Universidad de la República
Facultad de Ingeniería
Instituto de Computación

Informe de proyecto de grado

Architecture Adviser

Asistente para construcción de arquitecturas en microservicios

Autores:
Mariano Goicoechea
Juan Domínguez

Tutor:
Guzmán Llambías

Cliente:
Carlos Soderguit (Pyxis)

Diciembre 2021

Resumen

Una arquitectura en microservicios es un estilo arquitectónico utilizado para la construcción de un único sistema, el cual está compuesto por muchos componentes o servicios más pequeños que se encuentran débilmente acoplados. A principios del 2010 grandes organizaciones como Ebay y Netflix empezaron a incursionar en esta arquitectura, logrando beneficios como alta disponibilidad, tolerancia a fallos y desarrollo de nuevas funcionalidades rápidamente. Esto llevó a que estas organizaciones evolucionen la mayoría de sus sistemas. El éxito que tuvieron llevó a que otras deseen optar por una arquitectura de este tipo.

Pyxis es una empresa local de desarrollo de software, que se encarga de desarrollar sistemas para distintas organizaciones. Esta migró algunos de sus proyectos existentes a microservicios y comenzó algunos de sus nuevos sistemas utilizando esta arquitectura. Esto llevó a que la empresa cuente con varios sistemas con arquitecturas de este tipo, siendo construido cada uno por un equipo de desarrollo, que cuentan con su autonomía técnica, y define la mejor solución en base a los requerimientos y posibilidades de cada cliente.

Dada la realidad de los distintos sistemas que desarrolló Pyxis, la empresa deseaba realizar un relevamiento de las arquitecturas de los distintos sistemas, generalizar las arquitecturas, y realizar un estudio de la industria y analizar el gap técnico que se tenía con respecto a ellas, para luego analizar los resultados y plantear posibles mejoras. Además de esto, la empresa deseaba contar con una herramienta que ayude en la toma de decisiones al crear proyectos en microservicios, logrando guiar y asesorar a los desarrolladores o arquitectos junior a la hora de crear sistemas con una arquitectura de microservicios.

Como resultado de este proyecto se logró realizar el relevamiento y la generalización de las arquitecturas de microservicio de la empresa, analizar el gap técnico con respecto a un estudio de la industria y realizar sugerencias de mejoras de las mismas. Por último se desarrolló una herramienta que asesora a los usuarios en la construcción de sistemas con arquitecturas de microservicios. Esta herramienta ayuda a crear una arquitectura a partir de los requerimientos no funcionales, guiando al usuario con los elementos que debe tener en cuenta a la hora de construir una sistema, detectar incompatibilidades entre componentes, documentar y obtener información de los componentes de la arquitectura creada. Permitiendo además, que la persona a cargo defina las decisiones que guían al usuario durante el proceso.

Palabras claves: arquitectura de microservicios, modelo de toma de decisiones, asesoramiento en microservicios, relevamiento de arquitecturas.

1 Introducción **7**
1.1 Descripción 7
1.2 Objetivos 8
1.2 Aportes 8
1.4 Organización del documento 8
2 Marco Conceptual **11**
2.1 Microservicios 11
2.2. Máquinas Virtuales y Contenedores 12
2.3 Patrones Arquitectónicos 13
2.4 Ingeniería de caos 20
2.5 Trabajo Relacionado 20
3 Análisis de Requerimientos **25**
3.1 Descripción de la realidad 25
3.2 Etapa Inicial 26
3.3 Etapa Final 27
3.4 Alcance 29
4 Resultado propuesta inicial **31**
4.1 Descripción general 31
4.2 Proceso de trabajo 31
4.3 Arquitecturas Relevadas 33
4.4 Propuestas de mejora 42
5 Architecture Adviser **45**
5.1 Descripción General 45
5.2 Proceso de trabajo 46
5.3 Decisiones de Diseño y Trabajo Relacionado 47
5.4 Descripción del modelo 49
5.5 Arquitectura/Diseño 51
6 Implementación **61**
6.1 Tecnologías 61
6.2 Herramientas de Desarrollo 62
6.3 Calidad de la solución 63
6.4 Dificultades Encontradas 64
7 Gestión del Proyecto **67**
7.1 Planificación inicial 67
7.2 Planificación Final 68
7.3 Metodología de Trabajo 69
7.4 Dificultades encontradas 70
8 Conclusiones y trabajo futuro **73**

8.1 Conclusiones 73
8.2 Trabajo Futuro 74

9 Referencias 77

10 Glosario 80

11 Anexos 82
11.1 Anexo 1: Requerimientos dentro del alcance 82
11.2 Anexo 2: Diagramas de interacción 84
11.3 Anexo 3: Requerimientos fuera de alcance 88

1. Introducción

Este capítulo brinda una introducción a la temática del proyecto de grado. Comienza con una descripción para brindar contexto y motivación de la realización del proyecto. Luego se describen los objetivos y los aportes del mismo. Por último se detalla la organización del documento.

1.1. Descripción

A principios del 2010, una de las arquitecturas utilizadas en las organizaciones eran monolitos. Los sistemas de este tipo iban evolucionando a lo largo del tiempo incrementando su tamaño y complejidad. Cuando la cantidad de usuarios se incrementaba demasiado, tenían que replicar sus sistemas y utilizar balanceadores de carga para que puedan cumplir con la carga esperada. Esto llevó a que los sistemas fueran aún más complejos, difíciles de mantener, utilicen más infraestructura y tengan una salida al mercado más lenta.

Grandes empresas como Ebay y Netflix, comenzaron a evolucionar sus arquitecturas monolíticas a arquitecturas de microservicios, la cual les brindó beneficios a sus sistemas que con las otras arquitecturas se le hacía muy complejo y costoso llevar a cabo [1]. Por ejemplo, unos de los beneficios obtenidos por Netflix fue la mejora de la disponibilidad de su sistema, ya que este pasó a estar compuesto por servicios débilmente acoplados, haciendo que un error afecte solo a los servicios involucrados en lugar de afectar a todo el sistema. El éxito de estas arquitecturas fue tal que llevó a que cada vez más organizaciones deseen migrar sus sistemas o comiencen sus sistemas nuevos con una arquitectura de este tipo.

Por otro lado, Pyxis es una empresa local de desarrollo de software, que se encarga de desarrollar sistemas para distintas organizaciones. Cada sistema es construido por un equipo de desarrollo, que cuenta con autonomía técnica y define la mejor solución en base a los requerimientos y las restricciones impuestas por cada cliente. Una de estas restricciones puede ser por ejemplo, un conjunto de tecnologías a utilizar. Esto hace que cada solución tenga arquitecturas, estrategias y tecnologías distintas haciendo que esta sea única.

Pyxis fue evolucionando sus proyectos a lo largo del tiempo y de los sistemas actuales que tiene la empresa, muchos cuentan con una arquitectura en microservicios. Algunos de ellos fueron construidos de esta forma, mientras que otros existían previamente como monolitos y fueron migrados a este tipo de arquitectura.

La principal motivación de este proyecto es mejorar los productos de Pyxis, para lograr esto se pretende mejorar el desarrollo, los procesos, la documentación y la calidad de sus sistemas basados en una arquitectura de microservicios.

1.2. Objetivos

Los objetivos generales del proyecto son, mejorar las soluciones de Pyxis que siguen una arquitectura en microservicios, conocer la situación actual de estas soluciones en base a la industria y utilizar de mejor manera el tiempo de los arquitectos senior permitiéndoles delegar parte de su trabajo.

En base a los objetivos generales, surgen los siguientes objetivos específicos:

- Realizar un relevamiento de las arquitecturas en los proyectos existentes, para luego realizar una generalización de las arquitecturas aplicadas por Pyxis
- Estudio de la industria y análisis de gap técnico respecto a la arquitectura de Pyxis relevada anteriormente.
- Propuesta de mejoras en las arquitecturas y desarrollo de un prototipo que contenga las mejoras.
- Desarrollo de una herramienta que brinde soporte a la toma de decisiones al crear un sistema con una arquitectura en microservicios.

1.3. Aportes

A continuación, se presentan los principales aportes luego de concluido el proyecto:

- Construcción de modelo para la toma de decisiones que permite guiar el diseño de sistemas basados en una arquitectura de microservicios.
- Desarrollo de una herramienta que asesora a los usuarios en la construcción de sistemas con arquitecturas de microservicios.
- Relevamiento y generalización de las arquitecturas de microservicios de Pyxis.
- Presentación de mejoras a realizar a las arquitecturas de microservicios de Pyxis.
- Conocimiento en microservicios, beneficios, dificultades, patrones utilizados y tecnologías.

1.4. Organización del documento

El contenido del documento está organizado en capítulos, los cuales se presentan a continuación.

En el capítulo 2 se desarrollan los conceptos y definiciones que son necesarios para comprender el resto del documento.

En el capítulo 3 se presenta un análisis de las diferentes problemáticas planteadas durante el proyecto, junto a una descripción de sus requerimientos y alcance del proyecto.

En el capítulo 4 se presenta una descripción general de la problemática de la etapa inicial del proyecto junto a su solución. También se describe el proceso de trabajo llevado a cabo, los resultados obtenidos y por último descripción de los mismos.

En el capítulo 5 se presenta una descripción general de la problemática de la etapa final del proyecto junto a su solución. También se describe el proceso de trabajo llevado a cabo y las decisiones de diseño tomadas. Por último brinda el modelo utilizado y una descripción de la arquitectura del prototipo.

En el capítulo 6 presenta información relevante sobre la implementación del prototipo. En este se comenta las tecnologías y herramientas utilizadas, luego se comenta las distintas estrategias utilizadas para brindarle calidad a la solución utilizada. Por último se comentan las dificultades encontradas en la búsqueda e implementación de dicha solución.

En el capítulo 7 se explica cómo fue llevada a cabo la gestión del proyecto.

En el capítulo 8 contiene las conclusiones del proyecto y el trabajo a futuro del proyecto.

2. Marco Conceptual

En este capítulo se explican los conceptos que se consideran necesarios para la comprensión de este documento. Comienza con breves explicaciones sobre Microservicios, Máquinas Virtuales, Contenedores, Patrones e Ingeniería del Caos. Luego sigue con una sección en donde se mencionan otros trabajos y aplicaciones que están relacionadas a la problemática que se busca resolver en este proyecto.

2.1. Microservicios

Para describir qué es un microservicio primero tenemos que entender qué es una arquitectura en microservicios. Una arquitectura en microservicios es un estilo arquitectónico utilizado para la construcción de un único sistema, el cual está compuesto por muchos componentes o servicios más pequeños que se encuentran débilmente acoplados. Estos componentes o servicios suelen tener su propia base de datos, se comunican a través de Rest APIs, event streaming o message brokers, y pueden ser implementados en distintas tecnologías ya que son independientes del resto de los servicios [2].

Actualmente no existe una definición formal de microservicio, pero existen diferentes características que ayudan a la construcción e identificación de microservicios. No todas las arquitecturas de microservicios tienen todas las características que se presentarán a continuación, sin embargo, se espera que cumplan con la mayoría de ellas.

Las características que suelen cumplir los microservicios son las siguientes:

- Un microservicio es responsable de una única capacidad.
- Un microservicio es desplegado de forma individual.
- Un microservicio consiste en uno o más procesos.
- Un microservicio tiene su propia base de datos.
- Un microservicio es reemplazable.
- Un pequeño equipo puede mantener uno o varios microservicios.
- Un microservicio no tiene restricciones tecnológicas impuestas por otros microservicios.
- Un microservicio es autónomo, es decir que se puede implementar, operar y escalar sin afectar el funcionamiento de otros microservicios.

Muchas entidades como Amazon, Netflix, The Guardian, UK Goverment Digital Service, han migrado sus sistemas a una arquitectura de microservicios. Esto se debe a los beneficios que una arquitectura de este estilo ofrece [3], [4].

Las arquitectura en microservicios tienen ciertos beneficios que las hacen atractivas. El código de los microservicios puede ser actualizado fácilmente, es decir que se puede añadir nuevas funcionalidades sobre los microservicios de forma independiente al resto del sistema. Los equipos pueden utilizar diferentes tecnologías para los distintos microservicios,

permitiendo seleccionar aquellas tecnologías que mejor satisfagan los requisitos de la solución o los conocimientos del equipo. Los componentes pueden ser escalados independientemente, es decir que se pueden generar réplicas de los microservicios de manera independiente para balancear la carga entre más nodos sin la necesidad de replicar el sistema completo, reduciendo el desperdicio de recursos asociado y el costo en infraestructura.

Estos beneficios suelen ser atractivos para sistemas de cierto tipo, pero no todos los sistemas deberían adoptar esta arquitectura ya que no todos los sistemas tienen problemas de tamaño, escalabilidad o infraestructura. Además, utilizar una arquitectura en microservicio también trae consigo muchas dificultades. Para comenzar, migrar un sistema monolítico a microservicios es una tarea difícil de diseñar y llevar a cabo, suele incluir la necesidad de añadir nuevas tecnologías para resolver distintos problemas que se presentan. Algunos aspectos de diseño e implementación como el manejo de errores debe cambiar completamente, ya que el error de un microservicio puede expandirse y generar una falla en todo el sistema. La depuración suele resultar mucho más difícil y en algunos casos hasta inaplicable, se debe contar con un sistema de logs mucho más extenso y preciso que facilite la detección de errores, esto suele incluir la necesidad de nuevas tecnologías. El manejo de datos se vuelve más complejo, ya que los datos del sistema pasan a estar en diferentes bases de datos, y usualmente modificar una base de datos implica la modificación de varias de ellas. Por último, la tarea de testing se vuelve mucho más complicada, ya que no solo hay que testear cada microservicio por separado, sino que hay que testear la integración de todos ellos lo cual suele ser una tarea difícil y costosa [2], [5].

2.2. Máquinas Virtuales y Contenedores

Una máquina virtual es un proceso que se ejecuta en una computadora para crear un entorno aislado que tiene su propia CPU, memoria, interfaz de red y almacenamiento. Este entorno es un componente de software, por lo que no tiene componentes de hardware reales, la máquina virtual puede funcionar sin componentes físicos debido a que utiliza los componentes de la computadora mediante un proceso de virtualización. Mediante este proceso se pueden tener varias máquinas virtuales aisladas en una única computadora, donde cada una de ellas tiene su propio sistema operativo, aplicaciones y procesos [6], [7].

Un contenedor es una unidad de software aislada que puede ser desplegada de forma individual, empaqueta todo el código y las dependencias que una aplicación necesita para que esta pueda ejecutarse de forma rápida y confiable de un ambiente a otro. Tienen todos los elementos necesarios para la ejecución de la aplicación deseada, incluyendo herramientas del sistema, bibliotecas y configuraciones. Muchos contenedores pueden ejecutarse en la misma computadora como procesos aislados compartiendo el sistema operativo [6], [8].

Cuando se trabaja con microservicios se puede utilizar cualquiera de estos dos enfoques, sin embargo la utilización de contenedores ofrece ciertas ventajas. Como puede observarse en la figura 1, cada máquina virtual debe tener instalado el sistema operativo, a diferencia de los contenedores que utilizan el sistema operativo en el que están siendo ejecutados. Esto genera ciertas diferencias, al no incluir el sistema operativo, los contenedores resultan mucho más

livianos, ocupando en general decenas de MB, mientras que las máquinas virtuales suelen ocupar decenas de GB. Por este motivo los contenedores requieren menos memoria y se puede tener muchos más de ellos, permitiendo ejecutar más microservicios con la misma cantidad de memoria. Otra diferencia es que al tener que iniciar un sistema operativo completo, las máquinas virtuales requieren un tiempo mayor para iniciar su ejecución, dado que los contenedores no deben iniciar el sistema operativo, el tiempo que requieren para iniciar suele ser muy corto.

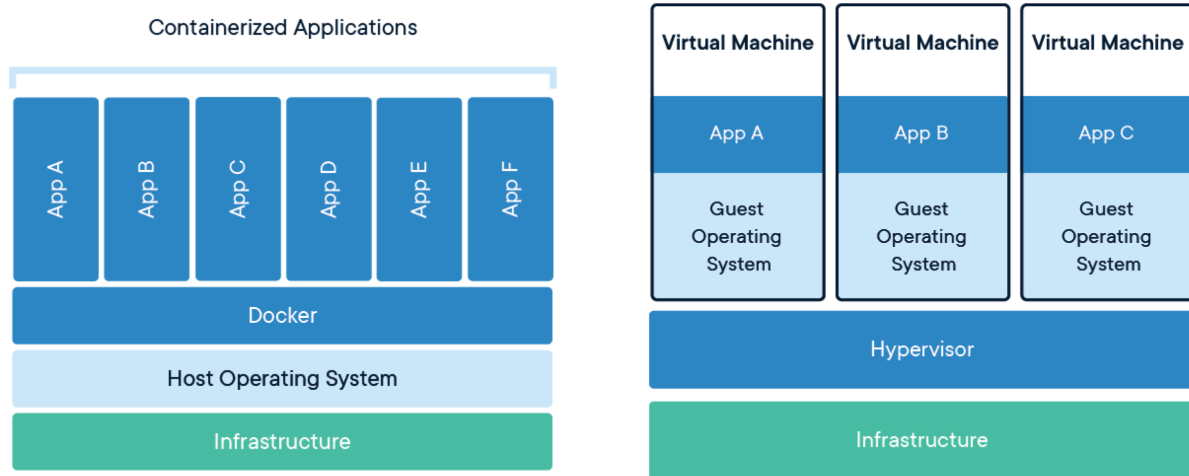


Figura 1. Funcionamiento de contenedores y máquinas virtuales [6].

2.3. Patrones Arquitectónicos

A continuación se presentan algunos patrones arquitectónicos que resulta importante conocer para comprender completamente el informe.

2.3.1. Api Gateway

Este patrón crea una única entrada de acceso a los servicios del sistema, actuando de intermediario entre el cliente que realiza una solicitud al sistema y los microservicios que pueden atenderla, logrando que el cliente vea al sistema como una única unidad a la cual envía solicitudes. La mayoría de veces, en un Api Gateway los servicios del sistema se exponen como una API Rest para los distintos tipos de clientes, por ejemplo clientes web o de aplicaciones móviles.

En un sistema de microservicios los clientes podrían tener acceso a las distintas APIs que brinda cada microservicio. Como los microservicios realizan funcionalidades tan específicas, el cliente se podría encontrar con la dificultad de que para realizar una funcionalidad en su sistema tenga que hacer un conjunto de solicitudes a distintos microservicios, teniendo que realizar el trabajo de componer y transformar las distintas respuestas, además de encargarse de la comunicación con el registro de servicios del sistema.

Cuando un sistema cuenta con un Api Gateway, el cliente tiene acceso a los servicios que brinda el sistema en su conjunto, en donde el Api Gateway se encarga de verificar que en cada solicitud el cliente se encuentre autorizado, luego se comunica con el registro de servicios para obtener las ubicaciones de los distintos microservicios, realiza las solicitudes, las transformaciones y composición de respuestas que se precisen. Otros beneficios que tiene Api Gateway es que los clientes no deben preocuparse por los distintos cambios de versiones que puede tener un microservicio, ya que al existir un único punto de entrada, este se encarga de realizar el cambio [9].

