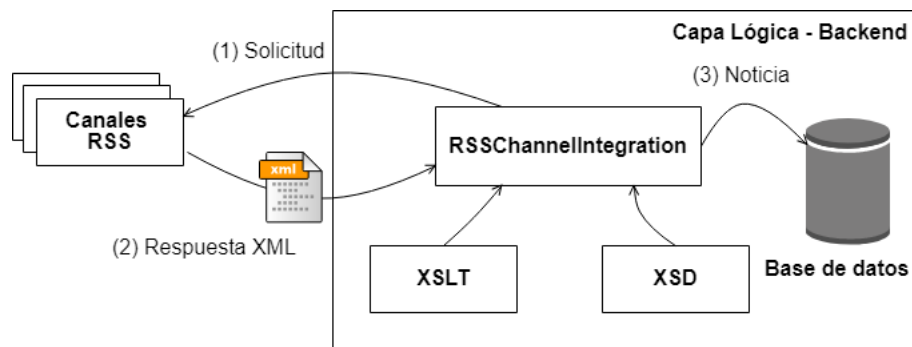


Figura 26: Interacción observatorio - canales RSS

En una primera instancia, el Observatorio realiza una solicitud al canal RSS para obtener las noticias (1) y el canal responde con el conjunto de noticias en un formato XML (2). Finalmente, el componente *RSSChannelIntegration* toma como insumo el XML del canal junto con el XSLT asociado al mismo, genera la información necesaria en el formato especificado por el XSD, para luego almacenar las noticias en la base de datos del Observatorio Tecnológico (3).

En el Apéndice B se detalla el esquema XSD definido y se presenta un XSLT de ejemplo para la extracción de noticias de un canal específico.

6.2.3. Capa de acceso a datos

El componente *DataAccess* es el encargado de implementar la capa de acceso a datos del Observatorio. Utiliza el ORM JPA para proveer a la capa de negocio de métodos de acceso a la base de datos de forma uniforme. Para esto, utiliza representaciones en alto nivel de los modelos de la base de datos mediante clases Java.

6.3. Capa de persistencia

La capa de persistencia consiste de una única base de datos PostgreSQL donde se almacenan todos los datos recolectados por el Observatorio.

6.4. Capa de presentación frontend

La capa de presentación del Observatorio consiste en una aplicación *frontend* implementada utilizando la herramienta React.js, compuesta por una interfaz de administración del Observatorio y una interfaz restringida para los usuarios comunes y organizaciones.

La versión destinada a los usuarios comunes y organizaciones provee un conjunto de reportes, listados de tecnologías, recursos, *web project templates* y *stacks* existentes, así como la posibilidad de dar de alta los mismos.

La versión de administración agrega listados y formularios de alta de ecosistemas, componentes, tecnologías principales, tipos de proyecto con configuración de *pipelines* y plataformas de ejecución, y configuración de canales RSS de noticias.

6.5. Implementación de funcionalidades

En esta sección se detalla la implementación de las principales funcionalidades consideradas en el prototipo.

6.5.1. Alta de tecnologías principales

Al momento de dar de alta una tecnología principal se puede indicar, junto con un nombre y el ecosistema al que pertenece, el estándar de versionado adecuado. Luego al momento de dar de alta una tecnología, se debe indicar el estado de liberación dentro del estándar asociado a la tecnología principal, lo que definirá una expresión regular a utilizar para validar el formato de la versión ingresada por el usuario.

6.5.2. Alta de versiones de tecnologías

Por otro lado, al momento de crear una tecnología perteneciente a determinada tecnología principal, el usuario tiene la posibilidad de configurar tanto compatibilidades como incompatibilidades con otras tecnologías existentes en el sistema, así como también indicar qué tecnologías son recomendadas en base a la tecnología en cuestión, a modo de ponderar positivamente su uso.

6.5.3. Recursos

Es posible adjuntar una serie de recursos asociados a las tecnologías, los cuales pueden ser de tipo documentación, o códigos de ejemplo. En caso de ser del tipo documentación, el usuario puede adjuntar un conjunto de enlaces a páginas web. En caso de ser del tipo código, debe ingresar la URL del repositorio.

6.5.4. Alta de tipos de proyecto

Los usuarios administradores tienen la posibilidad de dar de alta tipos de proyectos que indican el servicio de CI a utilizar, junto con el código del *pipeline* a ejecutar. En el caso de los tipos de proyecto que utilizan a GitLab como servicio de CI, los códigos deben seguir con el formato establecido por GitLab [24] a través de un archivo con formato YAML [32]. Este código debe contar con varias etapas que abarquen desde la instalación de dependencias del proyecto y ejecución de las pruebas unitarias, hasta la detección y manejo de errores o resultados satisfactorios. Por más información acerca de la configuración de los *pipelines*, se puede ver más detalle en el Apéndice C.

6.5.5. Ejecución de pruebas automáticas sobre recursos

En caso de que un proyecto disponga de casos de prueba automáticos y que el usuario desee que el Observatorio rastree la correctitud y compatibilidad del mismo, debe indicar qué tipo de proyecto es (lo que definirá el código del *pipeline* a ejecutar), en qué servicio de integración continua se deben ejecutar las pruebas (GitLab o Jenkins), rama donde residen los casos de prueba y, como se explicó anteriormente, otorgar al Observatorio los permisos requeridos.

Al momento de ejecutar las pruebas, el usuario puede ingresar las versiones específicas de las tecnologías utilizadas en el proyecto con las que se desea realizar las mismas, así como también indicar, mediante el tipo de proyecto, el servicio de integración continua junto con la versión de la plataforma de ejecución deseada. De esta forma, se pueden realizar pruebas y tomar decisiones en base a la compatibilidad del proyecto ante nuevas liberaciones de tecnologías, así como también visualizar el historial de ejecuciones junto con el resultado de los mismos.

6.5.6. Alta de componentes y Web Project Templates

La pantalla de creación de componentes permite indicar a qué otro componente pertenece el que se desea crear, a modo de establecer la estructura arborescente presentada anteriormente. Luego, la pantalla de creación de *web project templates* permite seleccionar los componentes que se desean incluir en el mismo, ofreciendo una representación gráfica del árbol representado. En la Figura 27 se muestra una captura de la pantalla de creación de

web project templates, donde se puede apreciar la AGPDAW de una posible aplicación web representada gráficamente mediante un árbol, donde se visualizan una serie de componentes (herramientas de desarrollo, persistencia, capa de presentación, capa lógica, despliegue).

Arquitectura de componentes

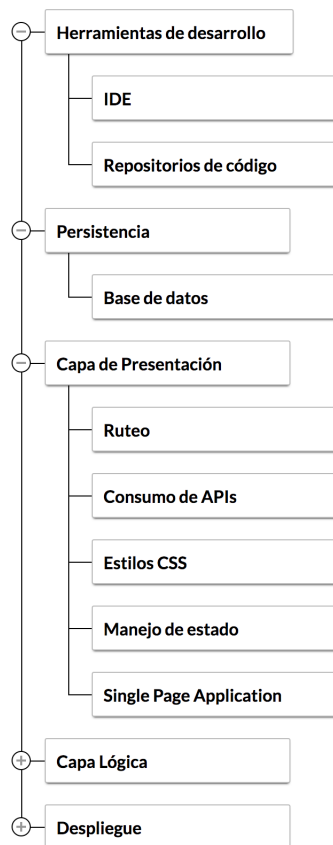


Figura 27: Representación arborescente de Web Project Template

6.5.7. Wizard

El Wizard de creación de *stacks* permite tomar un *web project template* existente y seleccionar las tecnologías que implementan el mismo alimentándose tanto de la información recolectada por el Observatorio, como de las preferencias ingresadas por el usuario. Para esto, el usuario puede indicar que desea seleccionar únicamente tecnologías de código abierto o en versiones finales, o simplemente filtrar por ecosistemas de preferencia.

A medida que el usuario selecciona las tecnologías para implementar cada componente

del *web project template*, la interfaz sugiere tanto el uso de otras tecnologías recomendadas, como también alerta de aquellas tecnologías las cuales no son compatibles con las seleccionadas previamente, basándose en la configuración de compatibilidad y sugerencia de las mismas.

Como resultado, se obtiene un conjunto de tecnologías compatibles seleccionadas por el usuario, las cuales dan forma a un *stack web* que implementa la AGPDAW representada por el *web project template* seleccionado.

Finalmente, el wizard ofrece la posibilidad de adjuntar recursos al *stack*, de forma análoga a las tecnologías.

6.5.8. Reportes

Se cuenta con una sección de reportes donde se muestran una serie de gráficas que presentan distintas métricas que podrían resultar de ayuda en la tarea de toma de decisiones.

Por un lado se presenta una gráfica donde se ilustran, a través de un gráfico de barras, los diez ecosistemas, tecnologías principales y tecnologías con mayor cantidad de recursos cargados en el Observatorio, brindando la posibilidad de inferir tendencias acerca de las herramientas mas utilizadas en el Observatorio.

Otro reporte interesante muestra las tecnologías principales, tecnologías y ecosistemas junto con la cantidad de ejecuciones de pruebas satisfactorias, fallidas y abortadas. Esta métrica puede dar una idea acerca qué tan robustas son las nuevas liberaciones de tecnologías en términos de retro compatibilidad con versiones anteriores, lo que podría ayudar notoriamente en la elección entre una tecnología u otra. Gracias a este reporte se podrían detectar tecnologías con tasa de errores muy altas, o tecnologías que no ofrecieron más soporte a partir de determinada versión.

6.5.9. Panel de noticias

El panel de noticias despliega un listado de todas las noticias extraídas de los canales RSS configurados en el observatorio. Mediante mecanismos de paginación, es posible acceder al historial de noticias extraídas en el pasado.

6.6. Despliegue

A modo de facilitar el despliegue del Observatorio en servidores, se configuró un método de despliegue del backend del Observatorio utilizando las herramientas Docker y Docker Compose.

6.6.1. Configuración del despliegue

El Observatorio cuenta con un archivo de configuración YAML *docker-compose.yml* donde se especifica la infraestructura de despliegue de los servidores necesarios junto con la configuración de la red privada y nombres de dominio de los servidores. A partir de este archivo, Docker es capaz de iniciar los contenedores, y configurar una red privada con intercomunicación entre los mismos, haciéndolos accesibles entre ellos [33].

La arquitectura de despliegue del Observatorio se conforma de tres contenedores Docker, tal como se indica en la Figura 28:

- OT_Backend
- Jenkins
- Postgres

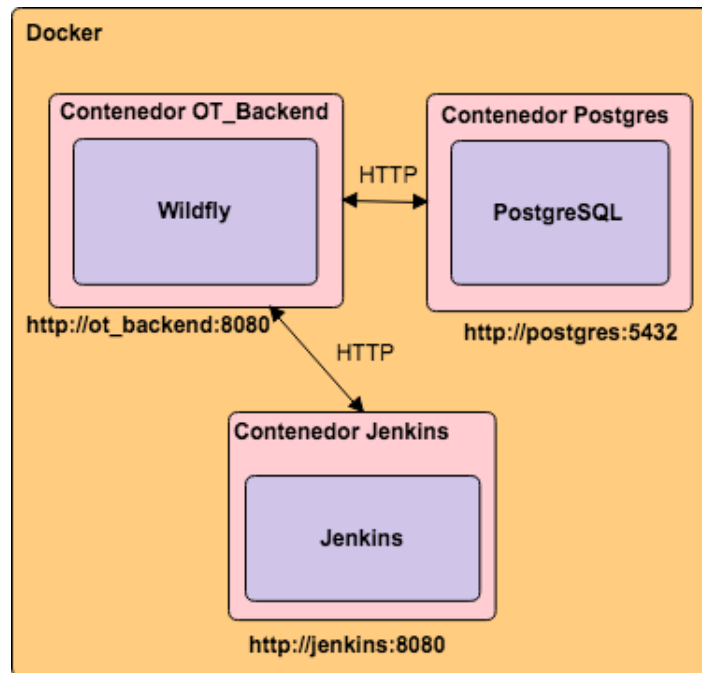


Figura 28: Arquitectura de despliegue del servidor *backend*

OT_Backend, construida en base a la imagen oficial de Wildfly [60], es un contenedor encargado de mantener el servidor de aplicaciones *Wildfly* en ejecución, exponiendo los servicios del backend del Observatorio.

Jenkins es un contenedor basado en la imagen oficial de Jenkins [61] que ejecuta el servidor de integración continua propio del Observatorio.

Finalmente, el contenedor *Postgres*, basado en la imagen oficial de PostgreSQL [62], es el encargado de ejecutar el servidor de base de datos del Observatorio.

Por otro lado, la aplicación frontend React fue construida utilizando la herramienta *create-react-app*[63], la cual permite crear un proyecto React preparado para ejecutar compilaciones del código fuente sin necesidad de lidiar con configuraciones de bajo nivel. Una vez compilado el proyecto, es posible desplegarlo en producción en servidores web estáticos.