2.3.2. Circuit Breaker

Este patrón busca solucionar un problema que se presenta cuando se utilizan llamadas remotas. Las llamadas remotas pueden fallar de manera rápida o mantenerse activas hasta agotar el tiempo disponible y generar un timeout, incluso podría suceder que muchos clientes realicen consultas a un proveedor que no está respondiendo. Esto genera que en cierto momento se acaben los recursos, desencadenando una serie de errores en cascada en distintas partes del sistema.

Para solucionar esto, el patrón propone evitar los timeouts innecesarios y saturación del proveedor utilizando una función o intermediario al que llamaremos circuit breaker para monitorear las fallas del proveedor (observe la figura 2).

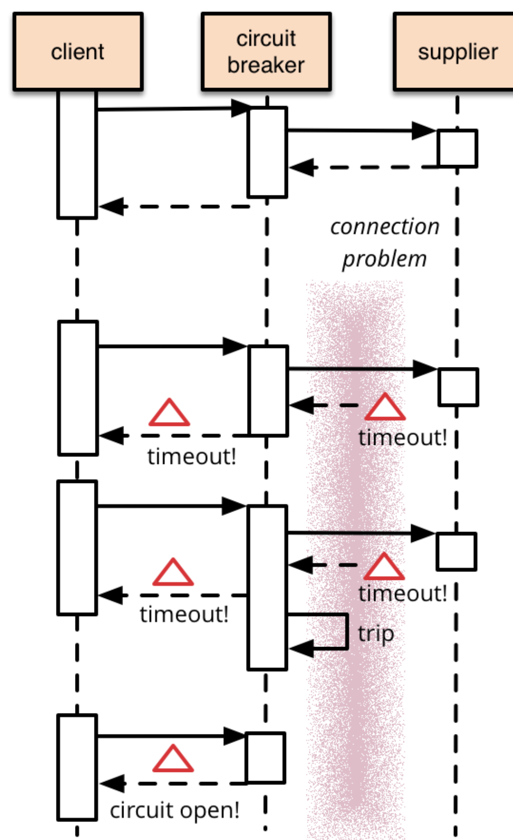


Figura 2.Funcionamiento de Circuit Breaker [10].

El circuit breaker tiene dos estados, abierto y cerrado. Cuando comienza la ejecución su estado pasa a estar cerrado, intersectando y redirigiendo todas las consultas que le llegan al proveedor. Cuando la frecuencia de fallas del proveedor sobrepasa un umbral definido, el circuit breaker cambia al estado abierto, en el cual deja de redirigir las consultas al proveedor, en su lugar responde inmediatamente con un error. De esta forma el circuit breaker evita la acumulación de consultas pendientes, generando tiempos de respuesta rápidos y dándole tiempo al proveedor para recuperarse en lugar de saturarlo con más consultas [10].

2.3.3. Service Discovery

Este patrón sirve para solucionar un problema que se presenta en sistemas distribuidos. Cuando se tienen muchos servicios distribuidos se debe poder ubicar a cada uno de ellos para realizar consultas, sin embargo, las direcciones IP y puertos de estos servicios son dinámicas, es muy común que cambien debido a fallas o actualizaciones.

Existen dos patrones de Service Discovery mayormente utilizados:

Client Side Discovery Pattern
En este caso, la aplicación cliente es responsable de determinar las direcciones de los servicios y balancear la carga. Para esto debe consultar el Service Registry, una base de datos que contiene todos los servicios disponibles [11].

Server Side Discovery Pattern
En este caso, el cliente realiza una consulta a un intermediario que suele encargarse del balanceo de carga. El intermediario se encarga de realizar la consulta al Service Registry y redirigir cada consulta a donde corresponda [11].

2.3.4. Service Registry

Este patrón es una parte clave del patrón Service Discovery, ya que es la base de datos encargada de contener todos los datos necesarios para realizar una consulta a cualquier instancia de servicio del sistema. Cualquier consulta que desee realizar una aplicación cliente a cualquier servicio será a través del Service Registry, por lo que debe ser de alta disponibilidad.

Es importante que los datos de la base estén siempre actualizados, ya que de lo contrario no se podrán redirigir las consultas, lo cual causará un error en la aplicación cliente. Para esto se debe contar con un sistema de registro que mantenga al Service Registry actualizado. Los dos patrones principales utilizados para esta tarea son Self-Registration y 3rd Party Registration. Para el caso de Self-Registration, cada servicio tiene la responsabilidad de registrarse en el service registry una vez que se encuentra activo, mientras que en 3rd Party Registration existe un tercero encargado de detectar cuando un microservicio pasa a estar activo o inactivo, para luego actualizar el Service Registry [11].

2.3.5. Health Check Api

A medida que un proyecto crece y la cantidad de servicios se incrementa, se vuelve más difícil mantener la observabilidad de qué servicios están en funcionamiento y qué servicios tienen alguna falla, una forma de solucionar esto es utilizar el patrón Health Check Api. Este patrón consta de un componente que se encarga de realizar consultas a los distintos servicios del sistema de forma constante, para verificar si estos están funcionando de manera correcta. Para poder realizar esto, cada servicio debe ofrecer un endpoint para que el componente Health Check consulte su estado.

El endpoint ofrecido por los servicios puede ser sencillo como responder un "OK" cada vez que esté disponible, o puede ser más riguroso y verificar ciertos elementos como la conexión a una base de datos, a una memoria caché, o incluso verificar la performance del momento [12].

2.3.6. Saga

El patrón saga se utiliza para mantener la coherencia de los datos en los microservicios. Cuando se trabaja sobre una única base de datos, este problema no se presenta, ya que se cuenta con transacciones, las cuales permiten ejecutar varias operaciones que automáticamente serán descartadas en caso de error. En los sistemas de microservicios es común que para realizar una funcionalidad se precise la colaboración de más de un microservicio, al implicar varios microservicios, se utilizan varias bases de datos, y no se puede utilizar las transacciones de la misma manera. Se debe lograr que los datos se mantengan coherentes cuando hay un error, para eso se debería volver atrás todos los cambios que se hicieron antes de suceder el error.

Una saga es un conjunto de transacciones locales que debe llevar a cabo un conjunto de microservicios para realizar una funcionalidad, si en el transcurso de la funcionalidad sucede un error en la transacción local que realiza un microservicio, entonces se realiza un conjunto de transacciones compensatorias. Las transacciones compensatorias son transacciones locales que revierten los cambios hechos en la base de datos de los distintos microservicios involucrados en la saga que se utilizó para realizar la transacción de negocio [13].

Hay dos formas de implementar el patrón saga:

Coreografía
Esta forma consiste, en que cada transacción local que compone la saga, se encarga de publicar un evento que desencadena otra transacción local en otro servicio. De esta forma, cada participante es el encargado de coordinar las siguientes transacciones.

Es habitual que en las sagas de este tipo se utilice un bus de eventos. Para implementarlo se utiliza una cola de mensajes, en la cual los servicios se suscriben para publicar y/o recibir eventos pertenecientes a la saga [13].

Orquestación
Esta forma consiste en la existencia de un proceso o servicio centralizado llamado orquestador, que se encarga de realizar la coordinación de las transacciones que se deben

hacer para cumplir con una transacción de negocio. El orquestador ejecuta las solicitudes correspondientes de la saga, almacena y comprende los estados de cada tarea. En caso de existir un error en alguna transacción, se encarga de coordinar las transacciones de compensación, revirtiendo los cambios realizados en el transcurso de la transacción [13].

2.3.7. Transactional Outbox

El patrón Saga se basa en una secuencia de transacciones que actualiza a cada servicio y publica un mensaje o evento para desencadenar el próximo paso del flujo. Si un paso falla, Saga ejecuta transacciones compensatorias para contrarrestar las transacciones anteriores.

Una situación que se puede dar en Saga se puede observar en la figura 3 y es que un servicio actualice su base de datos sin ningún problema, y una vez que hace el commit de la transacción envía un mensaje/evento al broker para seguir con el flujo. Pero si ocurre un error que evite que el mensaje sea colocado en el broker, supongamos por ejemplo que se cae el servicio. El sistema queda en un estado inconsistente, el primer servicio terminó su transacción y realizó los cambios en su base de datos, pero nunca llegó a envíar el mensaje para notificar a los otros servicios, por lo que estos no ejecutan ningún cambio [14].

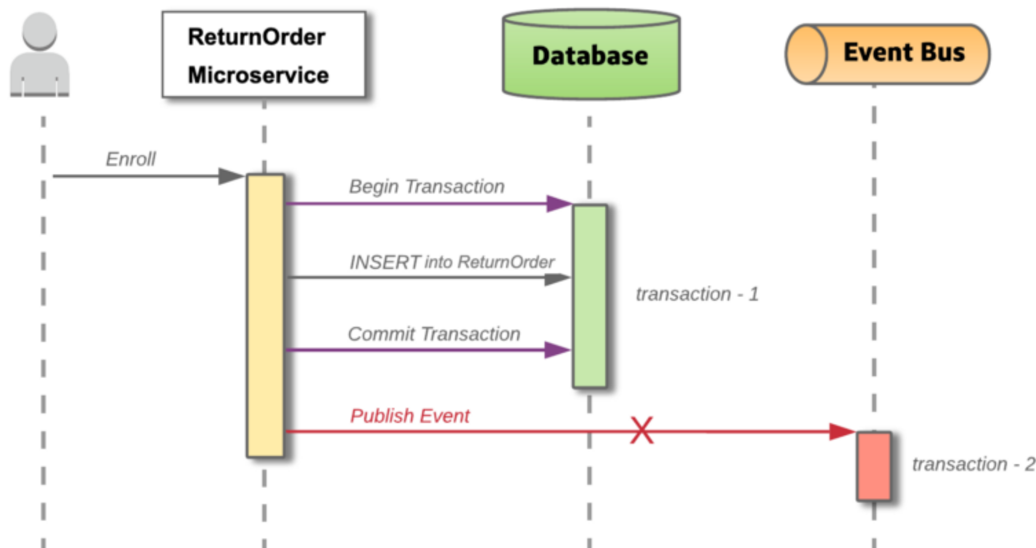


Figura 3. Problema que surge en Saga [15].

Una forma de solucionar esto es mediante el patrón Transactional Outbox. Como se muestra en la figura 4, el patrón transactional outbox coloca en la base de datos una tabla (en este caso llamada Outbox table) en la que se guardan los eventos que deben ser enviados. Luego el componente Message relay se encarga de leer de esa tabla para obtener el evento y enviarlo por el broker. Esto soluciona el problema presentado anteriormente porque la inserción en las tablas se realiza en la misma transacción.

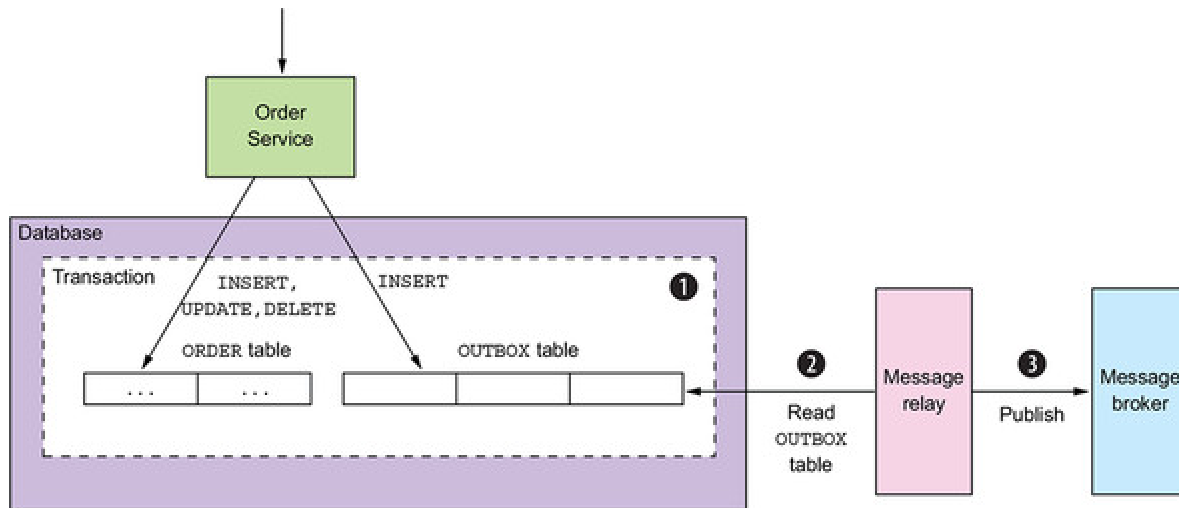


Figura 4. Funcionamiento Transactional Outbox [14].

Si pensamos en la misma situación que teníamos antes, en este caso al completarse la transacción, va a quedar un registro en la tabla outbox table. Una vez que el servicio vuelva a iniciar, el message relay se encargará de leer ese registro de la tabla, enviar el evento al message broker y posteriormente borrar el evento de la tabla, asegurando de esta manera la consistencia de los datos del sistema [14], [16].

Existen dos formas de implementar Transactional Outbox, conocidos como los patrones Polling Publisher y Transaction Log Tailing. Estos patrones difieren en la implementación del Message relay.

Polling Publisher consta en que el Message Relay se encargue de consultar la tabla de forma constante, es sin duda el más sencillo de implementar pero no es muy escalable. A medida que la aplicación crece el costo pasa a ser muy alto y puede causar problemas de performance.

Transaction Log Tailing no cuenta con este inconveniente, la idea base es que cada acción que se ejecuta sobre la base de datos se guarda en el log de transacciones, un Transactional Log Miner se encarga de leer los logs y por cada entrada genera un evento correspondiente que será envíado al message broker. Además todo lo relacionado a Transactional Log Miner sucede a nivel de base de datos, por lo que no requiere cambios a nivel de aplicación [14].

2.3.8. Contract Testing

Es una metodología que ayuda a verificar que la comunicación entre las distintas aplicaciones (o los distintos microservicios del sistema) cumplan ciertas reglas definidas en lo que se llama un contrato, de esta forma se asegura que estas aplicaciones o microservicios sean compatibles [17].

Para las aplicaciones que se comunican por HTTP el consumidor es quien inicia la consulta y el proveedor el que la responde. Para las aplicaciones que se comunican usando colas de mensajes, el consumidor será el que obtiene el elemento de la cola de mensajes, mientras que el proveedor el que lo inserta.

La idea detrás de Contract Testing es que exista un contrato propuesto por una de las partes y aceptado por la otra. Este contrato define los parámetros de la comunicación entre un consumidor y un proveedor. De esta manera, como se muestra en la figura 5, se pueden crear servicios del tipo mock en base al contrato para que cada una de las partes pueda realizar pruebas antes de subir una nueva versión, y así verificar si están cumpliendo con los parámetros que necesita la otra parte.

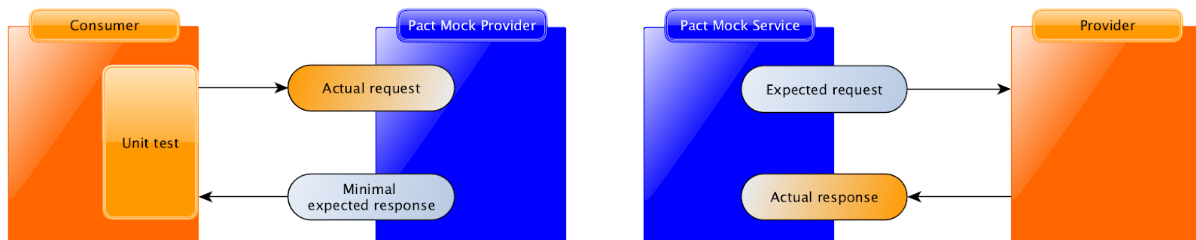


Figura 5. Funcionamiento Contract Testing Consumer And Provider [18].

Con estas pruebas el consumidor puede determinar si su comportamiento es adecuado al recibir la respuesta mínima que promete el proveedor, y el proveedor puede verificar si la respuesta que está entregando cumple con lo que requiere el consumidor [18].

Cuando se compila el proyecto proveedor se crea un test y un stub en base al contrato, este stub se coloca en un jar que luego se puede guardar en un artifactory para ser descargado y utilizado por el cliente. Esto soluciona el problema de subir nuevas versiones de los servicios proveedores. Un error que suele ocurrir con frecuencia en una arquitectura distribuida como en microservicios es que alguno de los servicios realiza un cambio y sube una nueva versión generando errores en los consumidores. Otro problema que suele surgir es que a pesar de que los consumidores generan tests de integración para verificar la comunicación con los proveedores, estos tests suelen quedar desactualizados a medida que los distintos proveedores realizan cambios.

Provider Driven Contract Testing:
En este caso el proveedor es quien define el contrato en base a la evolución de la API que provee. El proveedor genera una nueva versión de su API con un nuevo contrato que define los parámetros de la misma, luego al momento publicar su nueva versión actualiza el contrato en algún manejador de bibliotecas o repositorio. Luego el consumidor al compilar su proyecto obtendrá la nueva versión del contrato, con la cual se generan nuevos casos de prueba para realizar las pruebas.

Consumer Driven Contract Testing:
En este caso el consumidor es quien define los parámetros del contrato y lo publica en algún manejador de bibliotecas o repositorio al cual el proveedor tiene acceso. Luego el proveedor obtendrá la nueva versión del contrato, se generarán los nuevos casos de prueba y podrá verificar si cumple con los requisitos [19].

Las principales ventajas que ofrece el testing por contrato son las siguientes:

1. Son rápidos de ejecutar debido a que no deben comunicarse con múltiples sistemas.

2. Son fáciles de mantener.
3. Son fáciles de depurar y arreglar.
4. Son escalables ya que cada componente se prueba de manera independiente.
5. Posibilita la implementación del consumidor antes de disponer del proveedor.
6. Dado que las pruebas son locales, ayudan a descubrir bugs en etapas tempranas antes de desplegar [20].

2.4. Ingeniería de caos

Ingeniería de caos es una técnica que trata de poner a prueba un sistema distribuido provocando fallos intencionalmente. Para llevar esto a cabo se realiza un experimento en el cual introducen al sistema eventos que pueden surgir en un ambiente productivo. Por ejemplo, que dejen de funcionar bases de datos o servicios, que ocurran cortes o latencias en la red. De esta forma lograr ver cómo se comporta el sistema y cuál es su tolerancia a los distintos tipos de fallo.

Este experimento se realiza cuando el sistema se encuentra en producción y todo el equipo de trabajo se encuentra disponible, esto se hace para que el sistema esté en un estado real de ejecución, ya que por ejemplo sería muy costoso simular el tráfico real.

Con Ingeniería de caos se logran detectar las posibles debilidades del sistema para corregirlas, evitar situaciones inesperadas y construir sistemas más robustos [21].

2.5. Trabajo Relacionado

A continuación se presentan las aplicaciones y artículos estudiados que resultaron relevantes durante la búsqueda de una solución.

2.5.1. Microservice Architecture Assessment Platform

Microservices architecture assessment platform es una herramienta de asesoramiento para la construcción de software implementada por Chris Richardson. Esta herramienta consta de un conjunto de categorías de software, sobre las cuales se le realizan preguntas al usuario con respecto a su arquitectura. Presenta preguntas en forma de múltiple opción con una sola elección posible.

Una vez que el usuario responde todas las preguntas, el sistema analiza las respuestas ingresadas y califica los distintos aspectos de la arquitectura con valores de 0 a 100, ofreciendo gráficos para una visualización más simple [22].

2.5.2. IBM architectural Model

IBM presenta en su página web un centro de arquitecturas en la nube, en la cual se proveen distintas prácticas para la construcción de aplicaciones en la nube. IBM busca identificar el problema al que se enfrentan sus clientes presentando distintas opciones de las cuales elegir para que puedan seleccionar las soluciones que mejor se adaptan a sus necesidades.

Para esto utilizan un modelo de arquitecturas en el que se presentan diferentes tipos de arquitecturas. Este modelo se presenta de una forma muy interesante en la que se muestran los distintos tipos de arquitecturas junto a sus características más importantes, permitiendo ingresar a cada una de ellas para obtener una pequeña descripción de las características que se mostraban inicialmente, y permitiendo a su vez ingresar a cada una de estas para dirigirse a una página con información más detallada [23].

2.5.3. Capturing software architecture knowledge for pattern-driven design

Este artículo [24] busca ayudar a los arquitectos de software en la toma de decisiones mediante un proceso que ayuda a relacionar de forma directa patrones de diseño y atributos de calidad. Identifica la construcción de arquitectura de software como un proceso de toma de decisiones en el que un arquitecto considera varias soluciones posibles para resolver un problema y luego selecciona una que considera la más óptima.

El artículo identifica el diseño de una arquitectura de software como un proceso de cuatro pasos:

1. El arquitecto de software se enfrenta a un problema de diseño.
2. Busca posibles soluciones para resolverlo.
3. Analiza la descripción de varios patrones e identifica varios candidatos.
4. Identifica un patrón óptimo para solucionar su problema y verifica las tácticas para asegurarse que funciona dentro de su contexto.

Con el modelo que presenta en la figura 6 pretende solucionar los pasos 2 y 3 de este proceso [24].

Figura 6. Modelo basado en ISO/IEC/IEEE standard 42010 (ISO, 2011a)
para la toma de decisiones en arquitectura de software [24].

El principal enfoque de este artículo está dirigido a los patrones, las tácticas y los atributos de calidad, realizando una exhaustiva investigación y análisis para estudiar cómo estos se relacionan y concentrar toda esta información en un único lugar. Existen muchos patrones en el mundo académico pero cuando se busca responder a ciertas preguntas sobre ellos como por ejemplo, qué atributos de calidad satisface, resulta difícil encontrar información completa y consistente al respecto, ya que *"el conocimiento sobre patrones de arquitectura se encuentra disperso a través de diferentes estudios literarios"* [24]. Cuando se quiere analizar varios componentes no se tiene una fuente unificada que contenga toda la información sobre ellos, por lo que se debe realizar un costoso trabajo de investigación y análisis para responder las preguntas que uno se plantea, adicionalmente, resulta escasa la información que relaciona atributos de calidad y patrones directamente, *"la falta de conocimiento suficiente sobre patrones y sus impactos en atributos de calidad, sumado a sus aplicaciones de dominio en literatura, impide el progreso en el campo de arquitectura de software"* [24].

2.5.4. Assisting software architects in architectural decision-making using Quark

El artículo [25] presenta el método Quark, un método para asistir a los arquitectos de software al momento de tomar decisiones, y para conceptualizar la relación entre

requerimientos no funcionales y decisiones arquitectónicas, para poder así presentar y manejar el conocimiento sobre el campo.

Se centra en la idea de que los requerimientos no funcionales expresan ciertas cualidades deseadas en el sistema que se busca desarrollar, como lo pueden ser performance, disponibilidad, mantenibilidad, portabilidad, etc.

En Quark, el arquitecto de software emplea el rol central, se encarga de especificar los requerimientos no funcionales y las restricciones, también se encarga de seleccionar entre las decisiones arquitectónicas y decidir cuándo un proceso debe terminar. Quark se encarga de notificar sobre posibles incompatibilidades y presentar posibles acciones para resolverlas, pero el método no requiere que dichas acciones sean realizadas para continuar con el diseño, ya que no es obligatorio resolver las incompatibilidades, esta decisión queda en manos del arquitecto.

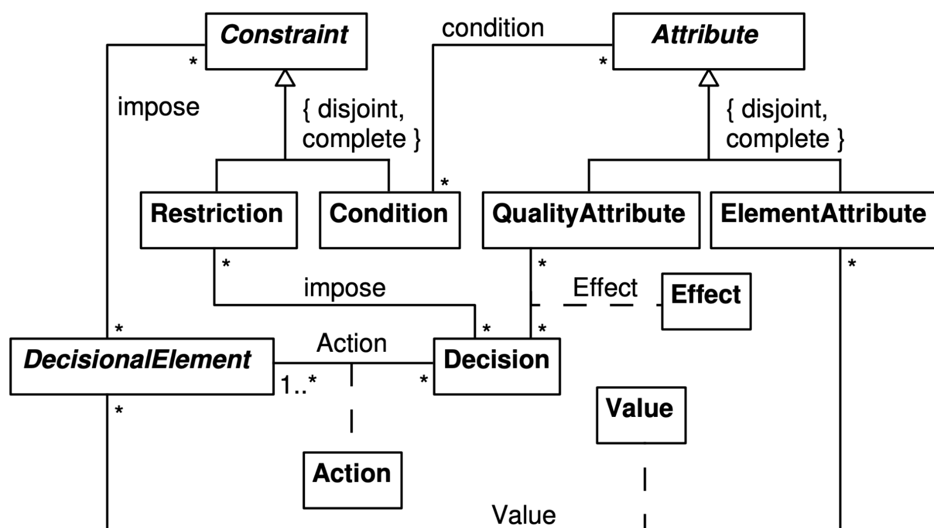


Figura 7. Modelo de toma de decisiones utilizado en Quark [25].

Quark utiliza un proceso iterativo en cuatro etapas que permite al arquitecto en todo momento realizar una nueva iteración para refinar más su arquitectura, y utiliza el modelo en la figura 7 para la toma de decisiones.

Tras la selección de requerimientos no funcionales y condiciones que realiza el arquitecto en el primer paso, Quark se encarga de recomendar un conjunto de *DecisionalElements*, los cuales impondrán *Decisions* que están determinados por el elemento *Action* (por ejemplo utilizar o excluir). Cuando se cumple un *Decision* se pueden generar nuevas restricciones para la arquitectura, lo cual impactará en la lista de *DecisionalElements* que se presentan al arquitecto en la siguiente iteración, también se puede generar la necesidad de nuevos *QualityAttributes*, lo cual generará *Conditions* que nuevamente impactarán en la lista de *DecisionalElements* que se presentan al arquitecto en la siguiente iteración [25].

3. Análisis de Requerimientos

Este capítulo presenta un análisis de las diferentes problemáticas planteadas durante el proyecto, junto a una descripción de sus requerimientos.

En este proyecto se pueden identificar dos grandes etapas debido a un cambio de objetivos que ocurrió durante su transcurso. Una primera etapa donde se busca realizar un relevamiento de arquitecturas para analizarlas y realizar sugerencias de mejoras, y una segunda etapa en donde se busca construir un sistema que ayude a los usuarios a construir una arquitectura con microservicios.

El capítulo está dividido en cuatro secciones. En primer lugar se realiza una descripción de la realidad, donde se explicarán las problemáticas planteadas en las dos etapas descritas, explicando además cuál fue la causa que motivó este cambio de objetivo. En las segunda y tercera sección se verán por separado las etapas recién mencionadas, realizando una descripción más detallada de sus requerimientos. Por último, en la cuarta sección se explicará el alcance definido para la herramienta construida durante la segunda etapa del proyecto.

3.1. Descripción de la realidad

Pyxis es una empresa local de desarrollo de software la cual posee varios proyectos que siguen una arquitectura de microservicios. Cada equipo cuenta con autonomía y define la mejor solución en base a las posibilidades de su cliente, lo cual genera que dentro de la empresa existan diferentes instancias de arquitecturas de microservicios. Tener muchas arquitecturas diferentes dificulta la documentación de las mismas ya que se debe realizar para cada una de ellas por separado, sin embargo, a pesar de que cada equipo es autónomo, la mayoría de estas arquitecturas tienen muchas similitudes.

De aquí surge la primera necesidad de Pyxis con respecto a este proyecto, realizar un relevamiento de las arquitecturas y generar una representación general que incluya sus similitudes. Una vez obtenida esta representación, se podría realizar un análisis más detallado de sus generalidades para evaluarlas y buscar posibles mejoras de las cuales, se implementaría un prototipo. Esto describe la realidad de la primera etapa del proyecto, la cual decidimos llamar *"etapa inicial"*.

La segunda etapa, la cual identificamos como *"etapa final"* surge debido a un cambio de objetivos que se presenta durante el transcurso del proyecto. Tras el trabajo de relevamiento, se llegó a un resultado final que intenta representar lo mejor posible las arquitecturas relevadas. Luego de validar dicho resultado con el cliente, propusimos un conjunto de posibles mejoras que consideramos podrían ayudar a la arquitectura. Si bien al cliente le parecieron interesantes algunas de las opciones, decidió que prefería aprovechar este proyecto para realizar otra herramienta que le aportaría más valor.

El objetivo de Pyxis desde un principio fue mejorar sus soluciones orientadas a microservicios. Luego de tener una idea más clara de la calidad de dichas soluciones, decidieron cambiar el enfoque. El objetivo seguía siendo el mismo, pero en lugar de apuntar directamente a mejorar sus arquitecturas, decidieron implementar un sistema que los ayudaría a lograr este objetivo mejorando sus procesos de desarrollo y documentación.

El nuevo objetivo es diseñar e implementar un sistema, el cual mediante un conjunto de preguntas, componentes y otros elementos, permite al usuario realizar el relevamiento de una arquitectura en microservicios. La idea principal es que los arquitectos con más conocimientos definan ciertas reglas para determinar las arquitecturas recomendadas por el sistema. Luego los desarrolladores o arquitectos con menos conocimientos podrían apoyarse en este sistema para la construcción de las arquitecturas, tan solo deberían responder un conjunto de preguntas definidas por los encargados del sistema para obtener una arquitectura recomendada.

3.2. Etapa Inicial

La propuesta original para este proyecto consistió en realizar un relevamiento de las arquitecturas de microservicios utilizadas por el cliente, identificando tanto componentes arquitectónicos, como patrones y tecnologías, buscando de esta forma obtener una generalización de las arquitecturas de microservicios utilizadas por la empresa. Una vez obtenidas se pretendía investigar posibles mejoras que podrían ser aplicadas sobre ellas, para posteriormente presentar al cliente y mediante un proceso de selección definiera un subconjunto de estas, para las cuales se implementaría un prototipo.

Para esta etapa se identificaron los siguientes requerimientos:

Relevamiento de la situación actual:
Se debe realizar un relevamiento de la situación actual de la empresa mediante distintas instancias que involucren el intercambio de información con el cliente.

Identificación de la arquitectura de Pyxis aplicada actualmente:
Mediante el relevamiento anterior se deben identificar y plasmar visualmente las arquitecturas de microservicios utilizadas por Pyxis, identificando componentes, patrones y tecnologías utilizadas.

Análisis del estado del arte en microservicios y arquitectura de software:
Realizar un estudio del estado del arte sobre microservicios y arquitectura de software buscando identificar posibles diferencias técnicas y/o tecnológicas con respecto a los resultados del relevamiento.

Propuesta de mejoras en base al análisis anterior:
Buscar y presentar nuevas prácticas, patrones y/o tecnologías que puedan mejorar las arquitecturas relevadas en cualquier aspecto relevante para el cliente.

Prototipos de nuevas tecnologías:

Tras todos los requerimientos anteriores, se debe implementar un prototipo de aquellas propuestas acordadas.

3.3. Etapa Final

El objetivo de la etapa final consiste en crear el prototipo de un sistema que permita a los usuarios (quienes serán desarrolladores o arquitectos junior de la empresa) ser asesorados para la construcción de una arquitectura en microservicios. Utilizando este sistema los usuarios podrán responder una serie de preguntas sobre la aplicación que desean implementar y en base a las respuestas ingresadas, el sistema recomendará un conjunto de componentes, patrones y tecnologías a utilizar junto con algunas fuentes de información que ayudarán al usuario con su implementación.
Mediante la utilización de este sistema la empresa podrá tener un historial de los proyectos realizados que permitirá facilitar los procesos de documentación. Además los arquitectos a cargo de la empresa, podrán delegar parte de su trabajo apoyándose en arquitectos o desarrolladores con menos conocimientos junto a la utilización de este sistema. De aquí surge otra gran necesidad que este sistema debe cumplir, los arquitectos a cargo deberán poder moldear la toma de decisiones a su gusto. En otras palabras, esto quiere decir que ellos deben poder gestionar las preguntas y respuestas del sistema definiendo ellos mismos cómo estas repercuten sobre la decisión de recomendar o no algún componente en el resultado final. Esto no solo implica permitir la modificación de preguntas y respuestas, sino también ser capaz de quitar componentes deprecados o añadir nuevos componentes que la persona a cargo considere relevante.

Resulta de suma importancia que el sistema cumpla con lo descrito en el párrafo previo, ya que impactará directamente en su capacidad para adaptarse. La construcción de software evoluciona constantemente, generando que surjan nuevos estilos arquitectónicos, patrones y tecnologías. Un sistema que no tenga la capacidad de adaptarse a estos cambios puede perder valor y utilidad fácilmente, lo cual genera costo de mantenimiento que el cliente podría considerar demasiado alto para pagar.

Requerimientos

Los requerimientos del sistema fueron definidos junto al tutor y el cliente en base a los objetivos deseados. Se definió una lista inicial de requerimientos lo suficientemente extensa para que en ningún momento surgieran tiempos ociosos debido a falta de trabajo. Estos requerimientos conformaron un backlog sobre el cual posteriormente se definió un alcance que satisfizo las necesidades del cliente y acotó el tiempo estimado para colocarse dentro de rangos de duración que consideramos razonable para un proyecto de este tipo.

El sistema previamente descrito debe cumplir con los siguientes requerimientos:

Gestión de proyectos:
Se debe poder crear, listar, acceder, modificar y eliminar proyectos del sistema.

Responder Preguntas:

Se debe poder visualizar correctamente las preguntas de un proyecto separadas por categoría, permitiéndole al usuario ingresar sus respuestas y validarlas.

Generar, visualizar y acceder dimensiones de un proyecto:
Una vez finalizada la etapa de preguntas se debe presentar al usuario una lista de las dimensiones del sistema, las cuales contendrán los componentes que correspondan y podrán ser accedidas para visualizarlos.

Generar, visualizar, seleccionar y acceder componentes de un proyecto:
Una vez finalizada la etapa de preguntas se deben generar los componentes del proyectos, a los cuales se podrá acceder a través de las dimensiones. Al acceder a una dimensión se deben mostrar los componentes de la misma, permitiendo al usuario desmarcar aquellos que considere innecesarios. Por último se debe poder acceder a los componentes de la lista para visualizar su descripción y sus fuentes de información.

Listado y acceso a fuentes de información de un componente:
Tras acceder a un componente se deben listar sus fuentes de información, permitiendo acceder a ellas en caso de ser una página externa, o descargarlas en caso de ser un archivo pdf.

Extensibilidad:
El encargado del sistema debe tener control total de su funcionamiento, es decir que el sistema debe continuar su funcionamiento correctamente aunque el encargado decida modificar sus preguntas, componentes o incluso la toma de decisiones misma. La toma de decisiones del sistema y cualquier elemento relevante a la misma que se le presenta al usuario debe permitir la extensibilidad del sistema.

Versionado:
Debe existir un concepto de versionado del modelo de datos del sistema, permitiendo así que en caso de que se desee modificar los elementos del sistema se pueda generar una versión nueva manteniendo la versión actual y sus proyectos.

Facilidad de uso:
La aplicación debe ser intuitiva, sencilla de utilizar y fácil de entender para los usuarios.

Aplicación responsive:
La aplicación cliente debe ser responsive para poder utilizarse desde cualquier dispositivo que resulte conveniente.

Aplicación Backoffice:
Debe existir una aplicación backoffice, o un modo administrador dentro de la aplicación que permita la gestión del modelo de datos utilizado, permitiendo ingresar nuevas versiones y realizar todas las funcionalidades relacionadas al requerimiento de extensibilidad. Esto incluye gestión de categorías, preguntas, respuestas, componentes, dimensiones, fuentes de información y de la toma de decisiones.

Para una explicación más detallada de los requerimientos puede ir al anexo 1.

3.4. Alcance

Desde la finalización del relevamiento se observó que la cantidad de requerimientos relevados para el sistema completo era demasiado grande y escapaba del alcance de este proyecto, fue por esto que se realizó una negociación con el cliente para identificar aquellos requerimientos de mayor prioridad y definir un alcance que resulte más razonable.

Tras una conversación con el cliente se llegó a la conclusión de que la aplicación de backoffice no era prioritaria, por lo que se dejó como un posible trabajo a futuro y nos enfocamos en los requerimientos de la aplicación cliente. Sin embargo, si bien los requerimientos del backoffice no fueron implementados estrictamente en este proyecto, se mantuvo el requerimiento de extensibilidad para que la implementación de la aplicación backoffice sea una posibilidad a futuro. Esto significa que al momento de tomar cualquier decisión de diseño importante sobre la solución de la aplicación cliente, debíamos tener en cuenta los requerimientos de backoffice. El requerimiento de extensibilidad se convirtió en uno de los de mayor prioridad, es lo que diferenciaría a una solución de una única aplicación de una solución aplicable a distintas realidades y contextos.

4. Resultado propuesta inicial

En este capítulo se presenta la solución para la problemática de la etapa inicial del proyecto. Está constituido por una descripción general en la que se proporciona un poco de contexto para el mejor entendimiento de la solución, una sección de proceso de trabajo en donde explica cuál fue la forma en la que se trabajó durante esta etapa, dos secciones en las que se explicarán cada uno de los resultados obtenidos, y por último una sección en la que se describen las propuestas de mejora que se le presentaron al cliente.

4.1. Descripción general

Los proyectos de microservicios en Pyxis los llevan a cabo dos arquitectos, Carlos y Pablo. Ellos diseñan las arquitecturas según las necesidades y los recursos que dispone cada cliente.

Carlos es el encargado de un proyecto que consiste en brindar una capa de API Rest a través de las cuales, los distintos clientes pueden acceder a diferentes servicios pertenecientes a una empresa de forma más sencilla. Este proyecto puede verse como un Api Gateway que se encarga de recibir solicitudes y enviarlas a donde corresponda. Los servicios de la empresa ofrecen información de forma muy distribuida y poco amigable al usuario, por este motivo, este proyecto se encarga de exponer servicios que tras una única consulta devuelve resultados para los cuales habría que consultar varios servicios de la empresa. Además de esto, este proyecto se encarga de realizar una transformación de la información proveída por la empresa para que se le proporcione una respuesta más amigable a la aplicación cliente.