6.6.2. Puesta en producción

A modo de probar el despliegue del Observatorio, se utilizaron los servicios en la nube provistos por Amazon Web Services [64], en particular los servicios *Elastic Compute Cloud* (EC2) [65] y *Simple Storage Service* (S3).

El servicio EC2 ofrece acceso a una máquina virtual Linux en la nube. De esta forma, se instalaron los servicios Docker y Docker Compose a modo de ejecutar los contenedores de Docker presentados anteriormente.

Por otro lado, el servicio S3 ofrece almacenamiento de archivos en la nube, con la posibilidad de servirlos a través del protocolo HTTP. De esta forma, se alojó la aplicación *frontend* en un espacio de almacenamiento con permisos de lectura públicos, haciéndola accesible desde Internet.

En la Figura 29 se presenta un diagrama de la arquitectura de despliegue del Observatorio en los servicios de Amazon Web Services.

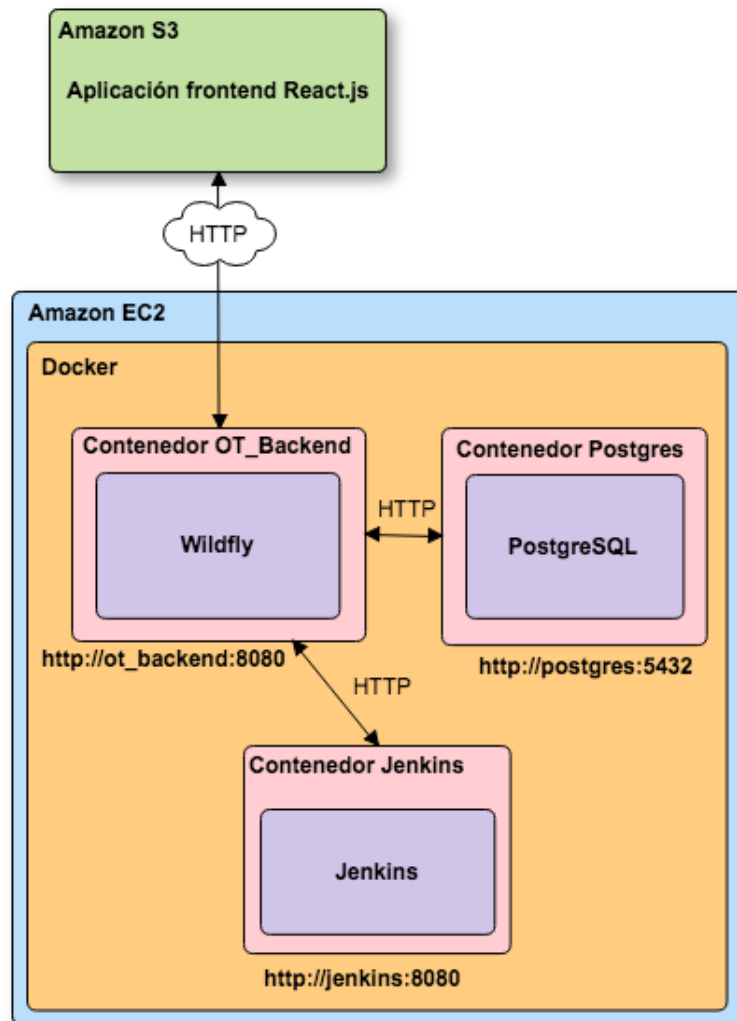


Figura 29: Arquitectura de despliegue en Amazon Web Services

6.7. Decisiones tomadas

En esta sección se detallan las decisiones tomadas a la hora de implementar las distintas funcionalidades del prototipo, presentando ventajas y desventajas.

6.7.1. Tecnologías

Se decidió implementar el servidor *backend* utilizando la plataforma Java EE dado un requerimiento del proyecto donde se solicitaba el uso de la misma. Por otro lado, la decisión de implementar la capa de presentación correspondiente a las interfaces de *BackOffice* y

FrontOffice mediante la librería React.js se basó no solo por el conocimiento de la herramienta por parte de los desarrolladores, sino a modo de seguir las tendencias actuales en términos de desarrollo de aplicaciones *frontend*.

6.7.2. Servicios de repositorios de código

La decisión de implementar la integración con GitLab y no con otros servicios de repositorios de código se resume en que el mismo provee un servicio de repositorios privados en la nube de forma gratuita. A su vez, ofrece la posibilidad de alojar instancias privadas de GitLab en servidores propios, tal como cuenta la Facultad de Ingeniería a disposición de todos los estudiantes y docentes. De esta forma, resultó conveniente implementar una integración que soporte cualquier tipo de servidor GitLab, y en particular, el servicio GitLab de la Facultad.

6.7.3. Servicios de integración continua

GitLab cuenta con un servicio de integración continua propio que ofrece la posibilidad ejecutar *pipelines* en los repositorios lo que permite, por ejemplo, ejecutar los casos de prueba que los proyectos alojados dispongan.

El método que provee GitLab para acceder a un repositorio a través de su API requiere de un *access token* generado por el propietario del mismo. Un inconveniente que esto presenta es que con dicho *access token*, se dispone de permisos de acceso sobre todos los repositorios del usuario. Es decir, no es posible restringir el *access token* para que solamente provea acceso a un repositorio específico. De esta forma surgió la necesidad de que el Observatorio disponga con una cuenta propia de GitLab a la que los usuarios otorgan permisos de forma explícita. Dado que se necesita de permisos de escritura en el repositorio para realizar las acciones requeridas, es necesario que el propietario del repositorio otorgue dichos permisos a la cuenta propia del Observatorio.

La razón por la que se decidió instalar un servidor propio de Jenkins como alternativa a GitLab es que este último provee, en su versión gratuita, una cantidad limitada de recursos para ejecución de *pipelines*, mientras que Jenkins no cuenta con dicha limitante.

La principal desventaja que presenta Jenkins contra GitLab reside en que el primero requiere de configuración manual para poder soportar nuevas versiones de plataformas de ejecución, a diferencia de GitLab, donde la versión de la plataforma a utilizar se especifica en el código del *pipeline* a ejecutar, teniendo la posibilidad de probar con nuevas versiones a demanda e incluir nuevos códigos de *pipeline* directamente desde la interfaz de administración del Observatorio. A modo de ejemplo, si se desea soportar una nueva versión de Java EE en Jenkins, es necesario instalar dicha versión en el servidor de Jenkins y configurarlo

para que los *pipeline* puedan utilizarla. Si en cambio se desea ejecutar un *pipeline* en GitLab con dicha versión de Java, basta con crear un nuevo tipo de proyecto en el Observatorio con un *pipeline* que indique la versión deseada.

6.7.4. Tipos de proyecto soportados

A los efectos del prototipo, se implementaron mecanismos de ejecución de pruebas en los servicios de integración continua y métodos de actualización de versiones de dependencias únicamente para proyectos NPM y Maven. Dado que el prototipo fue implementado utilizando la plataforma Java EE y la herramienta React.js utilizando los tipos de proyectos Maven y NPM respectivamente, a los efectos del proyecto resultó conveniente priorizar el soporte a dichas herramientas.

6.7.5. Recuperación de noticias

La gran mayoría de portales de noticias y *blogs* cuentan con canales RSS para la extracción de la información, por lo que de esta forma se decidió implementar el componente *RSSChannelIntegration* encargado de extraer, de distintos canales RSS, noticias relacionadas al mundo de desarrollo de aplicaciones web.

Si bien existen especificaciones de estándares de canales RSS [66], se observó que muchos de ellos no los siguen estrictamente, dificultando la extracción de la información. Por dicha razón, se realizó una especificación propia del Observatorio tal que, mediante la utilización de XSD y XSLT, sea posible extraer la información independientemente del formato del canal.

7. Caso de estudio

Esta sección tiene por finalidad clarificar cómo se usaría el Observatorio Tecnológico en un contexto real.

El caso de estudio presenta la realidad de una empresa perteneciente al rubro del desarrollo de software, dedicada al mantenimiento y venta de productos de software propios, a la cual le surge la oportunidad del desarrollo de un producto a medida para un cliente específico. A continuación se introducen algunas de las etapas del proceso de adopción del Observatorio por parte de la empresa:

- Motivación para formar parte del Observatorio Tecnológico.
- Búsqueda de soluciones de otras empresas a la problemática que afronta.
- Análisis de tecnologías, tendencias y toma de decisiones.
- Generación de la solución propia de la empresa brindado un *stack* de tecnologías.
- Aporte de conocimientos a la comunidad del Observatorio Tecnológico.
- Utilización de las funcionalidades de integración continua para mantener actualizados los proyectos.

7.1. Descripción del problema

Se considera una empresa de desarrollo ficticia líder en el comercio del software y la tecnología de la información en Uruguay denominada InspiraSoft. La empresa uruguaya ha comenzado a tener proyectos de trabajos con empresas grandes que cuentan con un gran número de funcionarios, como es el caso de una empresa de venta de insumos informáticos que quiere extender su mercado a través de la venta de sus productos por internet mediante una plataforma de *e-commerce*.

InspiraSoft no se dedica al desarrollo de software a medida, sino que dispone de una variada cantidad de productos desarrollados años atrás, por lo que no suelen trabajar con las últimas tecnologías y tendencias del mercado. A pesar que dentro de los productos desarrollados y vendidos por InspiraSoft no se encuentra ninguno de similares características a la plataforma web *e-commerce* requerida, InspiraSoft logra ganar la licitación abierta por la empresa de venta de insumos informáticos. Es así que la directiva de InspiraSoft decide contratar un grupo de desarrolladores de software con el fin de comenzar el desarrollo de la plataforma web de *e-commerce*.

La idea fundamental que se les planteó a los desarrolladores es que no es necesario el uso de las mismas herramientas con las se habitúa a trabajar en la empresa, aunque es

de interés estar alineado a los lenguajes de programación que actualmente se utilizan en InspiraSoft. No obstante, de probarse que otras tecnologías son más adecuadas, esta última consideración no debería ser una limitante para la selección que de soporte a la solución.

Para lograr su cometido, InspiraSoft decide registrarse como organización del Observatorio Tecnológico y a sus empleados como usuarios pertenecientes a la misma. Este observatorio, que alguna vez comenzó con muy poca información, no solo les será de utilidad a los nuevos desarrolladores de InspiraSoft, sino que con su aporte se nutrirá aún más con sus nuevas experiencias, y en particular, con los que se hagan de forma pública hacia la comunidad.

Luego de la etapa de relevamiento de requerimientos con el cliente, se obtienen los principales requerimientos:

- Interfaz web de administración de la plataforma de *e-commerce* donde se gestionan los productos ofrecidos.
- Reportes de ventas y finanzas de la empresa
- Gestión de *stock*
- Interfaz web centrada en la experiencia del usuario donde los mismos puedan buscar productos, agregar a un carrito de compras y ejecutar la misma mediante integraciones con procesadores de pago.
- Acceso *offline* a la tienda.

La Figura 30 representa los requerimientos planteados anteriormente.

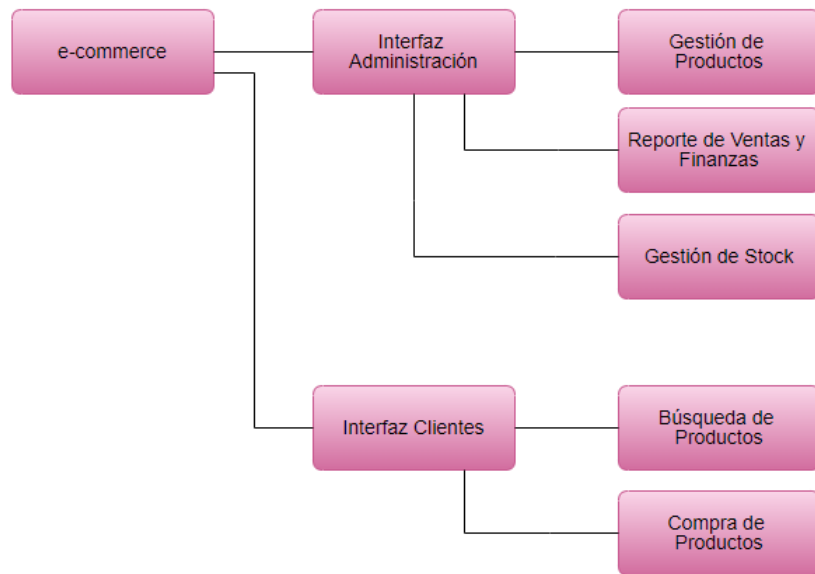


Figura 30: Esquema de requerimientos para *e-commerce*

7.2. Funcionalidades relevantes para la empresa

El Observatorio Tecnológico cuenta con un diverso conjunto de información que puede resultar relevante para los desarrolladores de InspiraSoft a la hora de brindar solución a los requerimientos, ya que cuenta con un conjunto de reportes que permiten a los usuarios inferir tendencias y determinar tecnologías más utilizadas, tal como se muestra en la Figura 31.