Pablo es el encargado de varios proyectos, la mayoría son para la misma empresa del proyecto de Carlos, estos brindan soluciones de streaming de video, audio y procesamiento de datos. Algunos son de gestión de negocio, los cuales deben tener la capacidad de despliegue rápido y parcial que hagan frente a los cambios de negocio.

En la sección 4.2 y 4.3 se brindan más detalles sobre cada una de las arquitecturas.

4.2. Proceso de trabajo

Para realizar el relevamiento se estableció un ciclo de trabajo, que está compuesto por 4 etapas: Investigación, Análisis, Generación de preguntas y Obtención de respuestas.

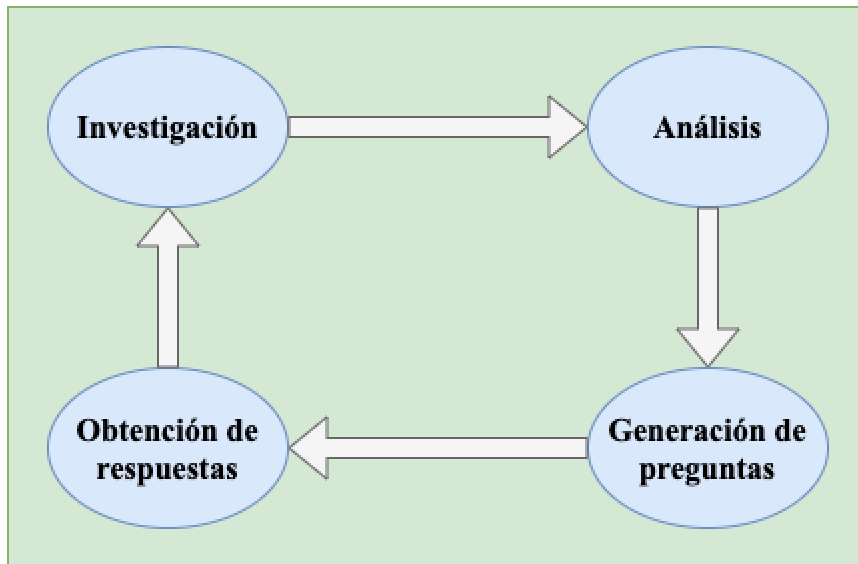


Figura 8. Ciclo de trabajo del relevamiento.

En la figura 8 se muestra un diagrama del ciclo de trabajo del relevamiento y a continuación se describen las 4 etapas.

Investigación

Es la etapa inicial del ciclo, en la misma se realiza la investigación correspondiente para obtener información sobre los distintos temas sobre microservicios. En el primer ciclo se realizó una investigación general y en los siguientes ciclos se realizó una investigación más específica a los temas que competen a los proyectos de cada arquitecto.

Análisis

Esta etapa consistió en el análisis de la información obtenida en la etapa de Investigación y de la que se tenía del relevamiento hasta el momento. El análisis se realizó manteniendo reuniones internas, en las cuales analizamos toda la información para establecer los distintos temas en que se tenía dudas o falta de información.

Generación de preguntas

Esta etapa consistió en la construcción de un conjunto de preguntas a realizar a los arquitectos sobre los temas que se establecieron en la etapa Análisis. Las preguntas se realizaron en un documento en el cual íbamos agregando preguntas de los distintos temas, para luego mantener una reunión y discutir todo el conjunto de preguntas.

Obtención de respuestas

Esta etapa consistió en la recolección de información de los proyectos de Pyxis. Una vez que se contaba con el conjunto de preguntas generado en la etapa Generación de preguntas, se mantenía una instancia en la cual los arquitectos nos daban una respuesta. Se tuvo dos tipos de instancia, una consistía en el envío de las preguntas por correo electrónico a los arquitectos y la otra en una reunión por video llamada con cada uno de los arquitectos.

Las preguntas enviadas por correo electrónico se podían responder en pocas líneas y se utilizó para hacer más ágil el relevamiento.

En las reuniones por video llamada se realizaban preguntas en las que se precisaba más información y según las respuestas podían llevar a realizar otras preguntas. Además en estas reuniones se pudo obtener un gran volumen de información ya que en el momento se iban generando nuevas preguntas.

Luego de obtenida toda la información se evaluaba si se realizaba una nueva iteración del ciclo de trabajo o finalizaba el ciclo.

El relevamiento se llevó a cabo durante cuatro ciclos de trabajo, el primer ciclo duró 4 semanas y el resto 2 semanas cada uno. El primer ciclo llevo un tiempo mayor ya que se realizó una investigación que abarcaban muchos temas generales de microservicios.

Una vez concluido el relevamiento se pasó a analizar las posibles mejoras a sugerir. Estas mejoras llevaron un proceso de investigación la cual ayudó a evaluar las distintas alternativas.

Por último se realizó una presentación con todas las posibles mejoras sugeridas, la misma fue realizada al equipo de Pyxis el cual estaba constituido por los arquitectos, encargados y tutor del proyecto. Finalizada la presentación se realizó una devolución de las sugerencias.

4.3. Arquitecturas Relevadas

A continuación se presentan las dos arquitecturas relevadas durante el proyecto, para las mismas se presenta una descripción con diagramas y una explicación de los componentes más relevantes.

4.3.1. Arquitectura Gateway

Esta arquitectura utiliza Kubernetes [26] como orquestador de contenedores, facilitando la administración y despliegue de los microservicios. Kubernetes tiene implementados patrones que se encarga de llevar un registro de los servicios disponibles, balanceo de carga y poder establecer la configuración externa que precisen los microservicios.

Tras el relevamiento de esta arquitectura se identificaron tres elementos principales: Backend, Plataforma de Integración, y Logging y Monitorización. Estos se presentan en la figura 9.

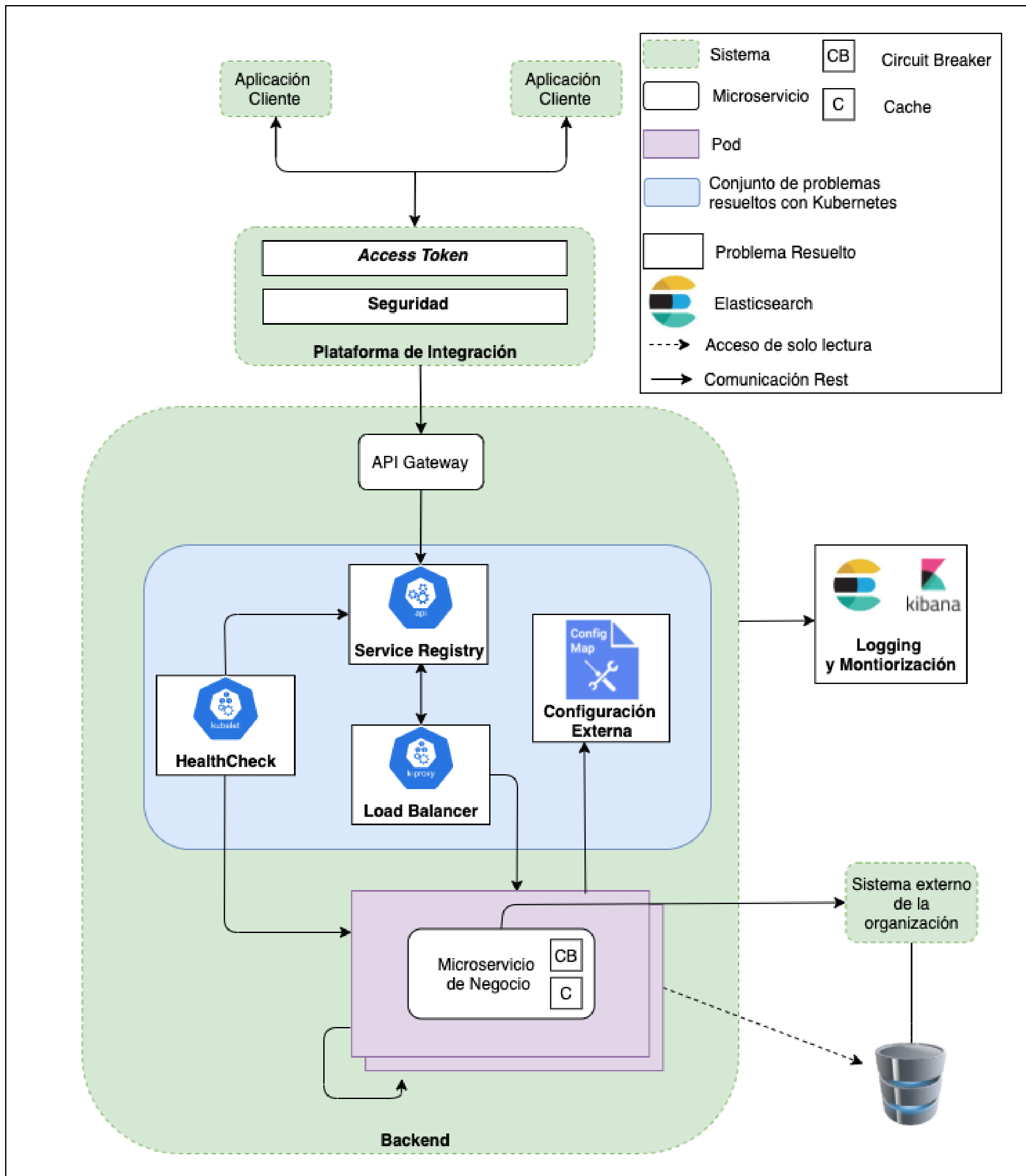


Figura 9. Diagrama de arquitectura gateway.

Backend

El backend está formado por Api Gateway, un conjunto de pods que cada uno de ellos contienen un microservicio y un conjunto de problemas resueltos por Kubernetes.

A continuación se describen los componentes que se encuentran en la figura 9.

- Api Gateway: Este componente proporciona una interfaz API Rest por la cual se puede acceder a los servicios que brinda la solución. Las solicitudes, que llegan al

componente, son enviadas al Service Registry de Kubernetes para que sea asignada a una instancia de microservicio disponible que pueda realizar la misma.

- Service Registry: Este componente lo implementa Kubernetes y se encarga de mantener los registros de los servicios como se especifica en el patrón Service Registry. La función de este componente es tener las direcciones de todas las instancias de los servicios para que cuando el Api Gateway reciba una consulta para cierto servicio, le pueda consultar al Service Registry hacia dónde debe dirigirla.
- HealthCheck: Este componente está implementado por Kubernetes y se encarga de brindar un mecanismo para saber si un contenedor contenido en un pod se encuentra disponible para recibir carga de trabajo. En caso que el contenedor no se encuentre disponible, se considera que el pod no está disponible. También cumple la funcionalidad de comunicarle al Service Registry cuando un pod deja de estar disponible, de esta forma el service registry puede eliminarlo de la lista y mantenerse actualizado [27].
- Configuración externa: Este componente está implementado por Kubernetes y se encarga de concentrar todos los datos de configuración que precisan los pods, que pueden ser variables de entorno, líneas de comando o ficheros de configuración. El mismo, está constituido por un conjunto de ConfigMaps de Kubernetes, estos son objetos que permiten almacenar datos de configuración que precise cada pod en formato clave-valor.
- Load Balancer: Este componente está implementado por Kubernetes y se encarga de balancear la carga entre las distintas instancias de microservicios, logrando distribuir la carga de trabajo entre ellos.
- Microservicio de negocio: Estos son los encargados de realizar las funcionalidades que brinda el sistema. La principal funcionalidad que tienen los microservicios de este proyecto es realizar una o varias peticiones a distintos sistemas externos pertenecientes a la empresa a través de distintos servicios que esta expone, en su mayoría SOAP, con respuesta asincrónica y sincrónica. Una vez que reciben las respuestas, realizan composiciones y transformaciones de las mismas, logrando simplicidad y un modelo natural a la web de la empresa o los distintos sistemas que realizan peticiones a la capa de APIs. Dada la funcionalidades que brindan los microservicios, estos no tienen necesidad de contar con una base de datos. Solo hay un microservicio que realiza consultas a una base de datos de la empresa, externa de solo lectura. Cada microservicio cuenta con un circuit breaker y un caché que puede ser compartido o en memoria. Estos microservicios se comunican entre ellos vía Rest.

Plataforma de integración

Es un componente que pertenece a la empresa, el cual se encarga de recibir las peticiones para los sistemas de la empresa. Luego de recibida una petición realiza la autenticación, autorización y redirige la solicitud al sistema que corresponda.

Logging y Monitorización

Este componente se encarga de la monitorización del sistema y de la gestión de logs. Cuenta con una base de datos de Elasticsearch [28], la cual gestiona la empresa.

Los logs son generados por los pod y guardan la información en un formato estandarizado, en archivos del sistema que contienen Kubernetes, luego los encargados de gestionar Elasticsearch obtienen estos datos de estos archivos para guardar la información indexada.

Para visualizar los logs almacenados se utiliza Elastic APM [29], el cual es un herramienta de Kibana [30] que se utiliza para realizar el monitoreo de los microservicios. En este caso la empresa tiene un usuario con acceso de lectura para monitorear los microservicios.

Patrones detectados

A continuación en la figura 10, se presenta un diagrama con los patrones detectados en la arquitectura, para esto se utilizó como referencia el diagrama de patrones arquitectónicos seccionado que presenta Chris Richardson en su página web sobre microservicios [31].

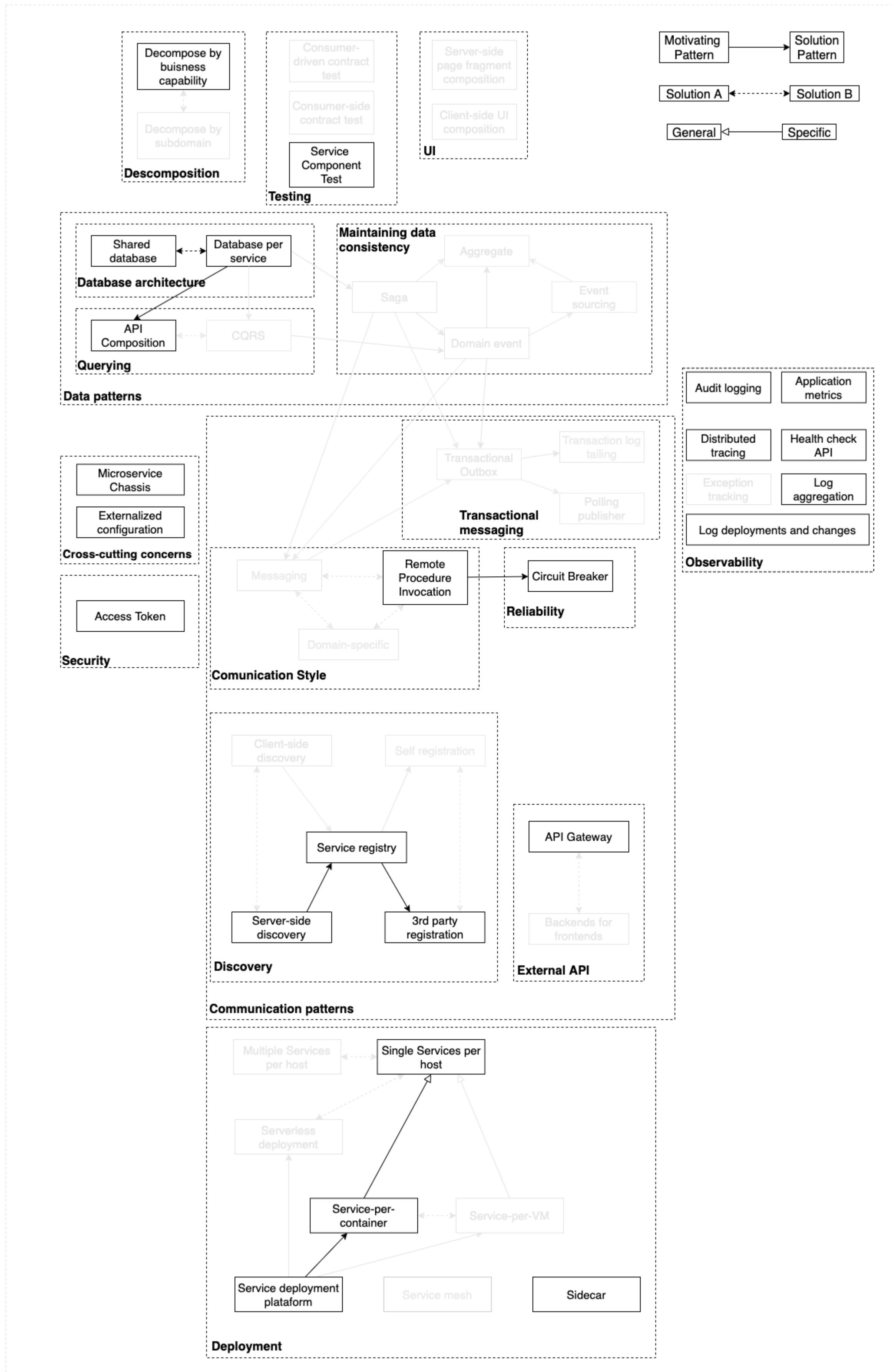


Figura 10. Diagrama de patrones arquitectónicos de la arquitectura gateway.

4.3.2. Arquitectura de Streaming

Para este caso, la arquitectura relevada intenta representar un amplio conjunto de proyectos similares. Este relevamiento incluye los elementos que todas estas arquitecturas (o la gran mayoría) tienen en común, buscando presentar una vista genérica a todos estos proyectos, que pueda ayudar a visualizar una base para la construcción de algún proyecto futuro. Por este motivo se dejan de lado elementos que sean específicos a un proyecto en particular y no aporten valor al momento de iniciar un nuevo proyecto.

Tras el relevamiento de estas arquitecturas se identificaron tres elementos principales: Core, Identity Provider y lo que nosotros llamamos Backend. Estos se pueden observar a continuación en la Figura 11.