Figura 31: Reportes Observatorio Tecnológico

Por otro lado, se cuenta con un conjunto de documentación, artículos y buenas prácticas de las diferentes tecnologías. En la Figura 32 y la Figura 33 se muestran dos ejemplos de documentación, donde la primera refiere al estándar de codificación de Java y la segunda a un conjunto de tutoriales recomendados de React. A su vez, se puede acceder a documentación de otros proyectos de carácter similar (que eventualmente podría ser igual al que se necesita) y que permita la evaluación de los riesgos de aplicar dicha solución en base a la documentación que se encuentra en el Observatorio.

Información del recurso

Nombre del recurso

Estándar de codificación Java



Versión

13.05.8

Descripción


Documento de estándar de codificación del lenguaje de programación Java

Links

<https://www.oracle.com/technetwork/java/index-135089.html>

Figura 32: Documentación - Estándar Codificación Java

Información del recurso

Nombre del recurso
Tutoriales de React 

Versión
Ingrese una versión

Descripción
Diversos tutoriales de react

Links

<https://reactjs.org/tutorial/tutorial.html>

<https://reactforbeginners.com/>

<https://advancedreact.com/>

Figura 33: Documentación - Tutoriales React

7.3. Primeros pasos con el Observatorio

El grupo de desarrolladores comienza a utilizar el Observatorio en busca de soluciones existentes, encontrando así la publicación de un *stack* de una organización (FingSoft) que implementó una plataforma de *e-commerce* similar que, en principio, podría satisfacer las necesidades del cliente. El *stack* consiste en un desarrollo orientado íntegramente al ecosistema Java EE, por lo que tanto la interfaz de usuario como los servicios están implementados

utilizando tecnologías de dicha plataforma. Esto parece conveniente dada la experiencia que los desarrolladores de InspiraSoft ya poseen en Java, por lo que se disponen a investigar el *stack* junto con el *web project template* asociado. Entre los recursos asociados al *stack*, se encuentra un proyecto de ejemplo alojado en un repositorio público de GitLab. Dado que dicho proyecto cuenta con un rico historial de ejecución de pruebas automáticas con resultados alentadores, donde prácticamente no se encuentran resultados fallidos ante nuevas liberaciones de las tecnologías involucradas, deciden clonar el repositorio para realizar una prueba de concepto.

A sorpresa de los desarrolladores, se encontraron con una capa de presentación compuesta por una *Multiple Page Application* con limitaciones de rendimiento y sin soporte de acceso *offline*, evidenciándose una pobre experiencia de usuario. Dado que entre los requerimientos del cliente se destaca la necesidad de contar con una capa de presentación orientada a la experiencia de los usuarios, se deciden a continuar la investigación dentro del Observatorio en busca de otras soluciones, centrando su análisis en el *web project template* asociado al *stack*, y en particular, al componente de Capa de Presentación.

7.4. Primeras conclusiones

De esta forma, concluyen que las tendencias actuales en torno a la construcción de capas de presentación plantean dos corrientes generales según las características deseadas para el cliente: *Multiple Page Application* y *Single Page Application*, como se muestra en la Figura 34.

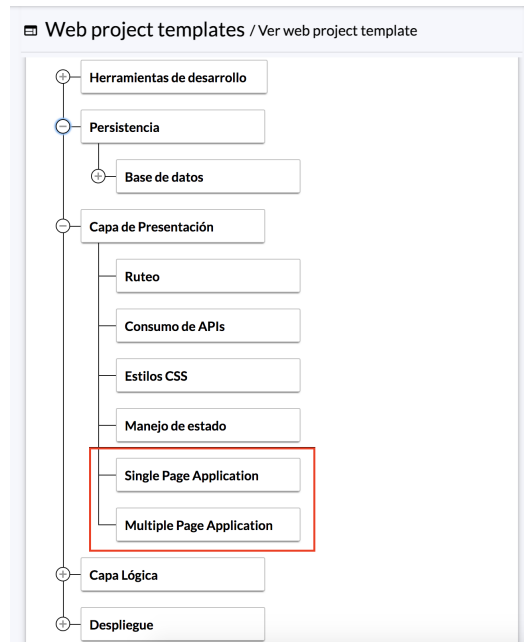


Figura 34: Web Project Template - Componentes de Capa de Presentación

En base a esto, los desarrolladores investigan las tecnologías asociadas a cada uno de estos componentes permitiéndoles relevar ventajas y desventajas, llegando a la conclusión que las tecnologías que más se adecuan a la resolución de sus requerimientos son aquellas que se encuentran asociadas al componente *Single Page Application*.

7.5. Ejecución del Wizard

El equipo de InspiraSoft procede a ejecutar el Wizard de creación de *stacks* a modo de crear un *stack* de tecnologías que den solución a la plataforma a desarrollar, tomando como base el mismo *web project template* utilizado por FingSoft.

Dado que el *stack* creado por FingSoft satisface en gran medida las necesidades de InspiraSoft, deciden seleccionar las mismas tecnologías, a excepción de aquellas asociadas al componente de Capa de Presentación. Puesto que el conjunto de tecnologías encargadas de implementar la capa de presentación deben pertenecer al componente *Single Page Application*, se selecciona dicho componente en el Wizard para visualizar las opciones de tecnologías disponibles, obteniendo varios candidatos posibles: React.js, Angular.js y Vue.js, tal como se muestra en la Figura 35.

☰ Wizard - Nuevo stack

Single Page Application ^

Buscar tecnologías

	Nombre	Dependencia/paquete	Versión	
<input type="checkbox"/>	React	react	16.5.2	←
<input type="checkbox"/>	jQuery	jquery	3.2.1	
<input type="checkbox"/>	Angular	angular	1.4	←
<input type="checkbox"/>	Vue.js	vue	1.2	←

Figura 35: Wizard - Tecnologías candidatas para componentes SPA

Como se muestra en la Figura 36, entre las tres opciones posibles, React.js es la tecnología que cuenta con una mayor cantidad de recursos, al igual que un número mayor de pruebas ejecutadas que dieron un resultado satisfactorio. De esta forma, en base a la cantidad de recursos asociados y a los reportes presentados en el Observatorio, deciden que React.js es la mejor opción.



Figura 36: Reportes - Tecnologías candidatas

La Figura 38 muestra la selección de la tecnología elegida. Una vez seleccionada dicha tecnología, el equipo de desarrollo descubre una nueva serie de tecnologías recomendadas por el Wizard: React Router, Redux y Styled Componentes, por lo que deciden también adjuntarlas al *stack*. En la Figura 37 se muestra un ejemplo de una de las recomendaciones. Adicionalmente, descubren que jQuery, tecnología con la que habitualmente trabajan desde hace años, es declarada incompatible con React.js, como muestra la Figura 38. Por dicha razón, deciden no incluirla en el *stack*.

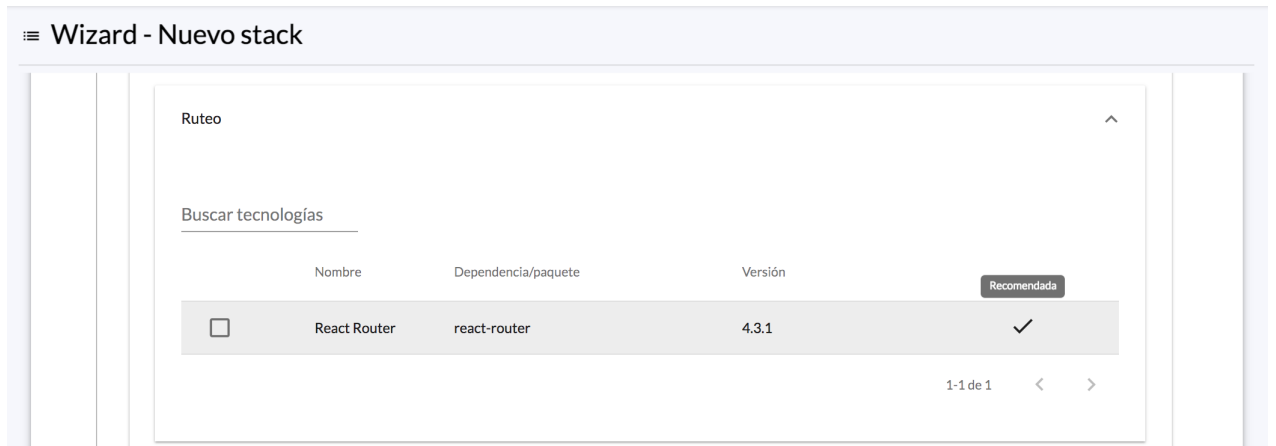


Figura 37: Wizard - Tecnología recomendada para React.js

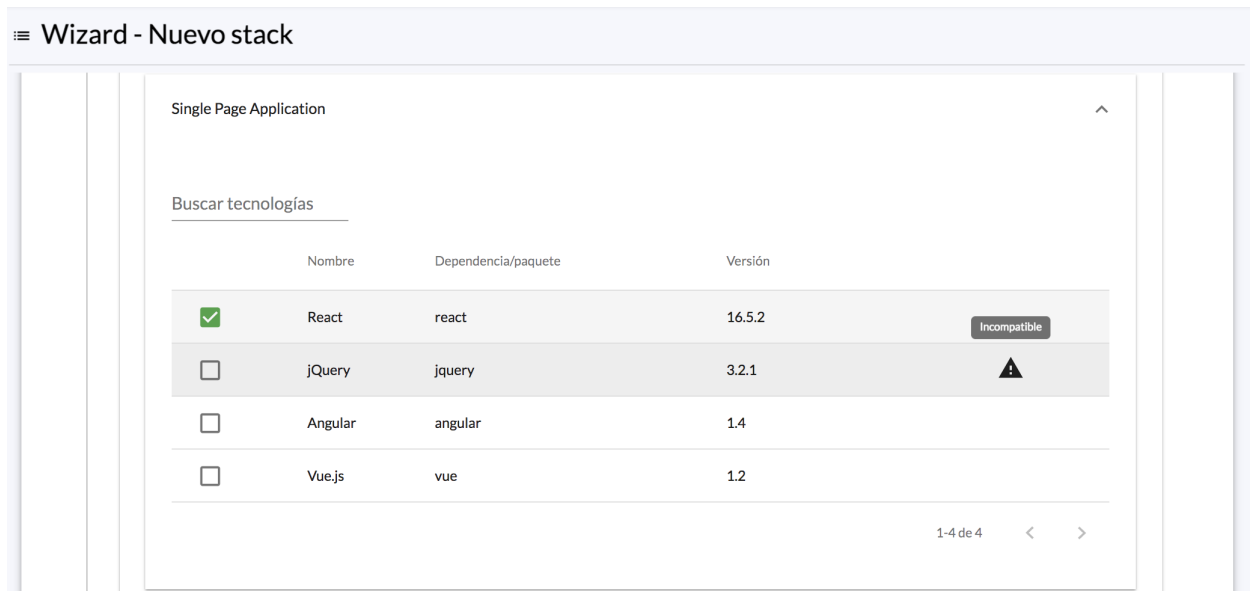


Figura 38: Wizard - Selección de Tecnología React.js

Una vez finalizada la selección de tecnologías para el *stack*, lo publican de forma privada a la Organización en estado *Work In Progress*, debido a que es posible que durante el proceso de desarrollo del proyecto surjan requerimientos nuevos que requieran de adjuntar o remover tecnologías. De esta forma, se obtiene como resultado un *stack* formado por tecnologías pertenecientes a dos ecosistemas: JavaScript para el *frontend* construido con React.js, y Java EE para el *backend*.

Finalmente, el grupo de desarrolladores recupera toda la documentación adjunta a las tecnologías seleccionadas a modo de formarse y prepararse para el desarrollo de la plataforma.

7.6. Aporte a la comunidad

Luego de finalizado el desarrollo de la plataforma, la Organización de InspiraSoft decide realizar un aporte a la comunidad del Observatorio, liberando el *stack* de forma pública a la comunidad, junto con el conjunto de tecnologías que dio solución al proyecto. Dicho *stack*, no solo resultará de utilidad para la comunidad, sino que disminuirá la curva de aprendizaje para nuevos desarrolladores de la empresa que deban trabajar en nuevas funcionalidades que surjan en el futuro.

Por otro lado, publican un recurso asociado al *stack* compuesto por un prototipo de la plataforma de *e-commerce* desarrollada, de modo que los usuarios del Observatorio puedan realizar pruebas de concepto del *stack* de InspiraSoft, tal como ellos lo hicieron con el *stack* de FingSoft.

7.7. Nueva problemática

Luego de finalizado el desarrollo de la plataforma para el cliente y con el proyecto en producción desde hace varios meses, un desarrollador de InspiraSoft propone actualizar la versión tanto de React.js como del entorno de ejecución Java. Como resultado, se encontraron que ante las actualizaciones realizadas, las pruebas fallan.