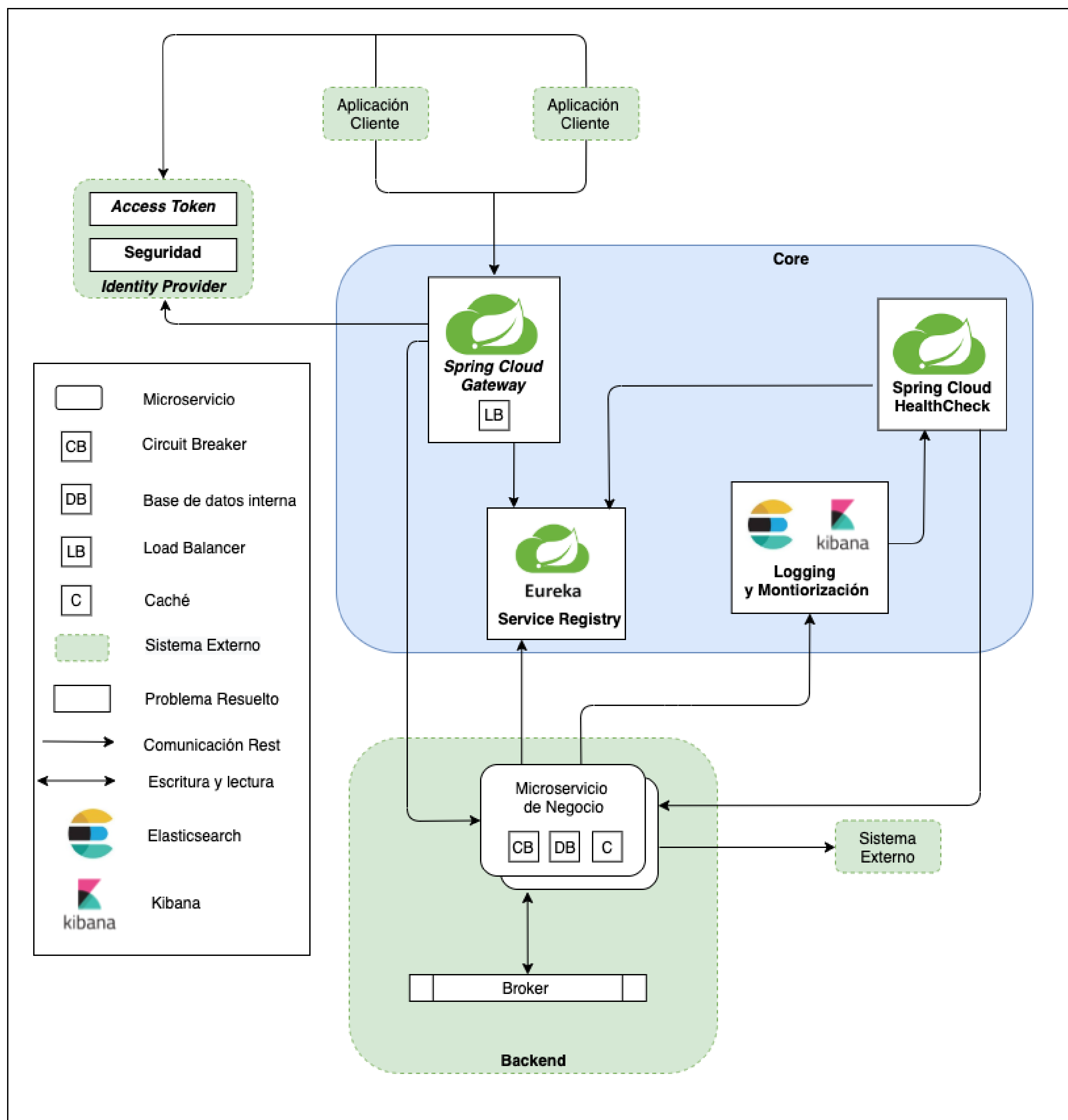


Figura 11. Diagrama de arquitectura de Streaming.

Identity Provider

Este elemento es externo a la aplicación y se encarga de la seguridad y la autenticación de los usuarios dentro de la arquitectura. Cuando una aplicación cliente quiere acceder al sistema debe comunicarse con el identity provider de preferencia para obtener un token de acceso (access token) que podrá utilizar para probar su identidad en el sistema.

Para lograr una comunicación exitosa con el sistema, la aplicación cliente debe enviar este token al Api Gateway, el cual se encargará de comunicarse con el identity provider que corresponda para verificar la validez del token enviado y obtener la información del usuario.

La principal capa de control de seguridad se encuentra a nivel de Api Gateway, los elementos de la arquitectura que se encuentran por detrás de este punto se encuentran en un ecosistema que se considera seguro, y que por tanto no necesita realizar grandes controles de seguridad.

Core

El core está compuesto por librerías y un conjunto de microservicios que se despliegan y se utilizan en la mayoría de los proyectos de la empresa, y que se consideran necesarios en la gran parte de los casos. Resuelve las tareas básicas que se encuentran en casi todos los proyectos como descubrimiento de servicios, Api gateway, monitorización, análisis de logs, trazabilidad, manejo de métricas y detección de errores. Fue construido a partir de los primeros proyectos de microservicios de la empresa, y se ha ido refinando tras la aparición de nuevos patrones, estrategias de diseño o tecnologías de microservicios para mejorarlo. Dado que este elemento ha ido evolucionando con el tiempo, pueden existir discrepancias entre lo que ofrece el core de un proyecto y lo que ofrece el core de otro, esto no se verá reflejado en el diagrama o en esta descripción.

Dentro del Core se detectaron cuatro componentes importantes (véase la figura 11):

- Api Gateway: Este componente se encarga de recibir las solicitudes, es el punto de entrada de la arquitectura. Cuando un cliente se quiere comunicar con la aplicación, el Api Gateway se comunicará con el identity provider que corresponda para verificar la validez del access token del cliente (Esto tiene como ventaja que la aplicación no está obligada a utilizar un identity provider específico). Este componente se encarga además, de realizar un balanceo de carga entre los microservicios.

- Service Registry: Este componente está implementado con Eureka [32] y se encarga de mantener los registros de los servicios como se especifica en el patrón Service Registry. Para registrar los servicios se sigue el patrón Self-Registration, es decir que cada microservicio tiene la responsabilidad de registrarse en el Service Registry una vez que pasa a estar activo. La función del componente Service Registry es tener las direcciones de todas las instancias de los servicios para que cuando el Api Gateway reciba una consulta para cierto servicio, le pueda consultar al Service Registry hacia dónde debe dirigirla.

- HealthCheck: Este componente está implementado con Spring Cloud [33] y se encarga de monitorizar la disponibilidad de los servicios y comunicarle al Service Registry cuando una instancia de servicio deja de estar disponible, de esta forma el Service Registry puede eliminarlo de la lista y mantenerse actualizado.
- Logging y Monitorización: Como su nombre lo indica, este componente se encarga de la monitorización del sistema y de la gestión de logs. Cuenta con una base de datos de Elasticsearch [28] en la que se guardan los logs del sistema para su posterior monitorización. Para la monitorización se utiliza Kibana aunque también puede haber algún sistema externo que se comunique con la aplicación para realizar monitorización.

Backend

Este elemento al que nosotros llamamos backend contiene todos los microservicios de negocio de la arquitectura y las colas de mensajes necesarias para que estos puedan comunicarse correctamente.

Los microservicios de negocio son los encargados de realizar las funcionalidades que ofrece el sistema, cada uno de ellos tendrá su propio Circuit Breaker, base de datos y caché local. Se comunicarán entre sí a través de colas de mensajes respetando el patrón Saga para la consistencia de datos.

Cada microservicio se encargará de registrarse en el Service Registry una vez que esté disponible y se comunicará con el componente de Logging y Monitorización para la generación de datos en Elasticsearch. También deberá ofrecer un endpoint que le permita al componente de Health Check consultar su disponibilidad.

El bus de eventos es un sistema de mensajería que está implementado con Kafka [34], se utiliza para toda interacción que deban realizar los microservicios en las que no precise una respuesta sincrónica. Este escenario suele darse en la mayoría de los casos ya que al comunicarse tantos microservicios de forma secuencial, los tiempos de respuestas serían muy altos si se hicieran de forma sincrónica.

A diferencia de la otra arquitectura relevada, esta no utiliza contenedores. En lugar de tener un microservicio por pod como en el caso anterior, se tiene varios microservicios por máquina virtual.

Patrones detectados

A continuación en la figura 12, se presenta un diagrama con los patrones detectados en la arquitectura, para esto se utilizó como referencia el diagrama de patrones arquitectónicos seccionado que presenta Chris Richardson en su página web sobre microservicios [31].

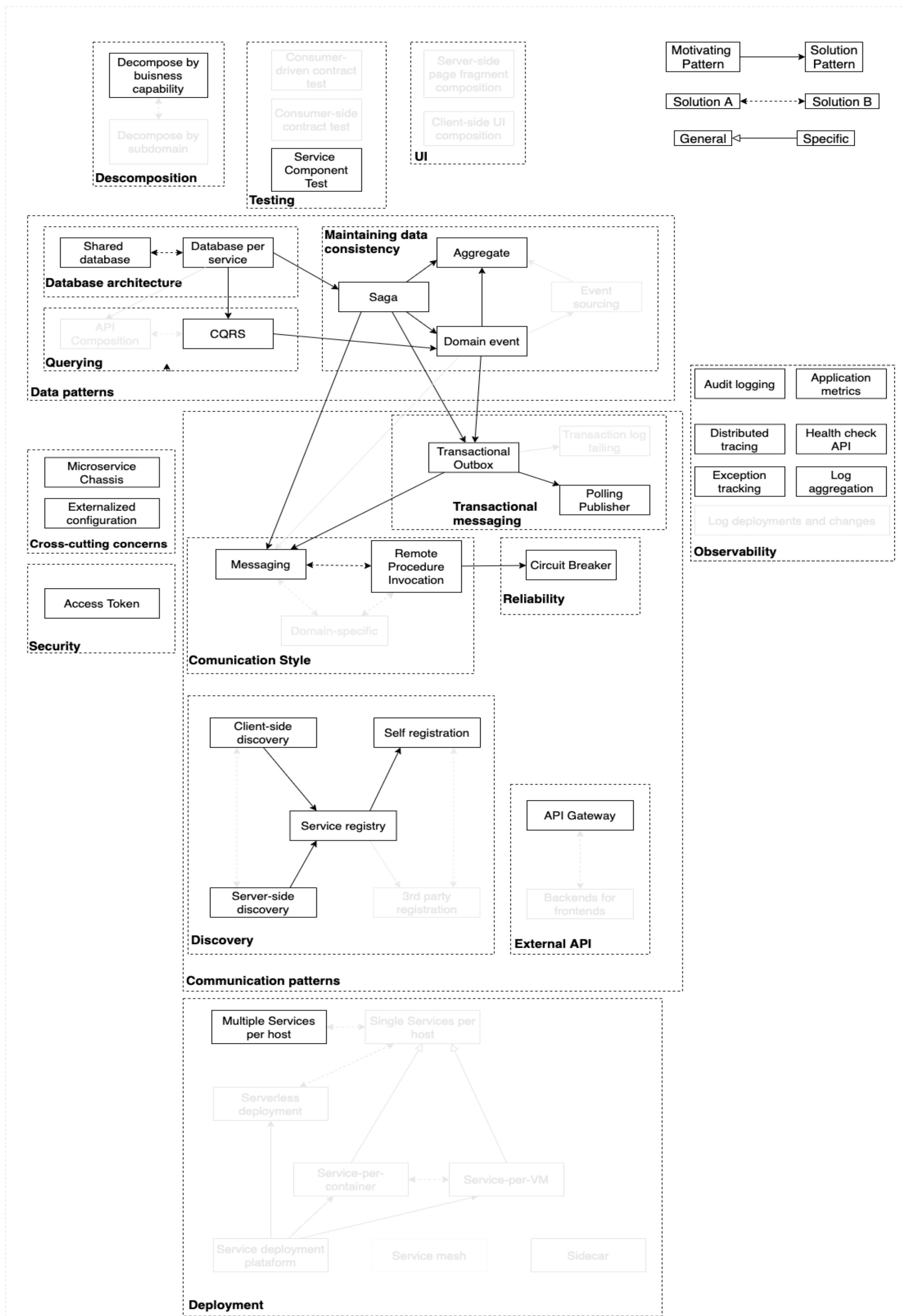


Figura 12. Diagrama de patrones arquitectónicos de la arquitectura de Streaming.

4.4. Propuestas de mejora

A continuación se presentan las propuestas de mejora que se le presentaron al cliente tras el análisis de los resultados obtenidos en el relevamiento.

Luego del análisis de los resultados consideramos que las arquitecturas relevadas resuelven de buena manera los diferentes problemas que se presentan para cada una de sus realidades. Además algunas mejoras que se podrían hacer, ya fueron consideradas por el cliente y descartadas por motivos fuera de su control. Estos elementos dificultaron encontrar propuestas que realmente aporten valor a las soluciones.

Durante las distintas reuniones del proyecto se notó una preferencia del cliente hacia mejorar sus herramientas de testing, por lo que la mayoría de estas propuestas fueron orientadas a testing.

4.4.1. Chaos Monkey For Spring Boot

Durante nuestras reuniones con el cliente surgió el tema de Ingeniería del Caos, tras una conversación al respecto, consideramos que el cliente se sentía intrigado por realizar pruebas de este estilo, sin embargo, le parecía algo muy riesgoso.

Tras investigar las posibilidades disponibles para este tema, encontramos esta biblioteca para Spring Boot, la cual fue inspirada en los principios de ingeniería del caos. Permite realizar distintos tipos de ataques caóticos en la aplicación:

- Latency Assault: Ataca al servicio generando altas latencias de manera al azar dentro de rangos especificados.
- Exception Assault: Genera excepciones aleatorias en tiempo de ejecución.
- KillApp Assault: Detiene la ejecución de la aplicación.
- Memory Assault: Consume la memoria de la máquina virtual de Java.

Una de las ventajas que tiene esta biblioteca es que se puede implementar de forma que permita consultar y modificar su configuración de forma dinámica mediante consultas http. Esto permite tener cierto control dentro del caos y podría por tanto, ser una buena opción para introducir un sistema en el mundo de la ingeniería del caos.

La configuración dinámica de la biblioteca permite utilizarlo de distintas maneras, por ejemplo, se podría ir activando y desactivando la biblioteca en distintos servicios para sus distintos tipos de ataque y observar cómo reacciona cada servicio para cada tipo de ataque y cómo esto impacta en la experiencia de usuario.

4.4.2. Contract Testing

Durante las reuniones con el cliente se detectó interés en mejorar los tests de integración del sistema, ya que realizar este tipo de testing resultaba costoso y presentaba algunos problemas

indeseados. Los tests de integración presentan no solo un alto costo de escritura sino de mantenimiento, hay que mantener los tests actualizados ante cada cambio en los microservicios.

Tras investigar más sobre los distintos tipos de testing utilizados en el área de microservicios, consideramos que Contract Testing sería una buena opción para integrar en sus sistemas, ya que presenta un costo de escritura y mantenimiento mucho menor. Además, Contract Testing detecta errores en etapas más tempranas que los tests de integración típicos.

Para la presentación de esta propuesta se investigaron las herramientas Pact [35] y Spring Cloud Contract [36].

4.4.3. Testing de Performance

En las distintas instancias que tuvimos con el cliente pudimos ver que realizaban una monitorización de los servicios, logrando detectar si su funcionamiento era correcto, pero no realizaban un test de performance antes de desplegar los servicios. Por esto nos pareció conveniente sugerir distintas estrategias de performance a realizar a los microservicios para poder validar que cumplía con los requerimientos no funcionales de rendimiento y lograr detectar problemas de performance.

Se sugirieron realizar las siguientes pruebas de performance:

- Pruebas de Carga: Esta prueba consiste en realizar distintas cargas de trabajo al sistema hasta llegar a la carga que debe soportar el sistema. Con este tipo de pruebas se puede detectar si el sistema soporta la carga esperada de peticiones. En las pruebas que se realicen se muestran los tiempos de respuestas los cuales se pueden utilizar si cumplen con el requerimiento no funcional. También se puede monitorizar otros aspectos para poder detectar si existen cuellos de botella.

- Pruebas de estabilidad: Esta prueba consiste en mantener la carga de trabajo esperada del sistema durante un tiempo prolongado. Logrando determinar si el sistema soporta una carga esperada continua. En este tipo de pruebas se realiza para detectar si hay alguna fuga de memoria.

- Pruebas de Pico: Esta prueba consiste en realizar cargas con distintas cantidades de usuarios del sistema. Se utiliza para observar el sistema y analizar cómo se comporta cuando tiene distintas cantidades de usuarios, tanto cuando tiene una mínima cantidad de usuarios o se llega a la cantidad de usuarios esperados.

- Pruebas de Estrés: Esta prueba consiste en realizar una carga más grande que la esperada para detectar cual es el límite de carga del sistema y analizar cómo reacciona el sistema a este tipo de sobrecarga. Se utiliza para establecer la solidez de la aplicación.

Luego de presentadas la sugerencia, nos comentaron que Pyxis no realiza este tipo de prueba porque las realiza el cliente en su ambiente de testing. Por lo que este tipo de pruebas estaban a cargo del cliente.

4.4.4. Transactional Outbox

Como se vio en la sección 2.4.7 (Transactional Outbox), Transactional Log Tailing es una forma de implementar el patrón Transactional Outbox, el cual surge como necesidad para solucionar un problema que puede ocurrir al utilizar el patrón Saga.

Durante la etapa de relevamiento se realizaron preguntas para determinar si se estaba utilizando este patrón, en base a las respuestas dadas por el cliente se llegó a la conclusión de que no se estaba utilizando a pesar de que sí se utilizaba el patrón Saga.

Debido a esto decidimos incluirlo entre las propuestas de mejora, ya que soluciona un error que se podría dar en el sistema actual en cualquier momento. Nuestra idea era recomendar la implementación Transactional Log Tailing, pero aún así decidimos presentar ambas implementaciones para que el cliente tenga toda la información y decida por su cuenta. Sin embargo, al presentarle el patrón al cliente y explicarle las dos implementaciones posibles, nos dijo que ya estaba utilizando el patrón Polling Publisher, aunque no sabía que se trataba de un patrón arquitectónico con nombre conocido.

Tras esta nueva información se comunicó al cliente que aún así dejaríamos como propuesta el cambio a Transactional Log Tailing, ya que su sistema suele crecer a medida que pasa el tiempo y Polling Publisher podría generar problemas de performance en un futuro.

5. Architecture Adviser

En este capítulo se presenta la solución propuesta para la problemática de la etapa final del proyecto.

Está dividido en 5 secciones, en la primera sección se da una descripción general del funcionamiento de Architecture Adviser. La segunda sección describe el proceso de trabajo que se siguió durante esta etapa, dando un poco de contexto sobre cómo se llegó a la solución final. La tercera sección habla de las distintas decisiones de diseño tomadas durante la búsqueda de la solución, así como de los artículos de trabajo relacionados y cómo estos sirvieron de inspiración para tomar alguna de estas decisiones. La cuarta sección presenta el modelo final de la solución explicando sus distintos componentes y su funcionamiento. Por último, la quinta sección presenta la arquitectura de la solución utilizando el modelo de vistas 4 + 1 de Philippe Kruchten [37].

5.1. Descripción General

Architecture Adviser es una aplicación web responsive, cuyo objetivo es asesorar a los usuarios al momento de construir una arquitectura de software basada en microservicios.

La solución se basa en la idea de que existe uno o varios encargados de la aplicación, quienes tendrán la tarea de definir el conjunto de preguntas, respuestas y componentes del sistema, así como las relaciones que existen entre ellos. Estos elementos deben ser cargados en el sistema previo a su utilización. Debido a lo acordado con el cliente, las funcionalidades de la aplicación para realizar esta tarea quedaron fuera del alcance, por lo que esta carga inicial deberá ser a través de un script sql.

Una vez realizada esta tarea el sistema se encuentra listo para ser utilizado por los usuarios finales, estos podrán ver los proyectos existentes en el sistema y crear nuevos proyectos para realizar el proceso de asesoramiento.

Luego de crear un proyecto, el usuario puede acceder y proceder a responder las preguntas que definieron los encargados del sistema. Luego de responder estas preguntas, el sistema valida las respuestas ingresadas por el usuario contra las decisiones definidas por los encargados, y ofrece como resultado final un conjunto de componentes arquitectónicos, patrones y tecnologías para describir la arquitectura recomendada.

Tras responder todas las preguntas, el usuario puede acceder a una grilla donde se muestran estos elementos, donde se indica además si el componente se encuentra como recomendación o si hay componentes que generen incompatibilidad entre ellos.

Por último el usuario puede desmarcar (y volver a marcar) los componentes en caso que considere que hay alguno que no se debe o no se puede utilizar. También puede acceder a cada uno de los componentes para obtener información sobre ellos. Tras acceder a un componente se presenta una nueva página con una descripción general del mismo (escrita por

los encargados) y una lista de fuentes de información que puedan ayudar con la comprensión o implementación del componente.

5.2. Proceso de trabajo

Para la determinación del modelo se realizó una investigación que debido a los tiempos planificados para el proyecto resultó ser un poco más acotada de lo deseado. Nos encontrábamos frente a un cambio de rumbo del proyecto que implicaba la repetición de varias etapas como análisis de requerimientos e investigación, y dado que no deseábamos que el proyecto se extendiera demasiado, al momento de volver a planificar tomamos la decisión de acotar el tiempo dedicado para algunas etapas, entre ellas la etapa de investigación y construcción del modelo.

Debido al tiempo acotado para esta etapa, la búsqueda de un modelo adecuado conformó un reto importante. Tras comenzar nos percatamos de que el modelo que buscábamos no era algo tan común como esperábamos. La mayoría de los modelos encontrados se orientaban más a la evaluación de resultados, es decir que tomaban decisiones en base a información relevante sobre rendimiento. Un ejemplo de esto podría ser que el modelo analice ciertos aspectos de una arquitectura ya construida, como los tiempos de respuestas y en base a ese análisis retorne un valor que de alguna forma indicaría la correctitud de la arquitectura en este aspecto.

El modelo que nosotros buscábamos es diferente debido a que la arquitectura aún no está construida y no se tienen datos que informen sobre el rendimiento final de la misma, y aunque la mayoría de los resultados encontrados están orientados a otros objetivos como la evaluación, se encontraron dos artículos dentro de los lineamientos deseados.

Tras la etapa de investigación de posibles modelos a utilizar, se realizó un análisis de los modelos encontrados para identificar cuál se adaptaría mejor a nuestra realidad. En dicho análisis se encontraron características particulares de cada uno de ellos que resultan interesantes, y similitudes que marcan cierta alineación en algunos caminos a seguir.

Como resultado concluimos que el modelo presentado por el método Quark resultaba muy interesante y bastante alineado con la problemática que debíamos resolver, por lo que decidimos utilizarlo como punto de partida y realizar un análisis más detallado. Para realizar este análisis se dedicaron varias semanas, en las cuales se analizó el comportamiento del modelo en los distintos requerimientos del sistema, verificando que todos sean soportados y analizando para cada requerimiento los posibles casos bordes que podrían generar problemas o complicaciones. A medida que se avanzó con este proceso se fueron encontrando casos de uso no soportados que generaron que el modelo se fuera modificando en un proceso de refinamiento. Durante este proceso también se incluyeron algunas reuniones con el cliente, para que pueda observar el modelo que se estaba construyendo y comentar cualquier sugerencia o inquietud que considerara relevante.

Tras algunas semanas de refinamiento e instancias con el cliente donde nuevos requerimientos siguieron surgiendo, el modelo fue evolucionando para adaptarse más a

nuestras necesidades y a los tiempos planificados para el proyecto. A pesar de que se utilizó el modelo de Quark y algunas de sus ideas como base para la construcción del nuestro, éste terminó convirtiéndose en algo muy diferente. Si bien pensamos que el método Quark es muy completo e interesante, ya sea desde nuestro punto de vista o desde el punto de vista del cliente, algunos de sus aspectos resultaron no ser la mejor opción para este proyecto.

5.3. Decisiones de Diseño y Trabajo Relacionado

A continuación se presentan las decisiones de diseño tomadas durante el transcurso del proyecto junto a un análisis del trabajo relacionado y cómo este impacto en dichas decisiones.

5.3.1. La toma de decisiones debe estar en la capa de datos

El sistema que se debe construir presenta ciertos aspectos que deben ser considerados desde el comienzo de la búsqueda de una solución, siendo el más importante, el requerimiento *"RA20 - Modelo extensible"*. Este requerimiento indica que en un futuro debe ser posible modificar y/o manejar no solo las preguntas con sus respuestas, y los componentes resultantes, sino también cómo todos estos se relacionan. Como consecuencia, todos estos elementos mencionados, deben pertenecer al sistema en forma de datos que eventualmente pueden ser modificados de tal forma que el sistema continúe funcionando correctamente.

Tras realizar este pequeño análisis, surge una de las primeras deducciones a las que se llegó sobre el diseño de la solución, la lógica de toma de decisiones debe existir en la capa de datos. Esto significa que la lógica que el sistema utilice para decidir qué componentes se deberían utilizar en base a ciertas respuestas ingresadas por el usuario, no puede ser directamente escrito en código, sino que el código debe funcionar con cualquier lógica de decisión válida que el encargado del sistema ingrese en los datos.

5.3.2. Usar el modelo de Quark como punto de partida

Como se explicó en la sección de marco teórico, el modelo presentado en la figura 6 dentro de la sección 2.5.3 carece de información suficiente para la completa comprensión de su funcionamiento. Tomando en cuenta esto, los requerimientos *"RA19 Facilidad de uso"* y "*RA20 Modelo Extensible"*, decidimos no utilizar este modelo, aunque se analizó la posibilidad de utilizar algunos de los componentes que presenta, permitiéndonos relacionarlo un poco con la realidad a la que nos enfrentamos y comenzar a formar algunas conclusiones. Por los elementos ya mencionados concluimos que de utilizar un modelo como este sería bastante complejo añadir un componente al sistema, ya que esto implicaría realizar una investigación y estudio de gran tamaño sobre cómo se relaciona un patrón con los distintos atributos de calidad.

Debido a que los objetivos del artículo explicado en la sección 2.5.3 no están orientados directamente al modelo que presentan, no se proporciona una explicación adecuada del mismo, de sus componentes, sus relaciones o de lo que estas representan. Si bien el diagrama se puede analizar de todos modos (tarea que se realizó), resulta ser un diagrama complejo y

con muchos componentes, para varios de los cuales se requiere una explicación de por qué están allí y cómo impactan en la toma de decisiones. Por otro lado, tras analizar mejor el modelo Quark, notamos que se alineaba mucho con la problemática que debíamos resolver y tenía información suficiente sobre su funcionamiento. Fue por esto que decidimos utilizar el modelo presentado por Quark como punto de partida para la búsqueda de nuestra solución.

5.3.3. Quitar los atributos de calidad del modelo

Los modelos analizados presentaban algunas similitudes, una de ellas es la aparición de atributos de calidad para representar de forma imparcial y fácil de entender, los beneficios que otorga un componente o patrón. Durante las primeras etapas de refinamiento, nuestro modelo incluía atributos de calidad ya que pretendíamos seguir estos lineamientos. Pero luego de una reunión con el cliente, se determinó que este elemento podría generar ciertas dificultades y que sería mejor comenzar con un modelo más simple. Debido a esto se decidió quitar los atributos de calidad y pasar a la idea de tener una *"relación directa"* entre preguntas y componentes. De esta manera, el arquitecto a cargo puede determinar cómo las preguntas resuelven la utilización de componentes, sin la necesidad de realizar el proceso de identificar específicamente cuales son aquellos atributos de calidad involucrados. Esto resulta estar alineado con los resultados de las investigaciones realizadas por Quark sobre el deseo de los arquitectos de tener el control total del sistema, ya que al quitar los atributos de calidad se ofrece más control al arquitecto.

5.3.4. Comparación de realidad entre este proyecto y Quark

Una de las diferencias entre el método Quark y la realidad a la que nos enfrentamos en este proyecto es que Quark parte de una lista de decisiones arquitectónicas, de las cuales el arquitecto debe seleccionar las que desea utilizar, mientras que nuestro sistema parte de preguntas múltiple opción que se deben responder. Son enfoques diferentes que tal vez no tengan gran relevancia en muchos de los casos, pero consideramos que el método Quark puede terminar enfrentándose al problema de tener demasiadas decisiones arquitectónicas, generando que muchas de estas pasen desapercibidas para el arquitecto. El beneficio de utilizar preguntas es que todas deben ser respondidas para continuar, por lo que una pregunta nunca pasará desapercibida por el usuario. Sin embargo, utilizar preguntas también puede traer sus dificultades ya que uno de nuestros requerimientos es que se puedan incluir preguntas que dependan de las respuestas de otras preguntas. Por ejemplo, el arquitecto a cargo del sistema tal vez quiera integrar varias preguntas sobre comunicación con sistemas externos, pero esto solo tiene sentido si dicha comunicación existe. En este caso se podría incluir la pregunta inicial *¿Existe comunicación con sistemas externos?* Si el usuario responde *"No"* a esta pregunta, todas las otras preguntas sobre este tema dejan de tener sentido, por lo que no se le deberían presentar al usuario. Este es un escenario sencillo en el que las preguntas dependen de la respuesta a una única pregunta, pero se pueden presentar otros que resultan un poco más complejos en los que una pregunta depende de la respuestas de varias preguntas.

5.3.5. Incluir elemento Decision

Los casos descritos en la sección anterior llevaron a un análisis importante para determinar cómo serían representados en el modelo. En nuestra búsqueda por un modelo sencillo, nos encontramos frente a dos problemas similares pero con algunas diferencias importantes en cuanto a complejidad, las precondiciones entre preguntas y la determinación de acciones sobre componentes. Dadas las similitudes presentadas por estos dos problemas, se buscó una solución que permitiera resolverlos ambos de la misma forma y lograr así evitar generar complejidad extra en el modelo a utilizar. Fue así como se decidió incluir el elemento “Decisión” en el modelo.

5.4. Descripción del modelo

A continuación en la figura 13, se presenta el modelo construido, junto con una descripción de sus elementos más importantes y cómo se relacionan.

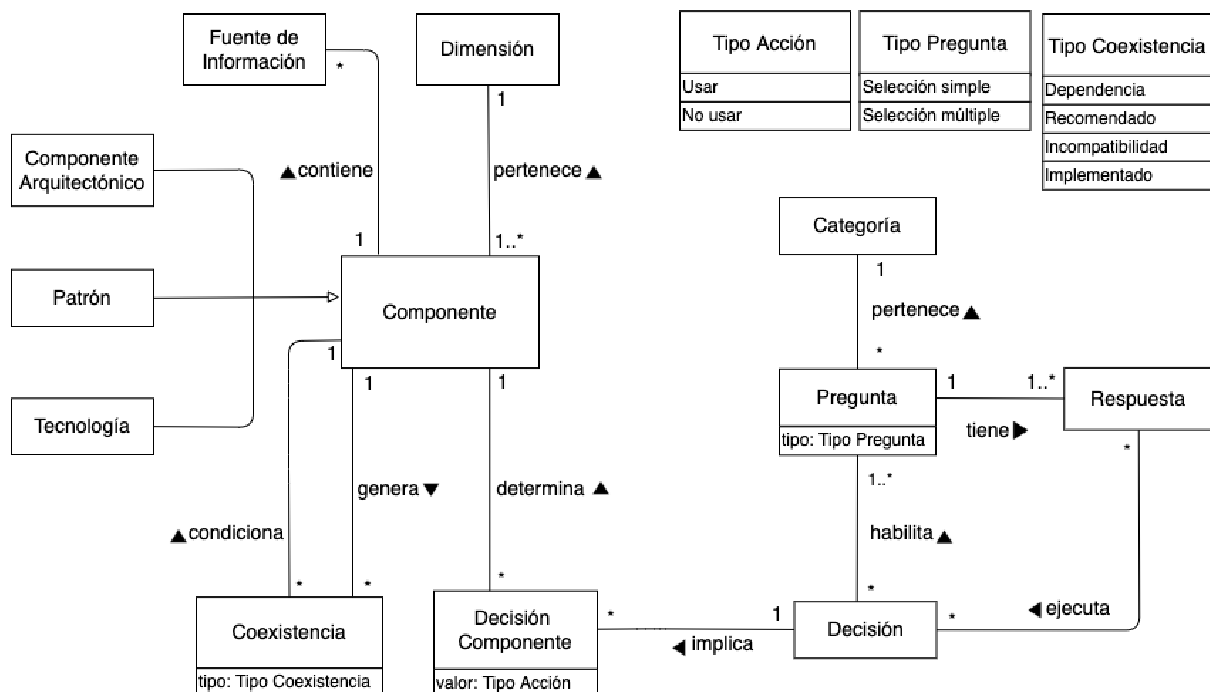


Figura 13. Modelo UML2.0 para toma de decisiones utilizado por Architecture Adviser.

Comenzaremos dando una breve explicación de los elementos del modelo que se enfocan más en la presentación de datos al usuario, estos son: Categoría, Pregunta, Respuesta, Dimensión, Fuente de información y Componente.

Las categorías sirven para agrupar preguntas y presentarselas al usuario de una forma más ordenada, cada pregunta debe pertenecer a una única categoría para ser presentada al usuario. A su vez, las preguntas tienen un conjunto de respuestas para que el usuario pueda responder, las respuestas solo pueden pertenecer a una única pregunta y serán presentadas debajo de esta, dentro de la categoría de dicha pregunta. Las preguntas pueden ser de dos tipos:

- *Selección simple*: Consta de respuestas excluyentes y sólo se puede seleccionar una respuesta al responder.

- *Selección múltiple*: Consta de respuestas que no son excluyentes y al responder se puede seleccionar una o varias de ellas.

Por otro lado tenemos las dimensiones, las cuales sirven para diferenciar los distintos aspectos de una arquitectura y agrupar componentes al momento de presentarlos al usuario. Por ejemplo, podemos tener la dimensión Observabilidad, en donde se pueden agrupar patrones como Health Check y Audit Logging.

Los componentes pertenecen a una única dimensión y representan cualquier elemento del sistema que podría aparecer en el diagrama arquitectónico de una solución. Se identifican tres tipos de componentes:

- Patrón: Son soluciones bien establecidas que pueden ser aplicables en diferentes situaciones para resolver problemas arquitectónicos recurrentes [38], [39]. Algunos ejemplos son Api Gateway, Health Check, Sagas, etc.

- Componente arquitectónico: Son los componentes utilizados para representar una arquitectura en un diagrama de alto nivel, suelen representar un conjunto de patrones, tecnologías y/o estrategias. Por ejemplo, la empresa podría detectar un conjunto de patrones de observabilidad como pueden ser Audit Logging, Distributed Tracing y Log Aggregation, que utilicen en todas sus soluciones, y crear un componente arquitectónico reutilizable que contenga estos elementos y permita una solución más sencilla de aplicar y representar.

- Tecnología: tecnología utilizada para implementar algún patrón o componente arquitectónico, por ejemplo PostgreSQL [40], Kubernetes , etc.

Cada componente tiene un conjunto de fuentes de información, las cuales sirven para que el usuario pueda acceder a información de utilidad con respecto al componente seleccionado. Las fuentes de información pueden contener una redirección a una página web con información, o un archivo pdf que se podrá descargar. Las fuentes de información pertenecen a un único componente.

Los elementos más importantes para la toma de decisiones son: Decisión, Decisión Componente y Coexistencia.

Las decisiones representan las posibles elecciones relevantes que un usuario puede tomar al responder las preguntas del sistema, determina qué acciones se toman en lo que respecta a los componentes del resultado final y representa la relación de precondiciones entre preguntas (habilita). Las decisiones tienen dos características importantes que se deben tener en cuenta, la primera es cuándo una decisión se cumple y la segunda es qué acción genera.

Para determinar si una decisión se cumple, se utiliza la relación que tiene con las respuestas (ejecuta). Una decisión se cumple solo si el usuario selecciona todas las respuestas a las que esta está asociadas, entonces esta se cumple y se debe aplicar las acciones que genere.

Una vez que una decisión se cumple, esta puede generar una o varias acciones en el sistema. Estas acciones pueden ser de dos tipos, dependencia de preguntas (relación habilita) y determinación de componentes (relación implica). La primera se utiliza para mostrar nuevas preguntas cuando se cumple una dependencia entre ellas, mientras que la segunda se utiliza para determinar si un componente debe ser incluido o excluido de la arquitectura resultante. Para esta última se utiliza el elemento Decisión Componente, el cual indica si el tipo de acción a tomar es de inclusión o exclusión.

Por último tenemos la Coexistencia, este elemento se encarga de modelar cómo se relacionan los componentes entre sí, para determinar si el resultado final debería agregar algún componente a la arquitectura o marcar alguna incompatibilidad. Existen cuatro tipos de coexistencia:

- Dependencia: Se utiliza para determinar cuando un componente depende de otro.
- Recomendación: Se utiliza cuando la utilización de un componente recomienda la utilización de otro.
- Incompatibilidad: Se utiliza cuando dos componentes no pueden coexistir en la misma arquitectura.
- Implementación: Se utiliza para representar qué tecnologías pueden implementar un componente.

Las coexistencias tienen un papel muy importante en el modelo, ya que son las que proveen la información para detectar incompatibilidades en el resultado final y presentarlas al usuario. También proveen una manera mucho más sencilla de representación de datos para los encargados del sistema. Ingresar preguntas, respuestas y decisiones puede ser un poco trabajoso por la cantidad de elementos que involucra, sin embargo una coexistencia resulta mucho más sencillo. Esto puede simplificar muchos casos en los que los encargados ya saben que existen dos o más componentes que siempre se utilizarán juntos. Lo mismo sucede con las tecnologías del sistema, mediante esta relación se puede tener un mapeo de qué tecnologías incluir por cada componente de la arquitectura, sin la necesidad de incluirlas en la lógica de las decisiones. Esto le ofrece un poco más de libertad a los encargados de como modelar los datos del sistema para que les resulte más conveniente.

5.5. Arquitectura/Diseño

En esta sección del capítulo se presenta la arquitectura y diseño de la solución, que se diseñó basándose en un arquitectura en capas las que se describen en este capítulo. Siguiendo el patrón de arquitectura tiers se logra una solución flexible y sencilla de mantener.

La arquitectura será presentada mediante el modelo de vistas 4 + 1 de Philippe Kruchtend y los diagramas utilizados para describir las vistas se realizaron según UML 2.0.

5.5.1. Vista de Casos de Uso

En esta vista se presentan un subconjunto de casos de uso que son los más relevantes para la definición de la arquitectura.

En el sistema se identifica el actor “Desarrollador” el cual podrá hacer uso de todas las funcionalidades que brinda la interfaz de usuario como por ejemplo crear, ver y eliminar un proyecto, editar respuestas de un proyecto, acceder a las dimensiones y componentes de un proyecto, y acceder a información relevante sobre los distintos componentes que conforman un proyecto de arquitectura en microservicios.