Esto presenta una nueva problemática a InspiraSoft, la cual puede ser mitigada mediante el uso de otra funcionalidad que brinda el Observatorio, no utilizada por InspiraSoft hasta el momento: ejecución de pruebas automáticas sobre versiones específicas de dependencias y entornos de ejecución.

Dado que el código fuente de la plataforma se encuentra alojado en dos repositorios de GitLab, uno para el *frontend* React.js, y otro para el *backend* Java EE, deciden crear dos nuevos recursos asociados al *stack* de forma privada, uno para el *frontend* y otro para el *backend*. A su vez, se configura la ejecución de las pruebas automáticas en los servidores de integración continua de GitLab, seleccionando los tipos de proyecto *NPM Node 8.11.0* y *Maven 3.5.4 - jdk7* para el recurso de *frontend* y *backend*, respectivamente.

Información del recurso

Nombre del recurso
InspiraSoft e-commerce - frontend

Link del repositorio
<https://gitlab.com/nmartinezb/e-commerce-inspirasoft-frontend>

Tipo de repositorio
GitLab

Quiero trackear el repositorio

Al activar esta opción, el sistema trackeará las compatibilidades del proyecto con nuevas versiones de esta tecnología. Para esto, se requiere que ingrese un access token del repositorio de GitLab donde está alojado el proyecto.

Servicio de CI
GitLab

Si selecciona Gitlab como opción, se ejecutarán pipelines en su repositorio. Si selecciona Jenkins, los pipelines se ejecutarán en un servidor Jenkins interno del observatorio.

Branch
master

Tipo de proyecto
Proyecto NPM Node 8.11.0 - GITLAB

Figura 39: Recurso de aplicación *frontend* React.js de *e-commerce* de InspiraSoft

Información del recurso

Nombre del recurso
InspiraSoft e-commerce - backend

Link del repositorio
https://gitlab.com/nmartinezb/e-commerce-inspirasoft-backend

Tipo de repositorio
GitLab

Quiero trackear el repositorio

Al activar esta opción, el sistema trackeará las compatibilidades del proyecto con nuevas versiones de esta tecnología. Para esto, se requiere que ingrese un access token del repositorio de GitLab donde está alojado el proyecto.

Servicio de CI
GitLab

Si selecciona Gitlab como opción, se ejecutarán pipelines en su repositorio. Si selecciona Jenkins, los pipelines se ejecutarán en un servidor Jenkins interno del observatorio.

Branch
master

Tipo de proyecto
Proyecto Maven 3.5.4 - jdk7 - GITLAB

Figura 40: Recurso de aplicación *backend* Java EE de *e-commerce* de InspiraSoft

De esta forma, ante nuevas liberaciones de versiones de las dependencias del proyecto, InspiraSoft es capaz de realizar pruebas automáticas y recibir notificaciones de los resultados, logrando mantener las dependencias en sus versiones más recientes.

Una vez configurados los recursos, se disponen a ejecutar las pruebas del recurso asociado al *frontend* con la versión 16.6.3 de React.js y el tipo de proyecto *NPM Node 8.11.0* sobre los servicios de CI de GitLab, tal como se muestra en la Figura 41.

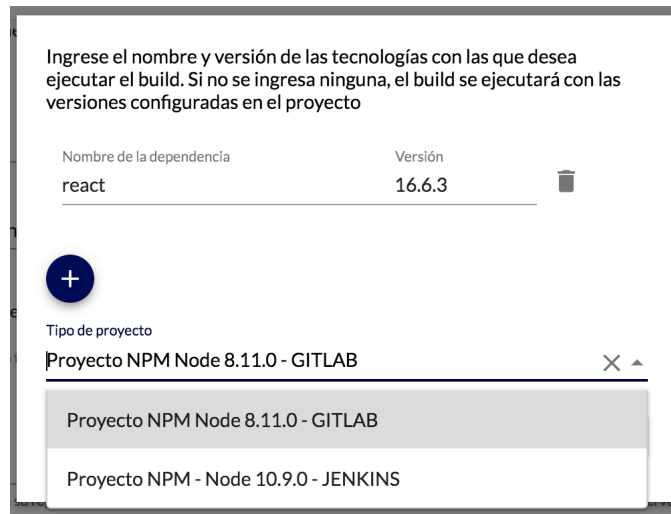


Figura 41: Ejecución de pruebas de recurso *frontend* utilizando la versión 16.6.3 de React.js

En la Figura 42 se muestra cómo se actualiza la versión de React.js en el repositorio una vez iniciada la ejecución de las pruebas a través del Observatorio.

Commit realizado desde el Observatorio Tecnológico

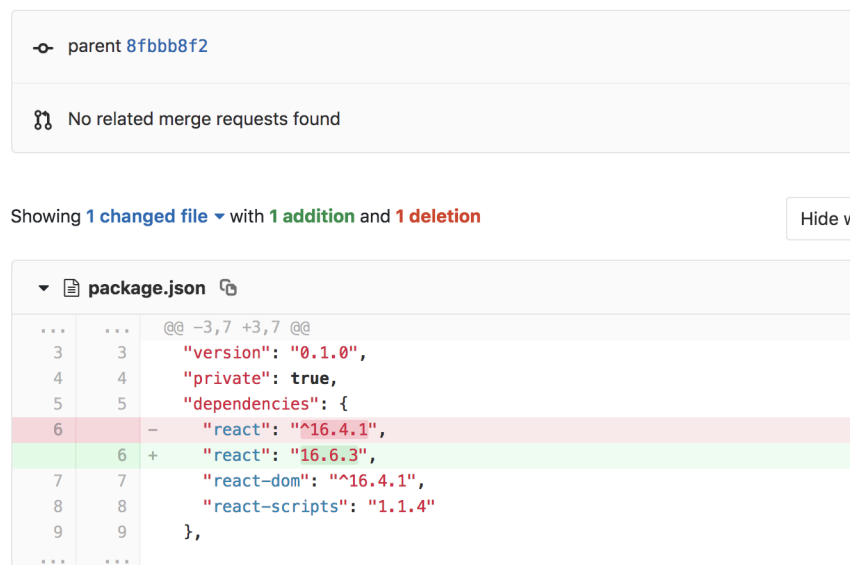


Figura 42: *Commit* del Observatorio para actualizar la versión de React

Finalmente, la Figura 43 muestra una captura del resultado de la ejecución de las pruebas en el repositorio GitLab. En primera instancia, se observa cómo se descarga la imagen de Docker correspondiente al entorno de ejecución Node 8.11.0, para luego ejecutar las pruebas y obtener un resultado fallido. De esta forma, los desarrolladores de InspiraSoft detectan el problema y proceden a trabajar sobre el proyecto para realizar los ajustes pertinentes en el código fuente.

```
Running with gitlab-runner 11.5.0 (3afdaba6)
on docker-auto-scale-ed2dce3a
Using Docker executor with image node:8.11.0 ...
Pulling docker image node:8.11.0 ...
Using docker image sha256:5baa6f270a5721b44bd2a38c24df7238d7e1df69229dda0ef939640ca
Running on runner-ed2dce3a-project-9845030-concurrent-0 via runner-ed2dce3a-srm-154

Cloning repository... Entorno de ejecución Node 8.11.0
Cloning into '/builds/nmartinezb/ecommerce-inspirasoft-frontend'...
Checking out 81c30c86 as ZQIPCJFZJ...
Skipping Git submodules setup
Checking cache for default...
Downloading cache.zip from https://storage.googleapis.com/gitlab-com-runners-cache/
Successfully extracted cache
$ npm install
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN ajv-keywords@3.2.0 requires a peer of ajv@^6.0.0 but none is installed. You
npm WARN react-dom@15.4.1 requires a peer of react@^15.4.1 but none is installed. You
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.4 (node_modules/fsevent
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2
{"os":"linux","arch":"x64"})

added 7 packages and updated 1 package in 10.653s
$ export PATH=$PATH:/usr/local/bin
$ npm run test

> pipeline-jenkins@0.1.0 test /builds/nmartinezb/ecommerce-inspirasoft-frontend
> react-scripts test --env=jsdom

Resultado fallido de pruebas

FAIL src/App.test.js
  ● Test suite failed to run

    Cannot find module 'react/lib/ReactComponentTreeHook' from 'ReactDebugTool.js'

      at Resolver.resolveModule (node_modules/jest-resolve/build/index.js:179:17)
      at Object.<anonymous> (node_modules/react-dom/lib/ReactDebugTool.js:16:30)

Test Suites: 1 failed, 1 total
Tests: 0 total
Snapshots: 0 total
Time: 1.045s
Ran all test suites
```

Figura 43: Resultado fallido de pruebas sobre recurso *frontend*

7.8. Nuevo entorno de ejecución

Al cabo de unos meses, el equipo de desarrolladores observa en el panel de noticias del Observatorio, la existencia de un nuevo tipo de proyecto basado en la versión de Java

1.8 como entorno de ejecución. De esta forma, deciden ejecutar las pruebas automáticas del recurso asociado al *backend*, seleccionado el nuevo tipo de proyecto *Maven 3.5.4 - jdk8* como se ilustra en la Figura 44. Al obtener resultados satisfactorios, se procede a actualizar el entorno de ejecución del ambiente de producción de la plataforma.

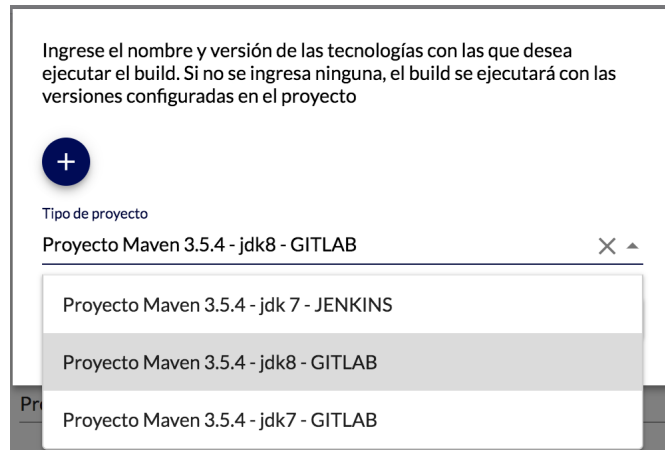


Figura 44: Selección de Java 1.8 para ejecución de pruebas

7.9. Conclusión del caso de estudio

El caso de estudio permitió presentar la aplicabilidad del Observatorio a la realidad de una empresa.

La realidad planteó un conjunto de problemas que usualmente suelen enfrentar las empresas que desarrollan software, desde la investigación de nuevas tendencias, elección de tecnologías, mantenimiento de proyectos que perduran en el tiempo y formación de nuevos empleados.

Se mostró que un sistema de estas características podría resultar de utilidad tanto para las empresas como para los entusiastas del mundo del desarrollo de aplicaciones web, dado que el Observatorio Tecnológico es capaz de acompañar el ciclo de vida de las diferentes aplicaciones web, tanto para las empresas como para la comunidad.

Por lo tanto, se puede concluir que el Observatorio Tecnológico soporta la realidad planteada para el caso de estudio.

8. Conclusiones y trabajo a futuro

En esta sección se presentan conclusiones del proyecto y el trabajo a futuro a realizarse como una extensión del mismo.

8.1. Resumen y conclusiones

En un contexto donde la tecnología está en constante evolución y actualización, resulta de utilidad disponer de una plataforma que cuente con la capacidad de brindar soporte a las distintas etapas del desarrollo de aplicaciones web. De esta forma, el objetivo general del proyecto consistió en el diseño y construcción de un Observatorio Tecnológico orientado al desarrollo de aplicaciones web que brinde soporte al ciclo de vida de los proyectos de desarrollo, acompañando a los usuarios en el proceso de selección de tecnologías, implementación, puesta en producción y mantenimiento.

En una primera instancia, se realizó un estudio de las arquitecturas de las aplicaciones web, enfocándose en sus componentes principales y los problemas que resuelven. En base a este análisis, se introdujo el concepto de arquitectura general de proceso de desarrollo de aplicaciones web (AGPDAW), destinado a modelar los componentes involucrados en el desarrollo de las mismas, abarcando las áreas tanto de diseño, implementación como soporte tecnológico para el desarrollo. Por otro lado, se realizó un relevamiento de soluciones de observatorios tecnológicos existentes, no logrando encontrar otras soluciones que resolvieran la problemática planteada. No obstante, mediante dicho relevamiento fue posible identificar las características principales que son deseables en los Observatorios, independientemente del área en el que se especialicen.

A partir de esto se realizó el análisis de los requerimientos, el cual permitió presentar una solución conformada por un modelo conceptual que da soporte a la problemática planteada. Se propusieron un conjunto de funcionalidades que permitieron satisfacer tanto las características que se determinaron relevantes para los observatorios tecnológicos, como los requerimientos funcionales y no funcionales relevados en la etapa de análisis. Por otro lado, se propuso una arquitectura lógica del Observatorio la cual representa una visión global del sistema, brindando detalle de los componentes involucrados en la implementación de las funcionalidades propuestas.

Posteriormente, a modo de realizar una prueba de factibilidad técnica, se implementó un prototipo del Observatorio Tecnológico orientado al desarrollo de aplicaciones web donde se incluyeron las funcionalidades más relevantes e innovadoras que no se encontraron en los observatorios relevados. Entre ellas, se encuentra la creación de tecnologías con la posibilidad de asignación de recursos (documentación, casos de prueba, etc.), un asistente de selección de tecnologías, integración con distintos servicios de integración continua y repositorios de código para el almacenamiento y mantenimiento de proyectos de usuarios.

Finalmente, se desarrolló un caso de estudio que permitió mostrar la aplicabilidad del Observatorio Tecnológico en un contexto real. El caso de estudio abarcó diferentes funcionalidades del Observatorio, planteando la realidad de una empresa ficticia y su necesidad de incorporarse al Observatorio ante un nuevo requerimiento presentado por un cliente. Durante el caso de estudio se presentaron las distintas problemáticas a las que se fue enfrentando, mostrando cómo el Observatorio logró proveer una solución adecuada.

Como resultado de este proceso, se obtienen las siguientes conclusiones:

- Si bien en el relevamiento realizado se encontraron soluciones de observatorios tecnológicos enfocados en las Tecnologías de la Información, no se encontraron observatorios tecnológicos orientados al desarrollo de aplicaciones web de forma explícita. En base a dicho relevamiento, se concluye que existe un conjunto de funcionalidades que deben cumplir los observatorios tecnológicos independientemente de su enfoque.
- La implementación de un Observatorio Tecnológico para un área específica requiere de la conceptualización de la información que debe manejar, lo que implica una amplia comprensión del dominio específico para que la solución contemple todos los conceptos requeridos.
- Las principales funcionalidades de la solución propuesta que fueron incluidas en la implementación del prototipo son viables técnicamente. Por ejemplo, integraciones con servicios de repositorios de código, integración continua, canales de noticias, junto con la extensibilidad de la herramienta en lo que refiere a la incorporación de nuevo servicios de este tipo.
- A partir del desarrollo del caso de estudio, se concluye que la solución propuesta es aplicable a la realidad de una empresa como la presentada en el mismo.

Como conclusión final, en base al relevamiento y al estudio realizado, se entiende que el Observatorio Tecnológico constituye una herramienta que puede ser de utilidad para el desarrollo de aplicaciones web, en la medida que varias organizaciones y usuarios adopten su uso. Sin embargo, la puesta en práctica del mismo conlleva un costo de tiempo de desarrollo, mantenimiento y administración considerable. Para que el Observatorio sea de utilidad, debe contar con un amplio volumen de información, por lo que es necesario encontrar mecanismos automatizados para la carga y validación de la misma. No obstante, de contar con los recursos necesarios, sería de utilidad contar con una herramienta de estas características.

8.2. Trabajo a futuro

En esta sección se describen las posibles extensiones y mejoras al trabajo realizado.

8.2.1. Funcionalidades del prototipo

A continuación se listan las funcionalidades propuestas no implementadas en el prototipo que resultan prioritarias para la inclusión en un trabajo a futuro.

- **Gestión de organizaciones, usuarios, roles y permisos**

Si bien la solución propuesta soporta el manejo de usuarios, roles y permisos, el prototipo implementado no cuenta con dicha funcionalidad, la que resulta imprescindible para la aplicación del Observatorio en un contexto real donde coexistan diferentes usuarios y organizaciones. Se propone como trabajo a futuro la implementación de mecanismos de gestión de usuarios, roles y permisos.

- **Espacios de trabajo**

Contar con distintos espacios de trabajo le permite a los usuarios la posibilidad de elegir entre trabajar sobre proyectos personales y trabajar sobre proyectos de las organizaciones a las que pertenece. De esta forma, se propone la implementación de mecanismos de gestión de espacios de trabajo para los usuarios.

- **Sistema de puntos**

Es imprescindible que la información recolectada por el Observatorio mantenga estándares de calidad mínimos. De esta forma, se propone la implementación del sistema de puntos propuesto en la solución, a modo de mantener un estándar de calidad mínimo.

- **Notificaciones y alertas a los usuarios**

Se plantea para trabajo a futuro la implementación de envío de notificaciones a los usuarios en tiempo real a modo de alertar acerca de la liberación de nuevas tecnologías, resultados de ejecución de pruebas automáticas, entre otros. Para implementar este punto, se propone el envío de *e-mails* y *web push notifications* [67] a la capa de presentación.

8.2.2. Nuevas integraciones con plataformas y servicios

A continuación se propone un conjunto de integraciones y servicios a incorporar al Observatorio.

- **Servicios repositorios de código**

La arquitectura del Observatorio fue diseñada de forma extensible con el fin de permitir la integración con nuevos sistemas de repositorios de código. El prototipo

cuenta con una interfaz con métodos genéricos para la creación de *commits* y ramas, la cual puede contar con distintas implementaciones mediante el consumo de las APIs de los diferentes servicios basados en Git. De esta forma, se propone la implementación de dicha interfaz consumiendo la API de GitHub² a modo de proveer soporte para el almacenamiento de los proyectos de los usuarios en dicho servicio.

- **Herramientas de integración continua**

Las integraciones con servicios de integración continua incluidas dentro del alcance del prototipo se consideran que fueron satisfactorias ya que cumplen con los requerimientos. Aún así, resulta de interés para un trabajo a futuro la investigación de nuevos servicios que incluyan nuevas funcionalidades que puedan enriquecer al observatorio.

- **Instancia privada de servidor GitLab**

Dadas las limitaciones de recursos que establece GitLab en su servidor público, se propone como trabajo a futuro la inclusión de un servidor privado de GitLab propio del Observatorio, de forma análoga al servidor de GitLab de la Facultad de Ingeniería.

8.2.3. Mejoras y complementos

A continuación se proponen mejoras y complementos a incorporar al Observatorio.

- **Detección automática de liberaciones de versiones de tecnologías**

El prototipo implementado permite el registro de nuevas versiones de tecnologías únicamente de forma manual. Como trabajo a futuro se propone la investigación de mecanismos genéricos de detección de nuevas liberaciones de tecnologías. Dentro de las soluciones candidatas, se vio que los repositorios públicos de GitHub de las tecnologías, servicio donde se aloja la gran mayoría de los proyectos de código abierto, cuentan con un canal RSS de liberaciones, los cuales podrían ser consumidos por el Observatorio de forma periódica para detectar nuevas versiones.

Para el caso de las tecnologías que no se encuentran alojadas en repositorios GitHub, tales como Java EE y .NET, se propone la implementación de *web scrapping*³ sobre los *blogs* oficiales de dichas plataformas.

- **Soporte de nuevos tipos de proyecto**

Para la actualización de versiones de tecnologías en los archivos de dependencia de los proyectos, se propone como trabajo a futuro un mecanismo genérico y extensible por los administradores del Observatorio que permita soportar nuevos tipos

²<https://developer.github.com/v3/>

³Técnica computarizada de extracción de información de sitios web

de proyecto bajo demanda. Para esto, se propone la implementación de *plugins*⁴ encargados de la actualización de versiones en archivos de dependencias. Estos *plugins* podrían ser incorporados al Observatorio mediante el panel de administración, adjuntándolo a un tipo de proyecto. De esta forma, al momento de actualizar la versión de un recurso asociado a un determinado tipo de proyecto, el Observatorio delegará la responsabilidad al *plugin* asociado.

- **Nueva interfaz de usuario**

La capa de presentación del prototipo implementado incluye tanto la interfaz de administración del Observatorio, como la interfaz destinada a los usuarios finales y organizaciones. Se propone como trabajo a futuro la implementación de una nueva capa de presentación orientada a la experiencia de los usuarios y organizaciones, que en conjunto con un equipo de diseño implemente las interfaces y casos de uso principales. De esta forma, el prototipo implementado permanecería únicamente como panel de administración del Observatorio.

- **Manuales de usuario**

En base a la cantidad de funcionalidades propuestas para el Observatorio, se propone como trabajo a futuro la generación de manuales de usuario y páginas de ayuda que brinden soporte acerca de las distintas funcionalidades del Observatorio.

- **Pruebas con usuarios**

Resulta de interés evaluar la aplicabilidad del Observatorio Tecnológico en un contexto real. Se propone realizar pruebas con diferentes usuarios y organizaciones diseñando un plan a seguir durante un tiempo acotado para luego evaluar los resultados obtenidos. De esta forma, se obtendría retroalimentación de los usuarios finales, lo que podría presentar nuevos requerimientos.

- **Carga de datos**

El principal valor del Observatorio reside en la información recolectada. En la medida que los usuarios y organizaciones comiencen a utilizar la plataforma, el mismo se nutrirá de aportes y experiencias, agregando aún más valor al mismo. De esta forma, se requiere contar con una carga inicial de información que facilite la incorporación de usuarios y organizaciones a la plataforma. Se propone como trabajo a futuro realizar un relevamiento inicial de tecnologías en el mercado del desarrollo de aplicaciones web para ingresar en la base de datos del Observatorio.

⁴Complemento de software que añade una funcionalidad adicional.

8.2.4. Aspectos técnicos

A continuación se listan mejoras técnicas a incorporar al Observatorio.

1. Almacenamiento de los datos

El Observatorio Tecnológico cuenta con una base de datos relacional que almacena toda la información recolectada por el mismo. No obstante, resulta de interés considerar otros tipos de base de datos que se adecúen a los tipos de datos almacenados. A modo de ejemplo, en el prototipo se trabaja con archivos XSLT para el procesamiento y transformación en memoria de los XML consumidos de los canales RSS. Una mejora a futuro consiste en utilizar una base de datos no relacional, tal como es ExistDB⁵, para el almacenamiento y procesamiento de los XML mediante la utilización de XQuery⁶. De esta manera, se libera el procesamiento en memoria de la noticias en el observatorio, delegando la transformación al motor de base de datos.

También se propone la utilización de una base de datos *Lightweight Directory Access Protocol* (LDAP)⁷ para el almacenamiento y gestión de los usuarios del Observatorio.

2. Almacenamiento en caché

Un caso de uso real del Observatorio, operando en un ambiente de producción con un gran número de usuarios concurrentes, requiere de tiempos de respuesta adecuados a modo de no comprometer la experiencia de uso. De esta forma, se propone como trabajo a futuro el almacenamiento en *caché* de los datos más consumidos por los usuarios, como pueden ser las diferentes tecnologías y *stacks*. Una posibilidad consiste en la inclusión de *clusters*⁸ de bases de datos en memoria que permitan un rápido acceso a la información. Ejemplos de este tipo de base de datos que se podrían utilizar son Infinispan⁹ y Redis¹⁰.

3. Seguridad

Dado que en un contexto real el Observatorio contará con información confidencial de diferentes organizaciones y usuarios, es imprescindible implementar mecanismos de seguridad que garanticen la disponibilidad, integridad y confidencialidad de la información. Se propone a futuro la implementación de mecanismos que contemplen los aspectos planteados.

⁵<http://exist-db.org>

⁶https://www.w3schools.com/xml/xquery_intro.asp

⁷LDAP (RFC4511) - <https://tools.ietf.org/html/rfc4511>

⁸Grupo de servidores independientes con mismo proposito.

⁹<http://infinispan.org/>

¹⁰<https://redis.io/>

Referencias

- [1] Microsoft. Microsoft application architecture guide - patterns & practices 2nd edition. chapter 3. [https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658117\(v%3dpandp.10\)](https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658117(v%3dpandp.10)), 2009. [En línea; accedido en Noviembre 2018].
- [2] MaxCDN. What is a web applications. <https://www.maxcdn.com/one/visual-glossary/web-application/>, 2016. [En línea; accedido en Noviembre 2018].
- [3] mozilla.org. Http. <https://developer.mozilla.org/es/docs/Web/HTTP/Overview>, 2018. [En línea; accedido en Diciembre 2018].
- [4] What is web application architecture? how it works, trends, best practices and more. <https://stackify.com/web-application-architecture/>, sep 2017. [En línea; accedido en Noviembre 2018].
- [5] Microsoft. Microsoft application architecture guide - patterns & practices 2nd edition. chapter 21. <https://msdn.microsoft.com/en-us/library/ee658099.aspx>, 2009. [En línea; accedido en Noviembre 2018].
- [6] Microsoft. Microsoft application architecture guide - patterns & practices 2nd edition. chapter 21 - general design considerations. <https://msdn.microsoft.com/en-us/library/ee658099.aspx#GeneralDesignConsiderations>, 2009. [En línea; accedido en Noviembre 2018].
- [7] AWS. Caché. <https://aws.amazon.com/es/caching/>, 2018. [En línea; accedido en Diciembre 2018].
- [8] Udacity. 3 web dev careers decoded: Front-end vs back-end vs full stack. <https://blog.udacity.com/2014/12/front-end-vs-back-end-vs-full-stack-web-developers.html/>, 2018. [En línea; accedido en Diciembre 2018].
- [9] Javascript. <https://developer.mozilla.org/es/docs/Web/javascript>, 2018. [En línea; accedido en Julio 2018].
- [10] Ajax. <https://developer.mozilla.org/es/docs/Web/Guide/AJAX>, 2018. [En línea; accedido en Diciembre 2018].
- [11] Html. <https://developer.mozilla.org/es/docs/Web/HTML>, 2018. [En línea; accedido en Julio 2018].
- [12] Rfc1866 (html). <https://tools.ietf.org/html/rfc1866>, 1995. [En línea; accedido en Julio 2018].
- [13] Introducción a la Ingeniería de Software Facultad de Ingeniería UdeLaR. Arquitectura de software. <https://www.fing.edu.uy/tecnoinf/mvd/cursos/ingsoft/material/teorico/is05-ArquitecturaDeSoftware.pdf>.