A continuación en la figura 14 se presenta un diagrama de casos de uso en donde se muestra la interacción entre el actor “Desarrollador” y los casos de uso del subconjunto previamente mencionado. Luego se describe los casos de uso “Responder y validar preguntas” y “Generar dimensiones y componentes”

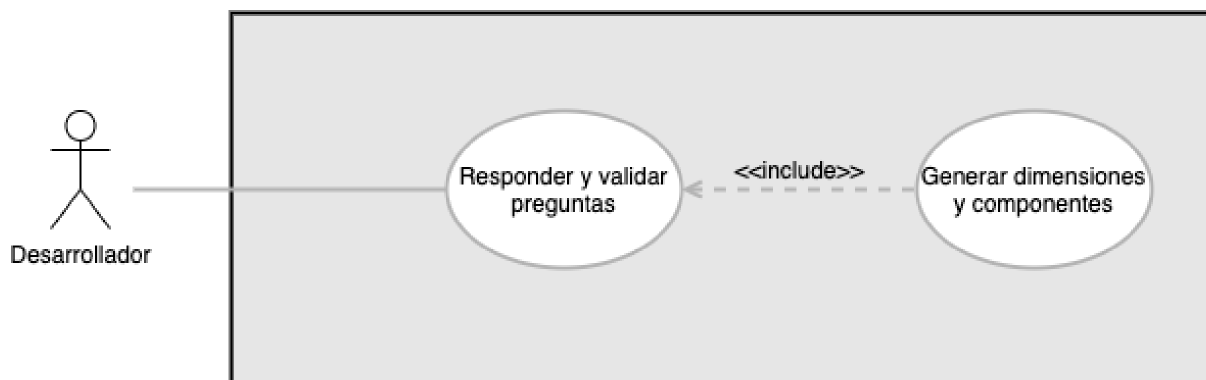


Figura 14. Diagrama de casos de uso.

Nombre: Responder y validar preguntas de un proyecto.
Actores: Desarrollador
Precondiciones: El sistema está en funcionamiento. El usuario selecciona un proyecto el cual está en estado “En Proceso”.
Descripción: El caso de uso comienza cuando el usuario luego de acceder a un proyecto que tiene preguntas sin responder, decide responder una o varias preguntas y para validar las respuesta y saber si completó el proceso de preguntas o si se generan nuevas preguntas. Luego que el sistema termina de validar las respuesta del proyecto, si el proyecto no tiene más preguntas para responder entonces el sistema pasa a ejecutar el caso de uso “Generar dimensiones y componentes de un proyecto”. Luego de finalizado el caso de uso pasa el estado del proyecto a “Finalizado” y el usuario puede comenzar a ver la arquitectura sugerida por la aplicación. En caso que el usuario haya dejado preguntas sin responder, el sistema indica las preguntas pendientes resaltando en color rojo las que falta responder, y en caso de que se generen nuevas preguntas, el sistema mostrará un mensaje informativo.

Flujo normal: 1. El usuario se encuentra en la pantalla de preguntas, en un proyecto el cual se encuentra en estado “En Proceso” y tiene preguntas pendientes para responder. 2. El usuario responde un conjunto de preguntas que están sin responder del proyecto y selecciona “Validar y guardar”. 3. El sistema persiste las respuestas. 4. El sistema valida las respuestas. 5. Luego de validar las respuestas detecta que no hay preguntas sin responder entonces ejecuta el caso de uso incluido “Generar dimensiones y componentes de un proyecto”. 6. Incluye el caso de uso Generar dimensiones y componentes de un proyecto. 7. Cambia el estado a “Finalizado”. 8. El sistema presenta un mensaje de finalización y bloquea las respuestas permitiendo al usuario verlas pero no modificarlas.. 9. Habilita al usuario para que pueda visualizar la arquitectura sugerida. 10. Fin del caso de uso.
Flujo alternativo: **5.A. El sistema detecta que quedan preguntas sin responder en el proyecto.** 1. El sistema indica al usuario las categorías con preguntas sin responder. 2. El sistema indica al usuario las preguntas sin responder en color rojo.
Flujo alternativo: **5.B. El generó nuevas preguntas para el usuario.** 1. El sistema presenta un mensaje para informar al usuario que se generaron nuevas preguntas e indicando las categorías en las que sucedió. 2. El sistema actualiza la lista de preguntas para agregar las nuevas preguntas.
Post-condiciones: En caso de que se respondieron un conjunto de preguntas y quedan preguntas pendientes. El sistema persiste las respuestas. En caso que se respondieron todas las preguntas pendientes del proyecto, el sistema persiste las respuestas, genera las dimensiones y los componentes, luego cambia el estado del proyecto a “Finalizado”.

Nombre: Generar dimensiones y componentes de un proyecto.
Actores: Desarrollador
Precondiciones: El sistema está en funcionamiento y todas las preguntas del proyecto fueron completadas.
Descripción: El caso de uso comienza cuando el usuario ejecuta el caso de uso “Responder y validar preguntas de un proyecto” y el sistema detecta que se completaron todas las preguntas. El sistema debe crear los componentes correspondientes a las respuestas completadas por el usuario, crear componentes que se generan por las coexistencias, detectar incompatibilidades y luego crear las dimensiones. Luego persiste las dimensiones y los componentes con sus incompatibilidades si las hay.

Flujo normal: 1. El sistema se encuentra ejecutando el caso de uso “Responder y validar preguntas de un proyecto” y detecta que el proyecto no tiene preguntas pendientes para responder. 2. El sistema obtiene las decisiones que se cumplen por las respuestas seleccionadas. 3. El sistema crea los componentes en base a las decisiones. 4. El sistema crea componentes nuevos en base a las coexistencias entre componentes. 5. El sistema busca si existen incompatibilidades entre componentes. 6. El sistema crea las dimensiones y le asigna los componentes correspondientes a cada dimensión. 7. El sistema persiste las dimensiones y los componentes del proyecto. 8. Fin del caso de uso.
Post-condiciones: Se crean y persisten las dimensiones y componentes de un proyecto.

5.5.2.Vista lógica

En esta sección se brinda una vista lógica del diseño de la arquitectura del sistema. Para representar la vista lógica se brinda un diagrama de los componentes de la arquitectura, descripción de los mismos y diagramas de secuencia correspondientes a cada caso de uso logrando visualizar la interacción entre componentes lógicos del sistema.

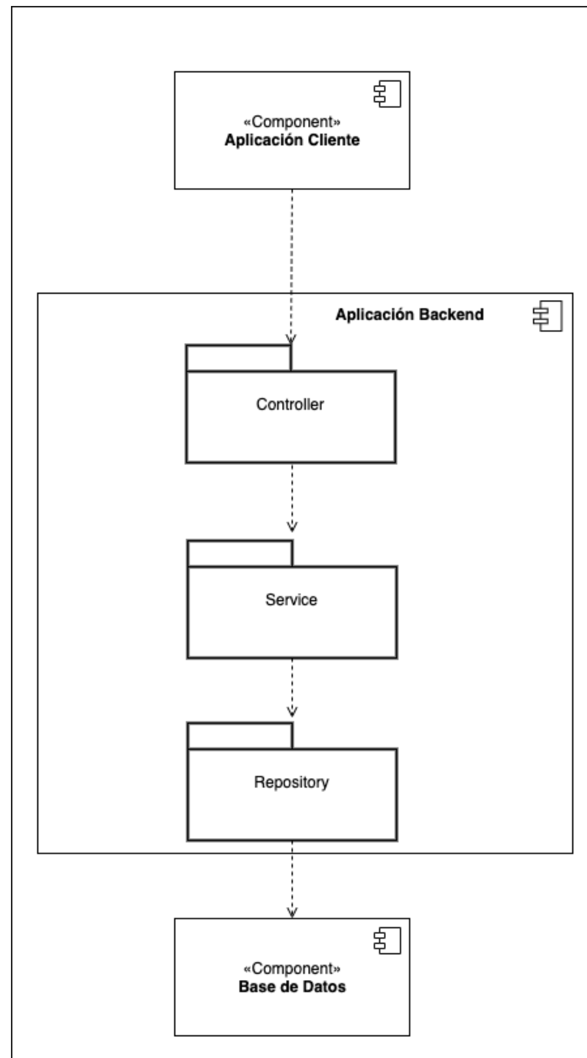


Figura 15. Diagrama de componentes.

En el diagrama de componentes de la figura 15 se brinda los componentes de la arquitectura y los paquetes que se encuentran en el componente "Aplicación Backend".

Aplicación Cliente
Este componente contiene la interfaz de usuario que consiste en una web que le presenta al usuario las distintas funcionalidades de la solución, por ejemplo: crear un nuevo proyecto, ver las dimensiones y componentes de un proyecto, ver las fuentes de información de un componente, etc.

Se comunica con el componente Aplicación Backend a través de la API RESTful que ofrece.

Aplicación Backend
Se encarga de implementar la lógica de negocio y de comunicarse con la base de datos para obtener y persistir los datos del sistema, ofreciendo la API anteriormente mencionada para permitir la ejecución de las distintas funcionalidades. Este componente está integrado por los paquetes Controller, Services y Repository.

Controller

Este paquete pertenece al componente Aplicación Backend, en él se encuentran todos los controladores, qué en conjunto exponen una API RESTful para la ejecución de las funcionalidades de crear, modificar, buscar y eliminar recursos de la aplicación. Algunos de los recursos ofrecidos son: Proyecto, Dimensión, Componente y Archivo.

Service
Este paquete pertenece al componente Aplicación Backend, en él se encuentran todos los servicios que realizan la lógica de negocio de la aplicación y atiende las solicitudes recibidas por los controladores. Cuando este componente precisa obtener o modificar datos se comunica con los repositorios.

Repository
Este paquete pertenece al componente Aplicación Backend, en él se encuentran todos los repositorios, los cuales son los encargados de la persistencia del sistema, por lo que se comunican con el componente Base de Datos, toman los datos y los modela para poder atender las solicitudes de los servicios.

Base de Datos
Este componente cuenta con una base de datos relacional en la cual se encuentra almacenada toda la información necesaria para el funcionamiento del sistema.

En el Anexo 2 se encuentran los diagramas de secuencia que completan la vista lógica.

5.5.3.Vista de procesos

En esta sección se brinda una vista de procesos, la cual consiste en un diagrama de actividad correspondientes a los casos de uso planteados en la sección 5.5.1 (de vista de casos de uso). En el diagrama se van a mostrar acciones encadenadas en alto nivel para representar cada proceso que se está ejecutando en el sistema para cada caso de uso.

Responder y validar preguntas, generar dimensiones y componentes de un proyecto.

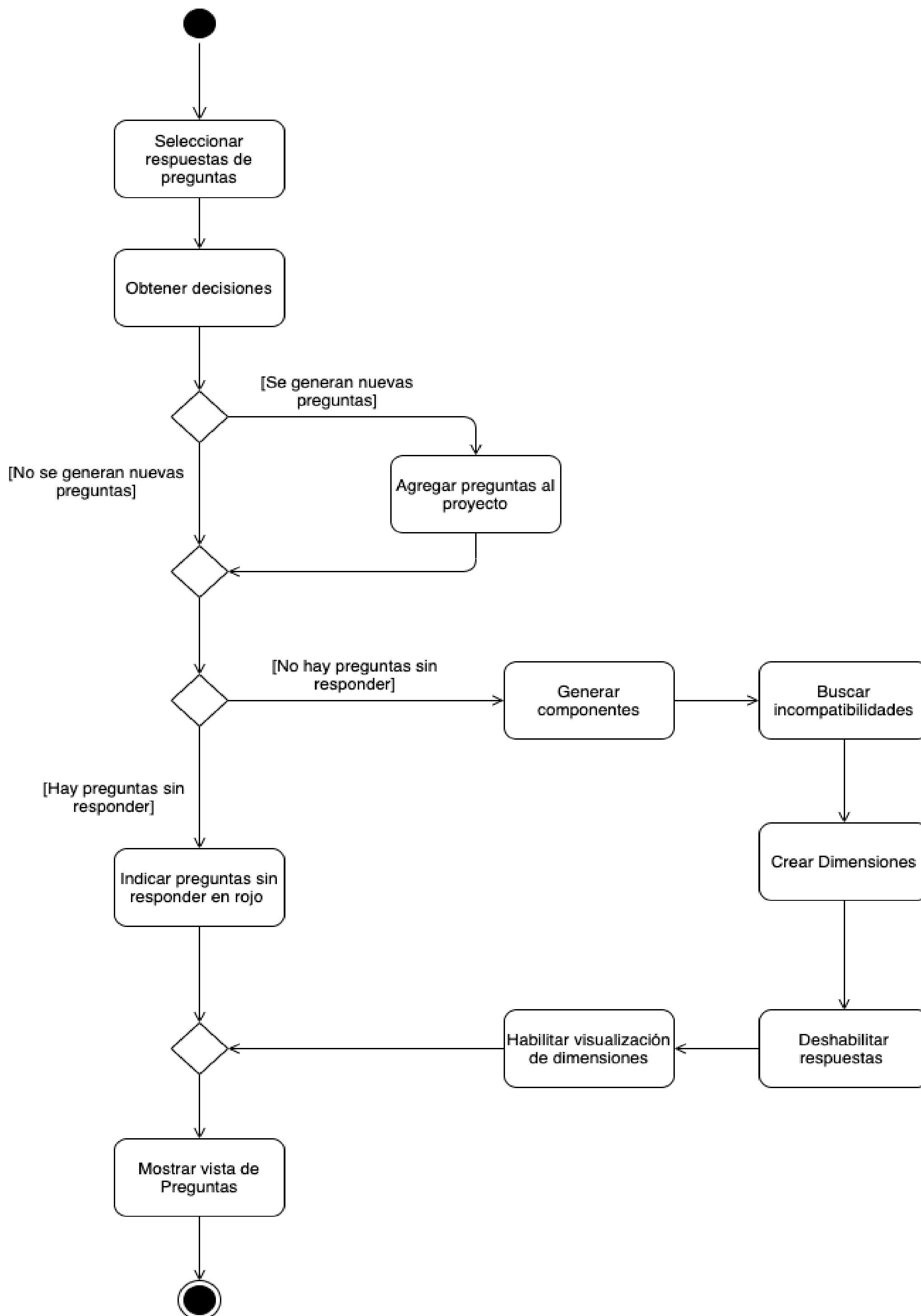


Figura 16. Diagrama de actividad del casos de uso *"Responder y validar preguntas de un proyecto"* y *"Generar dimensiones y componentes de un proyecto"*.

En el diagrama de actividad de la figura 16 muestra las actividades que debe realizar el sistema para validar preguntas, generar dimensiones y componentes de un proyecto. Primero se comienza con las actividades de validar preguntas de un proyecto, para eso el usuario debe seleccionar respuestas para un conjunto de preguntas para luego el sistema obtener las decisiones y validar si se generan nuevas preguntas. En caso que se hayan generado nuevas preguntas el sistema las agrega al proyecto. Luego valida si hay preguntas sin responder. En caso que haya preguntas sin responder, muestra al usuario las preguntas pendientes, en caso

contrario el sistema comienza las actividades que corresponden, generar dimensiones y componentes del proyecto. Comienza por generar las componentes, luego busca si hay incompatibilidades entre las componentes, genera las dimensiones, una vez que se generaron las componentes y las dimensiones del proyecto sigue por deshabilitar la opción de modificar una respuesta y habilita al usuario para que pueda acceder a las dimensiones. Por último se muestran todas las preguntas y respuestas del proyecto.

5.5.4.Vista de desarrollo

En esta sección se brinda una vista de desarrollo que muestra de forma estática la organización de los componentes del software en el ambiente de desarrollo.

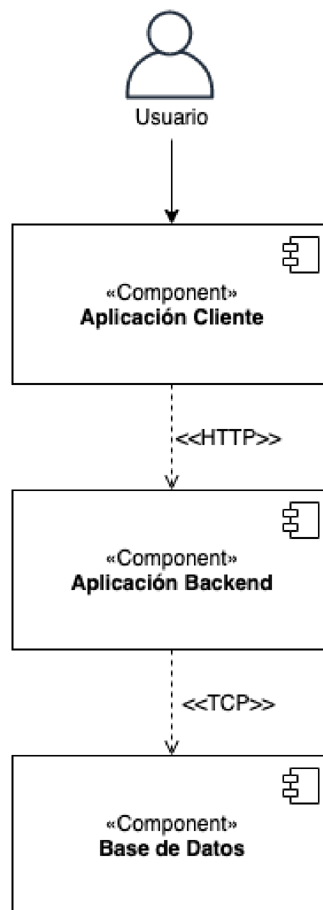


Figura 17. Diagrama de componentes que describe la vista de desarrollo del sistema.

En la figura 17 se muestran los componentes físicos: Aplicación Cliente, Aplicación Backend y Base de Datos. Cada componente representa un artefacto, los componentes Aplicación Cliente y Aplicación Backend consiste en un ejecutable war cada uno, el componente Base de datos consiste en una base de datos relacional que contiene todos los datos del sistema.

El desarrollador accede a la aplicación a través del componente “Aplicación Cliente” que brinda la interfaz de cliente. Luego la “Aplicación Cliente” se comunica vía http con el componente “Aplicación Backend” que realiza la lógica de las solicitudes recibidas. Cuando

el componente “Aplicación Backend” precisa obtener o persistir datos se comunica vía TCP con el componente “Base de datos”.

5.5.5.Vista física

En esta sección se brinda una vista física la cual consiste en un diagrama de despliegue que muestra la distribución de los componentes en tres nodos que son necesarios para desplegar el sistema.

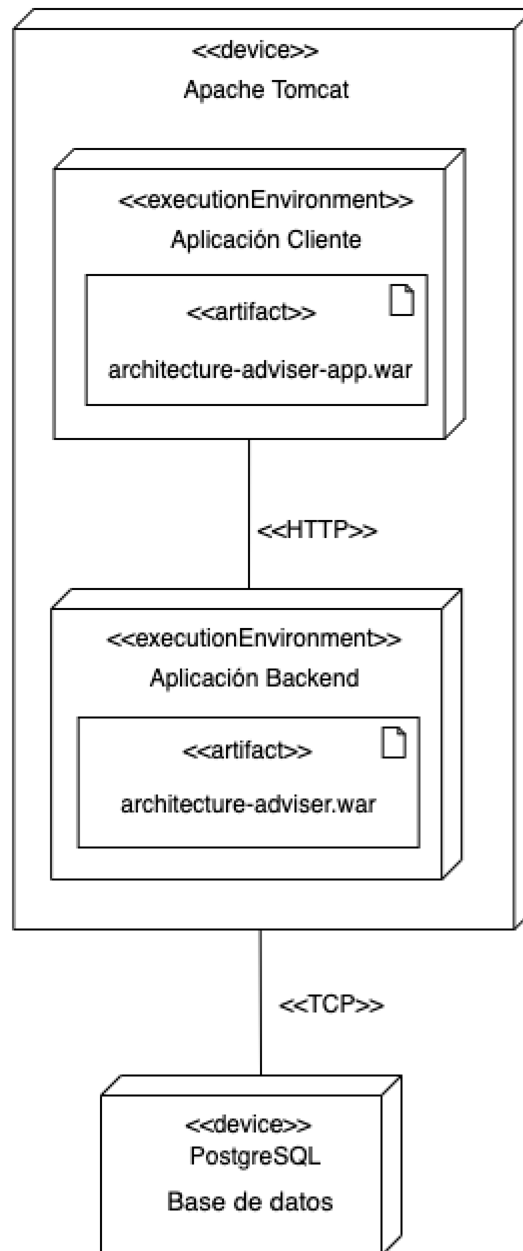


Figura 18. Diagrama de despliegue de los componentes del sistema.