- [14] Google. Search engine optimization (seo). <https://support.google.com/webmasters/answer/7451184?hl=en>, 2018. [En línea; accedido en Diciembre 2018].
- [15] Spa vs mpa. <https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58>, 2018. [En línea; accedido en Diciembre 2018].
- [16] Ajax approach. <http://adaptivepath.org/ideas/ajax-new-approach-web-applications/>, 2018. [En línea; accedido en Diciembre 2018].
- [17] Lazy loading. <https://blog.stackpath.com/glossary/lazy-loading/>, 2017. [En línea; accedido en Diciembre 2018].
- [18] UpWork. Back-end technology: The role of the back-end web developer. <https://www.upwork.com/hiring/development/back-end-web-developer/>, 2018. [En línea; accedido en agosto 2018].
- [19] RedHat. What are apis? <https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces>, 2018. [En línea; accedido en agosto 2018].
- [20] Python.org. Webframeworks - python wiki. <https://wiki.python.org/moin/WebFrameworks>, 2018. [En línea; accedido en Noviembre 2018].
- [21] Git. <https://git-scm.com>. [En línea; accedido en Diciembre 2018].
- [22] Gitlab. <https://docs.gitlab.com>.
- [23] AWS. ¿qué es la integración continua? <https://aws.amazon.com/es/devops/continuous-integration/>, 2018. [En línea; accedido en Diciembre 2018].
- [24] GitLab. Gitlab ci - pipelines. <https://docs.gitlab.com/ee/ci/pipelines.html>. [En línea; accedido en Diciembre 2018].
- [25] Ubuntu. Package dependencies. https://help.ubuntu.com/community/InstallingSoftware#Package_Dependencies, 2014. [En línea; accedido en Diciembre 2018].
- [26] Oracle. Understanding maven version numbers. https://docs.oracle.com/middleware/1212/core/MAVEN/maven_version.htm#MAVEN401, 2015. [En línea; accedido en Diciembre 2018].
- [27] Semantic versioning. <https://semver.org/>. [En línea; accedido en Diciembre 2018].
- [28] Maven. <https://maven.apache.org/>, 2018. [En línea; accedido en Diciembre 2018].
- [29] NPM. npm help docs. <https://docs.npmjs.com>.

- [30] Docker. <https://docs.docker.com/>, 2018. [En línea; accedido en Noviembre 2018].
- [31] Docker container. <https://www.docker.com/resources/what-container>, 2018. [En línea; accedido en Noviembre 2018].
- [32] yaml.org. Yaml. <http://yaml.org/>.
- [33] Docker compose. <https://docs.docker.com/compose/overview/>, 2018. [En línea; accedido en Noviembre 2018].
- [34] W3.org. Extensible markup language (xml). <https://www.w3.org/TR/xml/>, 2008. [En línea; accedido en Noviembre 2018].
- [35] IETF.org. Rfc4825 - xml. <https://tools.ietf.org/html/rfc4825>, 2007. [En línea; accedido en Noviembre 2018].
- [36] W3.org. W3c xml schema definition language (xsd). <https://www.w3.org/TR/xmlschema11-1/>, 2012. [En línea; accedido en Noviembre 2018].
- [37] W3.org. Xsl transformations (xslt). <https://www.w3.org/TR/xslt-30/>, 2017. [En línea; accedido en Noviembre 2018].
- [38] W3 Schools. Xml rss. https://www.w3schools.com/xml/xml_rss.asp.
- [39] Microsoft. Microsoft application architecture guide - patterns & practices 2nd edition. chapter 10. <https://msdn.microsoft.com/en-us/library/ee658121.aspx>, 2009. [En línea; accedido en Noviembre 2018].
- [40] Microsoft. Microsoft application architecture guide - patterns & practices 2nd edition. chapter 9 - rest and soap. [https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658090\(v=pandp.10\)#validation](https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658090(v=pandp.10)#validation), 2009. [En línea; accedido en Noviembre 2018].
- [41] Oracle. What is auditing? https://docs.oracle.com/cd/E36784_01/html/E37127/auditov-2.html#scrolltoc, 2014. [En línea; accedido en Diciembre 2018].
- [42] Grupo de Seguridad Informática Facultad de Ingeniería UdelaR. Identificación, autenticación, autorización. https://eva.fing.edu.uy/pluginfile.php/58016/mod_resource/content/6/FSI-2018-IAA.pdf, 2018. [En línea; accedido en Diciembre 2018].
- [43] ITI. Iti technology - what is an observatory. <https://observatorio.iti.upv.es/about/>, 2018. [En línea; accedido en Noviembre 2018].
- [44] Rodrigo Díaz Ayala. Panorama y retos de los observatorios tecnológicos en México, 01 2017.

- [45] ITI. Iti technology observatory - portada. <https://www.iti.es/servicios/observatorio-tecnologico/>, 2018. [En línea; accedido en Noviembre 2018].
- [46] Observatoire des sciences et des technologies. <https://www.ost.uqam.ca/>, 2017. [En línea; accedido en Diciembre 2018].
- [47] European it observatory. <https://www.eito.com>, 2017. [En línea; accedido en Diciembre 2018].
- [48] Oracle. Introduction to java platform, enterprise edition 7. <http://www.oracle.com/technetwork/java/javaee/javaee7-whitepaper-1956203.pdf>, 2013. [En línea; accedido en Noviembre 2018].
- [49] Oracle. Javasever faces technology. <http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html>.
- [50] Oracle. Introducción a javasever faces. <http://www.jtech.ua.es/j2ee/publico/jsf-2012-13/sesion01-apuntes.html>, 2018. [En línea; accedido en Noviembre 2014].
- [51] Adam Bien. Java ee 7 and jax-rs 2.0. <https://www.oracle.com/technetwork/articles/java/jaxrs20-1929352.html>, 2013. [En línea; accedido en Noviembre 2018].
- [52] Oracle. Enterprise java beans (ejbs). https://docs.oracle.com/cd/E24329_01/web.1211/e24446/ejbs.htm#INTRO255.
- [53] Oracle. Java persistence api (jpa). <https://docs.oracle.com/javaee/7/tutorial/persistence-intro.html>.
- [54] Oracle. Working with realms, users, groups, and roles. <https://docs.oracle.com/javaee/7/tutorial/security-intro005.htm>.
- [55] Wildfly Org. Wildfly. <http://wildfly.org/>.
- [56] Eclipse. Eclipse ide. <https://www.eclipse.org>.
- [57] MySQL. Mysql. <https://www.mysql.com/>.
- [58] GitLab. Gitlab api. <https://docs.gitlab.com/ee/api/>.
- [59] Jenkins. Jenkins. <https://jenkins.io/>.
- [60] JBoss. Imagen oficial de wildfly. <https://hub.docker.com/r/jboss/wildfly/>, 2018. [En línea; accedido en Noviembre 2018].
- [61] Jenkins. Imagen oficial de jenkins. <https://hub.docker.com/r/jenkins/jenkins/>, 2018. [En línea; accedido en Noviembre 2018].

- [62] PostgreSQL. Imagen oficial de postgresql. https://hub.docker.com/_/postgres/, 2018. [En línea; accedido en Noviembre 2018].
- [63] Facebook. Create react app. <https://reactjs.org/docs/create-a-new-react-app.html>.
- [64] AWS. Amazon web services. <https://aws.amazon.com>.
- [65] AWS. Amazon ec2. <https://aws.amazon.com/es/ec2/>.
- [66] Harvard. Rss 2.0. <https://cyber.harvard.edu/rss/rss.html>, 2003. [En línea; accedido en Diciembre 2018].
- [67] Technical Writer Joseph Medley. Web push notifications: Timely, relevant, and precise. <https://developers.google.com/web/fundamentals/push-notifications/>, 2018. [En línea; accedido en Diciembre 2018].
- [68] Django project. <https://www.djangoproject.com/>, 2018. [En línea; accedido en agosto 2018].
- [69] React.js. <https://reactjs.org/>, 2018. [En línea; accedido en agosto 2018].
- [70] Django templates. <https://docs.djangoproject.com/en/1.11/ref/templates/>, 2018. [En línea; accedido en agosto 2018].
- [71] Django rest framework. <http://www.django-rest-framework.org/>, 2018. [En línea; accedido en agosto 2018].
- [72] Django url dispatcher. <https://docs.djangoproject.com/en/2.1/topics/http/urls>, 2018. [En línea; accedido en agosto 2018].
- [73] Django views. <https://docs.djangoproject.com/en/2.1/topics/http/views/>, 2018. [En línea; accedido en agosto 2018].
- [74] Django orm. <https://docs.djangoproject.com/en/2.1/topics/db/models/>, 2018. [En línea; accedido en agosto 2018].
- [75] Mongo engine. <https://docs.djangoproject.com/en/2.1/topics/auth/default/#topic-authorization>, 2018. [En línea; accedido en agosto 2018].
- [76] Django authentication. <https://docs.djangoproject.com/en/2.1/topics/auth/>, 2018. [En línea; accedido en agosto 2018].
- [77] Django admin. <https://docs.djangoproject.com/en/2.1/ref/contrib/admin/>, 2018. [En línea; accedido en agosto 2018].
- [78] Unicorn. <http://gunicorn.org/>, 2018. [En línea; accedido en agosto 2018].

- [79] Pycharm. <https://www.jetbrains.com/pycharm/>, 2018. [En línea; accedido en Noviembre 2018].
- [80] Postgresql en django. <https://docs.djangoproject.com/en/2.1/ref/contrib/postgres/>, 2018. [En línea; accedido en Noviembre 2018].
- [81] Microsoft. C guide. <https://docs.microsoft.com/en-us/dotnet/csharp/>, 2018. [En línea; accedido en Noviembre 2018].
- [82] Tom FitzMacken. Asp.net web pages. <https://docs.microsoft.com/en-us/aspnet/web-pages/overview/getting-started/introducing-aspnet-web-pages-2/getting-started>, 2015. [En línea; accedido en Noviembre 2018].
- [83] Web Camps Team. Build restful apis with asp.net. <https://docs.microsoft.com/en-us/aspnet/web-api/overview/older-versions/build-restful-apis-with-aspnet-web-api>, 2013. [En línea; accedido en Noviembre 2018].
- [84] Mike Wasson. Routing in asp.net web api. <https://docs.microsoft.com/en-us/aspnet/web-api/overview/web-api-routing-and-actions/routing-in-aspnet-web-api>, 2018. [En línea; accedido en Noviembre 2018].
- [85] Gleb Akimov Diatom Enterprises. Asp.net mvc: Business logic as a separate layer. <http://www.diatomenterprises.com/asp-net-mvc-business-logic-as-a-separate-layer/>.
- [86] Diego Vega. Entity framework (ef) documentation. <https://msdn.microsoft.com/en-us/data/ee712907>, 2016. [En línea; accedido en Noviembre 2018].
- [87] Microsoft. Asp.net identity. <https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity>, 2016. [En línea; accedido en Noviembre 2018].
- [88] Microsoft. Microsoft internet information services. <https://www.iis.net/>.
- [89] Microsoft. Visual studio. <https://visualstudio.microsoft.com/es/vs/>.
- [90] Microsoft. Sql server. <https://www.microsoft.com/es-es/sql-server/>.
- [91] Jenkins CI. Jenkins pipeline. <https://jenkins.io/doc/book/pipeline/>.

A. Instanciación

A.1. Ecosistema Python

Python es un lenguaje de programación interpretado multiparadigma que provee soporte tanto a orientación a objetos, como programación imperativa y funcional. Dentro del área del desarrollo web en Python, se encuentra el *framework* Django [68]. Django es un *framework* orientado al desarrollo de aplicaciones web de código abierto que implementa el *stack web* por completo. A continuación se presentan un *web project template* junto con una tecnología que implementa cada componente.

A.1.1. Web Project Template y Stack

En la Figura 45 se presenta un ejemplo de *web project template*, el cual consta de los siguientes componentes:

- Capa de Presentación - MPA (Multiple Page Application)
- Capa de Presentación - SPA (Multiple Page Application)
- Capa de Servicios
- Enrutamiento
- Capa de Negocio
- Capa de Acceso a Datos
- Manejo de usuarios, autenticación y control de acceso
- Herramientas de administración
- Entorno de ejecución
- Herramientas de desarrollo
- Persistencia

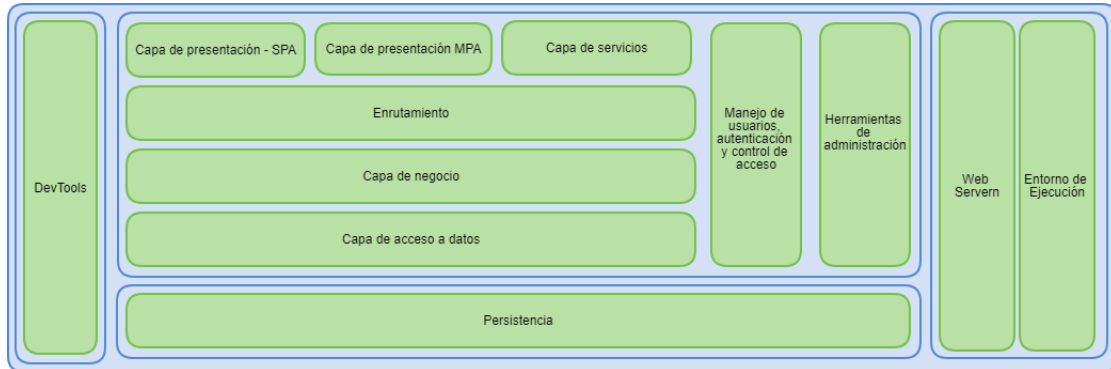


Figura 45: Web Project Template 2

Cada componente tiene asociado un conjunto de tecnologías, las cuales pueden pertenecer a más de un componente y a su vez, cada componente puede estar relacionado con tecnologías de diversos ecosistemas.

En la Figura 46 se plantea un ejemplo de instanciación de un *stack web* utilizando como base el *web project template* presentado anteriormente. Para ello, se considera el ecosistema Python como selección de tecnologías.

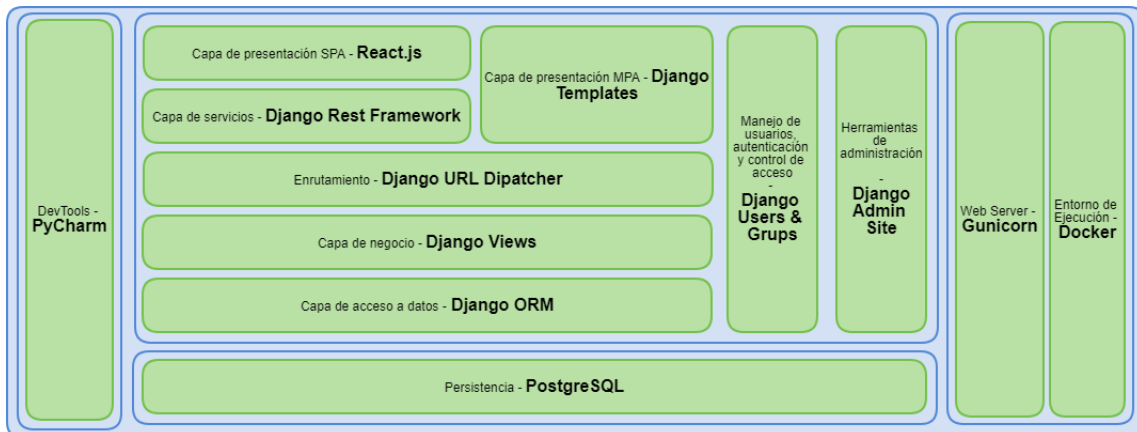


Figura 46: Stack para el ecosistema Python

A.1.2. Tecnologías

En esta sección se detallan las tecnologías seleccionadas para cada componente del *stack* asociado al *web project template* presentado en la sección A.1.

Capa de presentación - Single Page Application

La capa de presentación puede consistir en una SPA construida utilizando la herramienta React.js que consuma una capa de servicios. [69].

Capa de presentación - Multiple Page Application

Django Templates proporciona una metodología para definir la capa de presentación de la aplicación web, creando una clara separación entre la misma y la lógica de presentación. Para esto, introduce un lenguaje de templates o plantillas basadas en HTML, permitiendo la mantenibilidad de las mismas a usuarios no expertos con conocimientos básicos de HTML [70].

Capa de servicios

Django Rest Framework es una librería para Django que ofrece un conjunto de herramientas para implementar una API REST en una aplicación Django. Provee mecanismos para transformar objetos Python a formato JSON para ser transmitidos mediante HTTP a los clientes, así como también interfaces para alta, listado y modificación de las entidades mediante la API [71].

Enrutamiento

La tarea de enrutamiento de requests a métodos encargados de procesar las peticiones es manejada por el *URL Dispatcher* de Django. El programador debe definir las URLs genéricas mediante expresiones regulares, indicando la función encargada de procesar la petición, quien podría acceder a la capa de negocios y, mediante una plantilla, generar una página HTML de respuesta [72].

Capa de negocio

El acceso a la capa de negocio es implementado por las llamadas *Django Views*, quienes sirven de interfaz entre el cliente y la lógica de negocio. Estas vistas se forman, en su expresión más básica, de funciones que reciben una petición proveniente del cliente (mapeado por el URL Dispatcher), implementan toda la lógica necesaria para satisfacer ese pedido, y devuelve una respuesta. Durante todo este proceso es posible que requieran realizar consultas a la base de datos, consultas a sistemas externos y demás. En un caso ideal, esta lógica debería ser implementada por clases Python que expongan servicios a las Django Views a modo de mantener la lógica reutilizable, aunque en determinados casos de aplicaciones sencillas se podría incluir lógica en las Django Views [73].

Capa de acceso a datos

Para el acceso a base de datos, Django provee un ORM propio. Mediante esta herramienta, se definen clases Python con relaciones entre ellas (*One to One*, *One to Many*, *Many to Many*), que se mapean de forma transparente a un esquema de base de datos adecuado. Estas clases son definidas por Django como modelos [74].

Manejo de usuarios, autenticación y control de acceso

Django provee a los desarrolladores con un completo sistema de autenticación y autorización de usuarios [75], capaz de manejar cuentas de usuarios, grupos, permisos y sesiones. Por defecto, un usuario de Django mantiene los campos básicos característicos de un usuario: nombre, apellido, email, nombre de usuario y contraseña; dándole la posibilidad al programador de extender esta clase a una que se adecúe a las necesidades de la realidad. En términos de control de acceso, Django provee mecanismos de control de acceso basado en roles tanto sobre entidades como sobre instancias de las mismas. También posee mecanismos de autenticación de solicitudes web, de modo que solamente un pedido correctamente autenticado a nombre de un usuario existente sea atendido [76].

Herramientas de Administración

Una de las funcionalidades que provee Django casi sin requerir esfuerzo alguno del desarrollador, es el llamado *Django Admin Site*. Esta interfaz web, protegida bajo usuario y contraseña de administrador, permite, en su configuración más básica, tanto realizar consultas simples como insertar datos en la base de datos de forma amigable a través de una interfaz web. Para habilitar esta funcionalidad, se deben describir las clases del ORM de Django mediante configuraciones que definen por ejemplo, los modelos y campos se desean administrar [77].

Servidor Web

Un *Web Server Gateway Interface* (WSGI) es una interfaz a través de la cual los servidores web y las aplicaciones se comunican. Gunicorn es un WSGI para aplicaciones Python destinado a ambientes de producción [78].

Runtime

Mediante la utilización de Docker, es posible crear un contenedor virtualizado que ejecute un servidor WSGI (por ejemplo, Gunicorn) y lo exponga mediante HTTP.

Devtools - IDEs

Pycharm es un IDE para Python que provee soporte para el framework Django [79].

Persistencia

Django es compatible con el motor de base de datos PostgreSQL [80].

A.2. Ecosistema .NET

.NET es un *framework* de desarrollo destinado a la construcción de aplicaciones web y servicios. Provee todas las herramientas necesarias para la construcción de MPA, aunque también provee soporte para la implementación de servicios REST a ser consumidos por una SPA. El lenguaje de programación más utilizado para desarrollar aplicaciones .NET es C# [81].

A.2.1. Web Project Template

Para este ejemplo se utiliza el mismo *web project template* que fue definido en el ejemplo de la sección 5.1, el cual consta de los siguientes componentes:

- Capa de Presentación
- Capa de Servicios
- Enrutamiento
- Capa de Negocio
- Capa de Acceso a Datos
- Manejo de usuarios, autenticación y control de acceso
- Entorno de ejecución
- Herramientas de desarrollo
- Persistencia

Al igual que en los ejemplos anteriores, cada componente tiene asociado un conjunto de tecnologías. Para este ejemplo se toma de base el ecosistema .NET con el fin de seleccionar tecnologías. En la Figura 47 se muestra una posible selección.

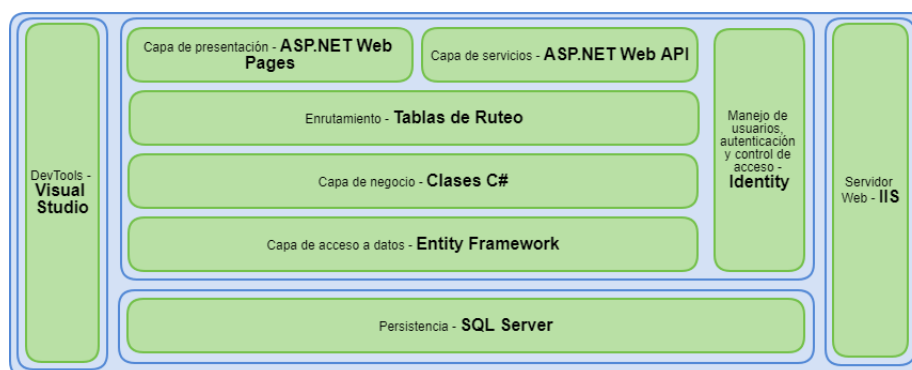


Figura 47: Stack para el ecosistema .NET

A.2.2. Tecnologías

A continuación se detalla, para cada componente del *stack*, la tecnología asociada.

Capa de presentación

ASP.NET Web Pages es una herramienta del framework .NET destinada a la creación de páginas web dinámicas. Para esto, se embebe código dentro de una plantilla HTML, el cual será ejecutado por el servidor, dando como resultado una página HTML. Este código embebido podría consumir los servicios de una capa de negocio para generar, por ejemplo, listados de datos extraídos de una base de datos [82].

Capa de servicios

ASP.NET Web API es la herramienta provista por .NET para la construcción de capas de servicio del tipo REST. Para esto, se dispone de una clase *ApiController*, la cual puede ser extendida para implementar el controlador que atienda las peticiones a un recurso [83].

Enrutamiento

Las *Routing Tables* son las encargadas de mapear URLs y métodos HTTP a controladores de la capa de servicios encargados de procesar las peticiones entrantes [84].

Capa de negocio

La capa de negocio puede ser implementada en los *ApiController* de la Capa de Servicios, aunque esto no es conveniente ya que impide la reutilización de código y no se estaría respetando la separación de capas. De esta forma, resulta conveniente implementar la capa de negocio en clases C# específicas, las cuales expongan una interfaz de acceso a sus métodos para ser consumidos tanto por los controladores de la capa de servicios, como por otros módulos de integración con servicios externos, entre otros [85].

Capa de acceso a datos

Entity Framework es un ORM perteneciente al ecosistema .NET que permite a los desarrolladores trabajar con bases de datos relacionales de forma transparente. El programador simplemente debe describir las entidades del dominio como clases con atributos relaciones entre ellos, y *Entity Framework* se encarga de crear y mantener las tablas que sean necesarias en la base de datos. Al proveer un lenguaje de consulta común de alto nivel, elimina la necesidad de lidiar con lenguajes consultas de bases de datos particulares [86].

Manejo de usuarios, autenticación y control de acceso

ASP.NET Identity es una herramienta provista por .NET que provee mecanismos de autenticación, autorización y control de acceso sobre los recursos de una aplicación .NET. Hace uso de *Entity Framework* para el almacenamiento y gestión de usuarios en las bases de datos [87].

Servidor Web

Internet Information Services (IIS) es un servidor de aplicaciones web .NET [88].

Devtools - IDEs

Visual Studio es un entorno de desarrollo integrado (IDE) para desarrollar aplicaciones .NET [89].

Persistencia

SQL Server es un sistema gestor de bases de datos creado por Microsoft compatible con aplicaciones .NET [90].

B. Integración con canales RSS

La Figura 48 presenta el esquema XSD definido por el Observatorio para la transformación de los datos obtenidos de los canales RSS.

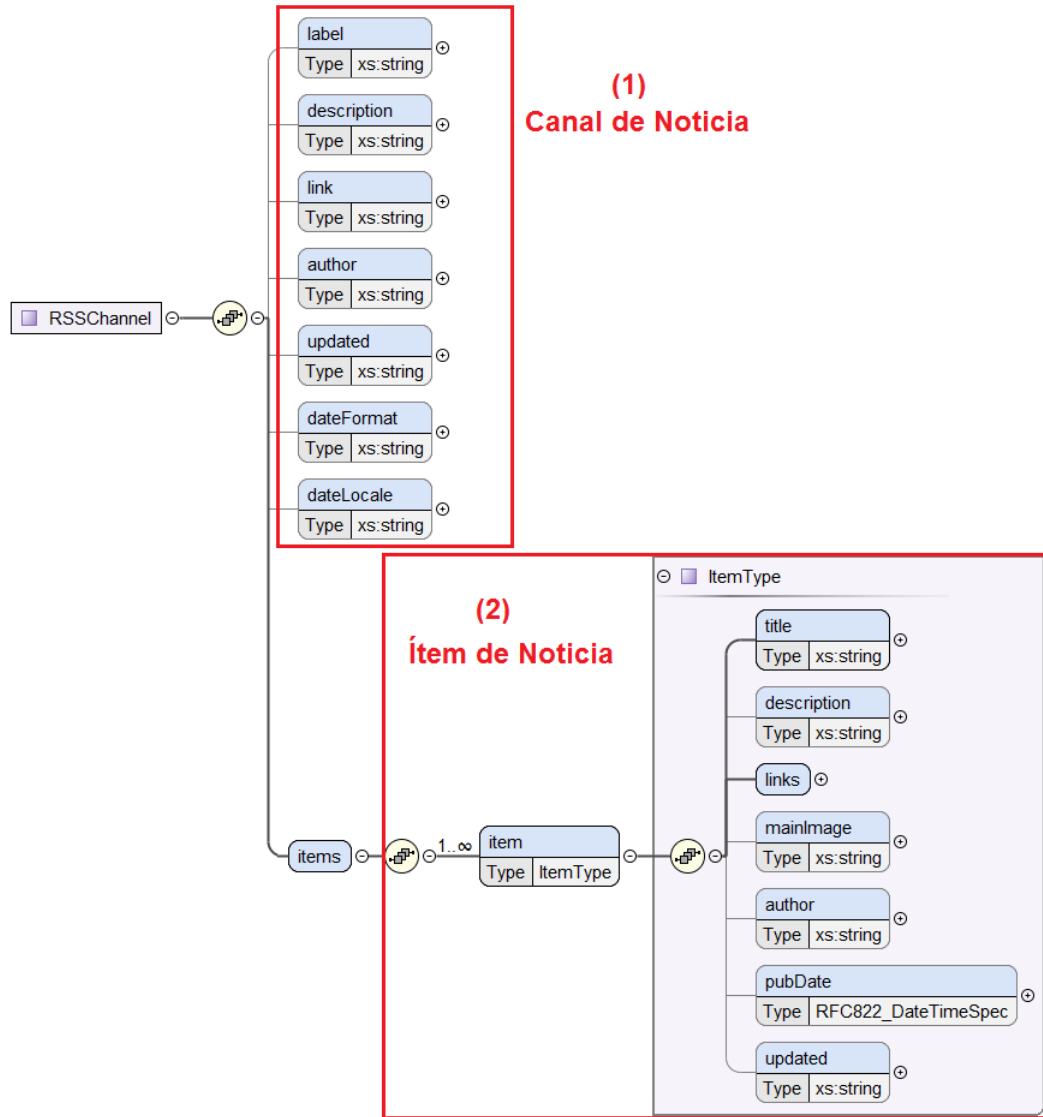


Figura 48: Esquema XSD definido por el observatorio.

El código XSLT correspondiente a un canal debe especificar cómo extraer información general del mismo (título, descripción, enlaces, autor, fecha de actualización, formato de

fecha utilizado) (1). Luego debe establecer, para cada noticia, cómo extraer el título, descripción, enlaces, etc. (2).

En la Figura 49 se presenta un ejemplo de código XSLT específico para extraer las noticias del canal RSS del blog oficial de Oracle¹¹.

¹¹<https://blogs.oracle.com/java/compendium/rss>

```

1 <?xml version='1.0' encoding='UTF-8'?>
2 <xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3   <xsl:output method="xml" indent="yes" encoding="iso-8859-1"/>
4   <xsl:template match="rss">
5     <rssChannel>
6       <label>
7         <xsl:value-of select="channel/title"/>
8       </label>
9       <description>
10        <xsl:value-of select="channel/description"/>
11      </description>
12      <link>
13        <xsl:value-of select="channel/description"/>
14      </link>
15      <updated>
16        <xsl:value-of select="channel/lastBuildDate"/>
17      </updated>
18      <dateFormat>EEE, dd MMM yyyy HH:mm:ss Z</dateFormat>
19      <dateLocale>en-US</dateLocale>
20      <items>
21        <xsl:for-each select="channel/item">
22          <item>
23            <title>
24              <xsl:value-of select="title"/>
25            </title>
26            <description>
27              <xsl:value-of select="description"/>
28            </description>
29            <links>
30              <link>
31                <xsl:value-of select="link"/>
32              </link>
33            </links>
34            <mainImage>
35              <xsl:value-of select="enclosure/@url"/>
36            </mainImage>
37            <author>
38              <xsl:value-of select="dc:creator"
39                xmlns:dc="http://purl.org/dc/elements/1.1/" />
40            </author>
41            <pubDate>
42              <xsl:value-of select="pubDate"/>
43            </pubDate>
44          </item>
45        </xsl:for-each>
46      </items>
47    </rssChannel>
48  </xsl:template>
49 </xsl:stylesheet>

```

Canal de
Noticia
(1)

Formato de fechas (2)

Ítem de
Noticia
(3)

Figura 49: Código XSLT para blog oficial de Oracle.

El código XSLT debe establecer la correspondencia de cada elemento del esquema XSD con el elemento del XML del canal. En primera instancia se establece la correspondencia de la información general del canal de noticias (1), para luego definir el formato de fecha utilizado por el canal (2). Finalmente, se especifica la correspondencia de la información de cada ítem de la noticia (3).

C. Configuración de *pipelines*

La Figura 50 muestra un ejemplo de código de *pipeline* que ejecuta los casos de prueba de un proyecto Maven sobre la plataforma de ejecución Java 1.7, y que posteriormente notifica los resultados al Observatorio.

```
1 stages:
2 - test
3 - webhook
4
5
6 test: Etapa test
7   cache:
8     paths:
9       - node_modules/
10  before_script:
11    - export PATH=$PATH:/usr/local/bin
12
13  image: maven:3.5.4-jdk-7 ← Plataforma de ejecución
14  stage: test
15  script:
16    - mvn install -B
17
18 notify_error: Etapa notificación: error
19   image: appropriate/curl:latest
20   stage: webhook
21   script: >-
22     curl
23     --request POST
24     --url ${WEBHOOK_URL}
25     --header 'Content-Type: application/json'
26     --data '{"buildId": "${BUILD_ID}", "result": "ERROR"}'
27   when: on_failure
28
29 notify_success: Etapa notificación: satisfactorio
30   image: appropriate/curl:latest
31   stage: webhook
32   script: >-
33     curl
34     --request POST
35     --url ${WEBHOOK_URL}
36     --header 'Content-Type: application/json'
37     --data '{"buildId": "${BUILD_ID}", "result": "SUCCESS"}'
38   when: on_success
```

Figura 50: Código de *pipeline* de Gitlab para proyecto Maven sobre Java 1.7

Como se puede apreciar, en una primera instancia se definen tres etapas del *pipeline*: *test*, *error* y *satisfactorio*. La etapa de *test* se encarga de ejecutar, mediante el comando *mvn install -B*, la instalación de dependencias y ejecución de pruebas automáticas sobre un contenedor Docker creado a partir de la imagen oficial de Maven 3.5.4 con la plataforma JDK 7, tal como se indica en la instrucción *image*.

Una vez finalizada la etapa *test*, se ejecutan las etapas de notificación encargadas de detectar y notificar los resultados de las pruebas. Estas etapas se componen de flujos condicionales representados mediante la instrucción *when*, donde se ejecutará uno u otro dependiendo del resultado de la etapa *test*.

Como se explicó anteriormente, la capa de servicios del Observatorio cuenta con un servicio donde el código del *pipeline* puede enviar los resultados, por lo que para notificar los mismos al Observatorio es necesario realizar una petición HTTP. De esta forma y dado que los contenedores Docker utilizados por defecto por Gitlab no contienen la herramienta *cURL*¹² para realizar peticiones HTTP, es necesario indicar que las etapas de notificación se ejecuten sobre un contenedor que contenga *cURL* instalado. Para esto, se seleccionó crear dicho contenedor en base a la imagen de Docker *appropriate/curl:latest*¹³.

Una vez indicada la imagen del contenedor donde ejecutar las etapa de notificación, se indica en la instrucción *script* el comando *cURL* a ejecutar, el cual consiste en una petición POST HTTP con un JSON en el cuerpo del mensaje con uno de los siguientes formatos:

- Cuerpo de mensaje POST de notificación de resultado de pruebas exitosas:

```
1 {
2   "buildId": "${BUILD_ID}",
3   "result": "SUCCESS"
4 }
```

- Cuerpo de mensaje POST de notificación de resultado de pruebas fallidas:

```
1 {
2   "buildId": "${BUILD_ID}",
3   "result": "ERROR"
4 }
```

Si se presta atención en los códigos de *cURL* indicado en las etapas de notificación, se puede apreciar la presencia de los parámetros *WEBHOOK_URL* y *BUILD_ID*. Estos parámetros definen la URL del servicio del Observatorio donde notificar los resultados y el identificador de la ejecución del *pipeline*, respectivamente. Estos parámetros serán reemplazados automáticamente por el Observatorio por los valores correspondientes a la hora de iniciar la ejecución del *pipeline*.

Si por el contrario se desea configurar un código del *pipeline* a ejecutar en el servidor de Jenkins del Observatorio, el código debe seguir con el formato declarativo especificado por

¹²<https://linux.die.net/man/1/curl>

¹³<https://hub.docker.com/r/appropriate/curl/>

Jenkins [91]. En la Figura 51 se presenta un ejemplo de *pipeline* para un proyecto Maven 3.5.4 sobre Java 1.7, cuya lógica es análoga a la del *pipeline* presentado anteriormente.

```

pipeline {
  agent any
  tools {
    maven 'Maven'
    jdk 'jdk'
  }
  stages {
    stage('Clonar Repo') {
      steps {
        git branch: env.BRANCH, url: env.REPO_URL
      }
    }
    stage('Initialize') {
      steps {
        sh """
          echo "PATH = ${PATH}"
          echo "M2_HOME = ${M2_HOME}"
          """
      }
    }
    stage('Build') {
      steps {
        sh 'mvn install'
      }
    }
  }
  post {
    success {
      script {
        def body = """
          {"buildId": "${env.BUILD_ID}", "result": "SUCCESS", "jobName": "${env.JOB_NAME}"}
          """
        def response = httpRequest httpMode: 'POST', requestBody: body, url: env.WEBHOOK_URL
      }
    }
    failure {
      script {
        def body = """
          {"buildId": "${env.BUILD_ID}", "result": "ERROR", "jobName": "${env.JOB_NAME}"}
          """
        def response = httpRequest httpMode: 'POST', requestBody: body, url: env.WEBHOOK_URL
      }
    }
    aborted {
      script {
        def body = """
          {"buildId": "${env.BUILD_ID}", "result": "ABORTED", "jobName": "${env.JOB_NAME}"}
          """
        def response = httpRequest httpMode: 'POST', requestBody: body, url: env.WEBHOOK_URL
      }
    }
    always {
      echo 'Pipeline finalizado'
    }
  }
}
  
```

Figura 51: *pipeline* Jenkins Maven sobre Java 1.7

En la sección *tools* se especifica la instalación de Maven y Java a utilizar dentro de las configuradas en el servidor Jenkins. A continuación se observan dos etapas principales, donde en una primera instancia se clona el repositorio del proyecto, y luego se ejecuta el comando *mvn install* de forma análoga al *pipeline* anterior.

Posteriormente se definen tres etapas a ejecutar según el resultado de la etapa de eje-

cución: *satisfactorio*, en caso de ejecución satisfactoria; *error* en caso de que ocurra algún error; y finalmente *abortado*, en caso de que la ejecución sea cancelada por un usuario desde el panel de administración de Jenkins. A partir de cada uno de estos casos, se realizará una petición HTTP al servicio de notificación del Observatorio indicando el resultado adecuado.