El nodo superior de la Figura 18 contiene la aplicación cliente, en el nodo del medio la aplicación backend y en el nodo inferior la base de datos *PostgreSQL*. La aplicación cliente y backend se comunican vía HTTP y son empaquetadas en los archivos *architecture-adviser-app.war* y *architecture-adviser.war*. Se utiliza *Apache Tomcat* [41] para

desplegar cada archivo. Por último la aplicación backend se comunica con la base de datos vía TCP.

Los tres nodos están localizados en un servidor físico de *Mi nube de Antel* [42], el cual tiene un procesador de 4 núcleos, 8 Gb de RAM, 150Gb de disco y sistema operativo *Linux Ubuntu 16.04*. Dado que el despliegue de la aplicación se realizó en 3 nodos, los mismos podrían ser localizados en servidores físicos distintos.

6. Implementación

En este capítulo se presentan las tecnologías y herramientas utilizadas en la implementación de la solución. Luego se comentan las distintas estrategias utilizadas que le brindan calidad. Por último se comentan las dificultades encontradas en la búsqueda e implementación de la solución.

6.1. Tecnologías

En esta sección se describe las tecnologías utilizadas en la solución, estas tecnologías fueron utilizadas en los tres principales componentes, Interfaz de usuario, Backend y Base de datos.

6.1.1.Interfaz de usuario

La interfaz de usuario fue implementada utilizando React [43] junto con Redux [44], implementando las vistas que interactúan con el usuario e invocando los servicios de la API RESTful que expone el Backend.

React es una librería de JavaScript [45] de código abierto utilizada para crear interfaces de usuario de página única (single page).

Está basado en componentes, lo cual permite implementar distintos componentes para los distintos elementos de la aplicación. Cada componente cuenta con su propio estado, diseño y lógica que permitirán a React renderizarlos de forma eficiente. A su vez el concepto de componente permite aislar distintos elementos de la aplicación, facilitando la separación de tareas y responsabilidades, y ayudando también a la fácil detección de errores.

Redux es una herramienta para la gestión de estado en aplicaciones Javascript. Ayuda a manejar el estado dentro de las aplicaciones imponiendo ciertas restricciones de cómo y cuándo pueden realizarse las actualizaciones, esto facilita mantener la consistencia del estado en toda la aplicación.

Se decidió utilizar esta tecnología para acelerar el desarrollo, ya que de las mencionadas por el cliente, React es la tecnología con la que se contaba más experiencia.

6.1.2.Backend

El backend se implementó como una aplicación Java [46] en capas, utilizando el framework Spring Boot [47].

Estas tecnologías fueron sugeridas por el cliente, ya que la mayoría de sus proyectos son aplicaciones Java y utilizan el framework Spring Boot. También se contaba con la experiencia profesional en construcción de aplicaciones Java lo cual ayudó a acelerar el proceso de implementación.

6.1.3.Base de datos

Como base de datos relacional se utilizó PostgreSQL. Se contaba con experiencia en el tipo de base de datos y en la tecnología. Esta tecnología fue sugerida por el cliente.

6.2. Herramientas de Desarrollo

En esta sección se describen las herramientas utilizadas en la solución.

6.2.1.GitLab

Git es un sistema de control de versionado diseñado para manejar proyectos de todo tamaño, acelerando su velocidad y mejorando su eficiencia. Fue utilizado como repositorio privado de código a través de una cuenta brindada por la Facultad de Ingeniería de la Udelar [48], [49].

6.2.2.Maven

Esta herramienta se utilizó para la compilación, empaquetado y gestión de dependencias en el proyecto Java del componente Backend. También fue utilizado en el proyecto de frontend, pero en este caso fue solo como herramienta de empaquetado [50].

6.2.3.NPM

Esta herramienta se utilizó para realizar la compilación y gestión de dependencias en el proyecto frontend [51].

6.2.4.Tomcat

Es un servidor web que se utilizó para desplegar los proyectos en una máquina virtual y así tener acceso público a la solución. Se utilizó una máquina virtual de mi nube de Antel, la cual fue provista por la Facultad de Ingeniería de la Udelar. En esta se instaló y configuró Tomcat para desplegar los proyectos de la solución.

Se puede acceder a la solución a través de la url:
http://179.27.98.28:8080/architecture-adviser-app

6.2.5.Swagger

Esta herramienta se utilizó para brindar documentación sobre los servicios que brinda la API RESTful que expone el proyecto Backend. Swagger [52] fue sugerido por el cliente, con él establecimos y validamos la información que contiene los servicios documentados.

6.3. Calidad de la solución

En esta sección se explican las estrategias y pruebas que se llevaron a cabo en el proceso de desarrollo. Estas ayudaron a aumentar la calidad de la solución, ya que con ellas se pudo detectar posibles errores y mejorar el funcionamiento de las funcionalidades implementadas.

6.3.1.Revisión de código

Para realizar la revisión de código se utilizó Merge Request de GitLab. Cuando un desarrollador iba a implementar una nueva historia, creaba una nueva rama a partir de la rama develop en el repositorio, en la cual se trabajaba. Una vez finalizada la historia, se subían todos los cambios realizados en la rama y se solicitaba un Merge Request al otro desarrollador. Este realizaba un revisión de código estática, en caso de detectar algún posible error o mejora en la implementación se abría un hilo de discusión entre los desarrolladores para comentar las posibles mejoras o errores. Cuando la rama se encuentra sin ningún hilo de discusión pendiente, el desarrollador aceptaba el Merge Request, lo que llevaba que la implementación de la historia se integre con la rama develop del repositorio.

6.3.2.Análisis de código automatizado

Para realizar el análisis de código automatizado se utilizaron dos herramientas, ESLint [53] y PMD [54]. ESLint se utilizó para el análisis de código de la implementación de la interfaz de usuario y PMD para la implementación de el Backend.

PMD se utilizó para detectar variables y métodos no utilizados, expresiones complejas, código duplicado, posible bugs (por ejemplo generados por variables sin inicializar o excepciones no contempladas) y si se respetan los estándares sugeridos por el lenguaje de programación.

ESLint se utilizó para mantener la calidad del código de frontend, proporcionando ciertas reglas para mantener el código legible y consistente.

Estos beneficios que brindan las dos herramientas colaboraron en la calidad de la solución.

6.3.3.Proceso de testing

Para realizar el testeo de la aplicación se llevó a cabo un proceso testing, el cuál contó con las estrategias de análisis de código automatizado y revisión de código mencionadas en la sección 6.3.1 y 6.3.2, pruebas de regresión y funcionales. Estas dos últimas se explican a continuación para luego seguir con el proceso de testing realizado.

Las pruebas de regresión consisten en un set de pruebas implementadas en JUnit [55]. Estas verifican que las funcionalidades implementadas en los servicios funcionen correctamente, evaluando las funcionalidades con distintas entradas y validando si las salidas son correctas. A medida que se iban implementando las historias se evaluaba si se debía implementar una nueva prueba para integrar al set de pruebas. En este tipo de prueba se realizó para validar

que tras la implementación de una nueva funcionalidad, las funcionalidades implementadas anteriormente no fueron afectadas.

Las pruebas funcionales consisten en realizar pruebas manualmente desde la interfaz de usuario, validando que las funcionalidades implementadas funcionen correctamente. Para este tipo de pruebas se contaba con flujos predeterminados para poder evaluar el correcto funcionamiento de las funcionalidades en su conjunto.

El proceso de testing contiene las siguientes etapas:

1. ***Testing en ambiente local:*** El desarrollador despliega el sistema en su ambiente local y realiza las pruebas de regresión, y funcionales verificando su correcto funcionamiento.

2. ***Análisis de código:*** Una vez finalizada las pruebas el desarrollador realiza el análisis de código automatizado, mencionado en la sección 6.3.2.

3. ***Revisión de código:*** Esta etapa consiste en subir los cambios realizados a la branch perteneciente a la funcionalidad y realizar el proceso de revisión de código, mencionado en la sección 6.3.1.

4. ***Despliegue en testing:*** Luego que la nueva funcionalidad está integrada a la rama develop, se crea el ejecutable y se despliega la aplicación en el ambiente de testing. Al crear el ejecutable se realizan las pruebas de regresión. En caso que no se pase las pruebas de regresión el ejecutable no se crea.

5. ***Pruebas en ambiente testing:*** Una vez que se despliega la aplicación en el ambiente de testing los desarrolladores realizan las pruebas funcionales verificando su correcto funcionamiento.

6.4. Dificultades Encontradas

En esta sección se presentan las dificultades encontradas en la búsqueda e implementación de la solución.

6.4.1.Versionado del modelo

Estando en la etapa de implementación surgió un nuevo requerimiento que no fue contemplado y resultaba necesario. Vimos que con el tiempo se iban a agregar, eliminar o modificar datos del modelo, lo que llevaría a que proyectos existentes no se encuentren actualizados según los datos modificados. Analizamos que la mejor solución era crear un versionado del modelo.

6.4.2.Ataques al ambiente de testing

Durante el proceso de implementación vimos que el ambiente de testing tenía una baja performance, analizamos la memoria RAM y CPU de la máquina virtual que se encuentra en

Mi nube de Antel y observamos que no tenía disponible ninguno de los recursos. Luego de buscar posibles causas detectamos que se habían establecido mineros, los cuales estaban utilizando recursos de la máquina virtual.

Para solucionar el problema tuvimos que crear y configurar nuevamente el ambiente de testing, y aumentar la seguridad.

La búsqueda y solución del problema implicó una carga de trabajo que no fue contemplada.

6.4.3.Construcción de pruebas en JUnit

Algunas de las pruebas implementadas en JUnit que validan el correcto funcionamiento de las funcionalidades llevaron más tiempo de lo esperado. Estas estaban relacionadas a la selección de respuestas y generación de componentes. Esta dificultad se dio debido a la gran cantidad de datos y flujos que se deben controlar para validar el correcto funcionamiento.

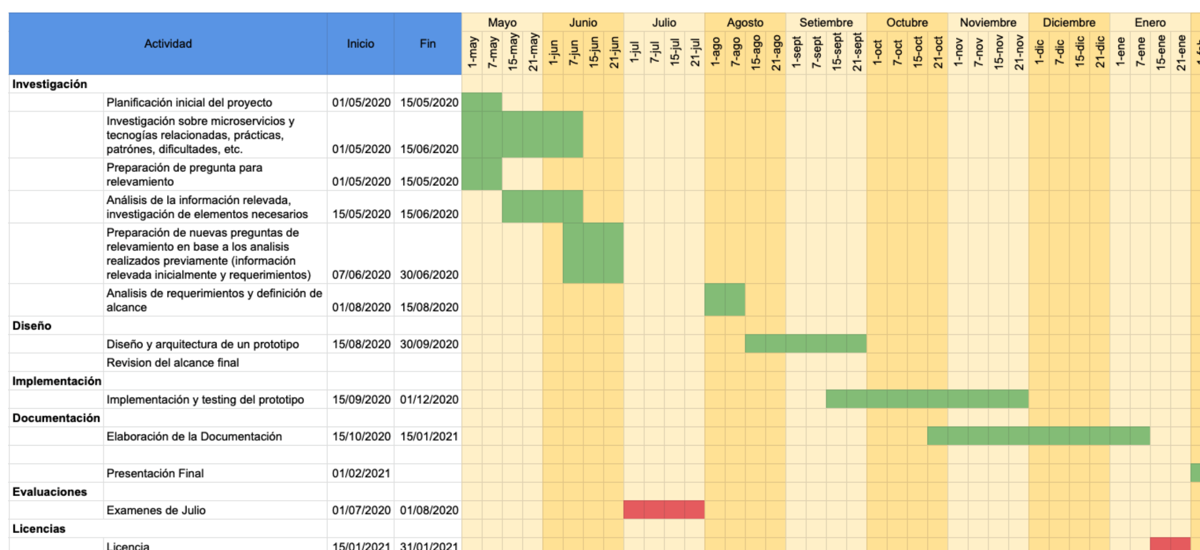
7. Gestión del Proyecto

A continuación se presenta el proceso de trabajo llevado a cabo durante todo el proyecto, identificando sus distintas etapas, presentando la planificación realizada, y como esta se contrasta con la realidad del proyecto. El capítulo comienza con la planificación de la etapa inicial, seguido de la planificación de la etapa final, finalizando con una sección de dificultades encontradas en lo que respecta a la gestión del proyecto.

7.1. Planificación inicial

Durante el comienzo del proyecto se realizó una planificación de todas las etapas que éste tendría, la cual se puede observar en la figura 19. La planificación comienza con las tareas de planificación, investigación y capacitación sobre microservicios en general, preparación de las preguntas iniciales para el relevamiento, análisis de la información relevada e investigación necesaria para la completa comprensión de las respuestas, preparacion de nuevas preguntas, y por último, analizar los requerimientos y definir el alcance.

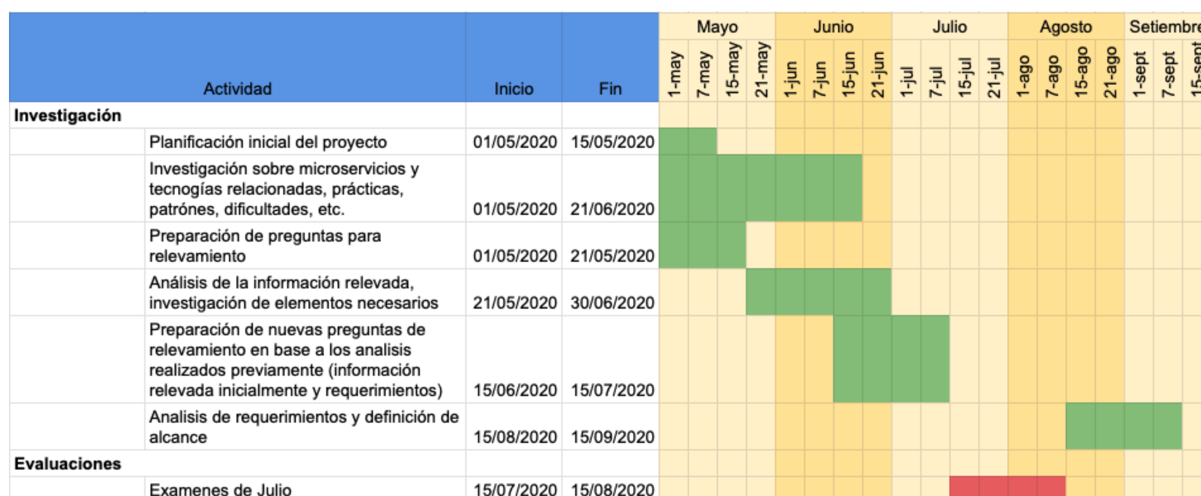
Las etapas recién mencionadas fueron las que realmente se llevaron a cabo siguiendo esta planificación, luego de estas se realizó una nueva planificación para el proyecto que contempla el cambio de objetivo que se mencionó en los capítulos anteriores.



Actividad		Inicio	Fin
Investigación			
	Planificación inicial del proyecto	01/05/2020	15/05/2020
	Investigación sobre microservicios y tecnogías relacionadas, prácticas, patrónes, dificultades, etc.	01/05/2020	15/06/2020
	Preparación de pregunta para relevamiento	01/05/2020	15/05/2020
	Análisis de la información relevada, investigación de elementos necesarios	15/05/2020	15/06/2020
	Preparación de nuevas preguntas de relevamiento en base a los analisis realizados previamente (información relevada inicialmente y requerimientos)	07/06/2020	30/06/2020
	Analisis de requerimientos y definición de alcance	01/08/2020	15/08/2020
Diseño			
	Diseño y arquitectura de un prototipo	15/08/2020	30/09/2020
	Revision del alcance final		
Implementación			
	Implementación y testing del prototipo	15/09/2020	01/12/2020
Documentación			
	Elaboración de la Documentación	15/10/2020	15/01/2021
	Presentación Final	01/02/2021	
Evaluaciones			
	Examenes de Julio	01/07/2020	01/08/2020
Licencias			
	Licencia	15/01/2021	31/01/2021

Figura 19. Planificación inicial del proyecto.

A pesar de que se intentó respetar la planificación lo más que se pudo, esta presenta algunos cambios con lo sucedido realmente. Algunas tareas se extendieron una o incluso dos semanas más de lo esperado, además las fechas de exámenes no fueron las esperadas y llevaron a que el receso planificado se pospusiera dos semanas. En la figura 20 se puede observar lo sucedido realmente durante el transcurso de esta etapa.

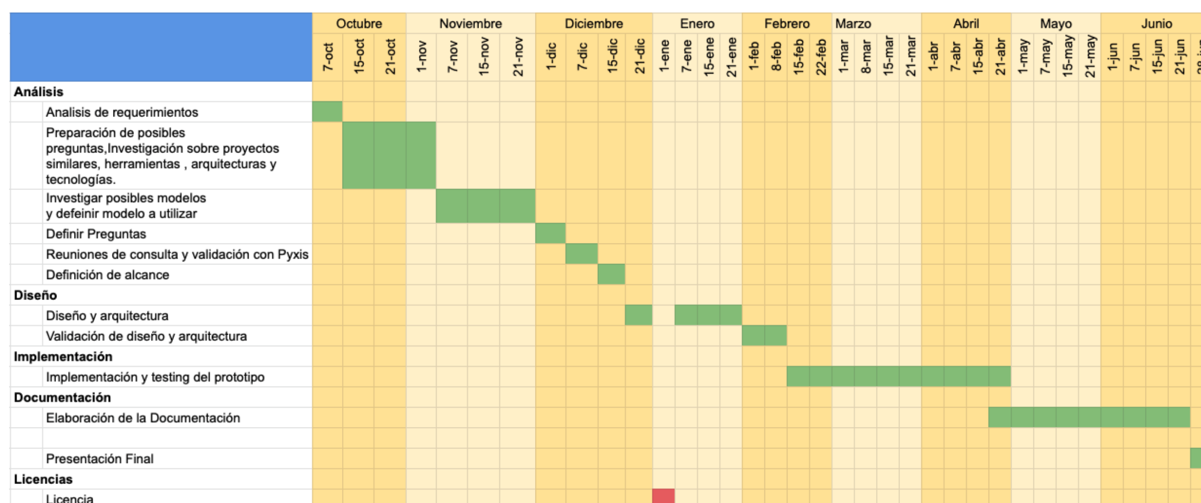


Actividad	Inicio	Fin
Investigación		
Planificación inicial del proyecto	01/05/2020	15/05/2020
Investigación sobre microservicios y tecnogías relacionadas, prácticas, patrónes, dificultades, etc.	01/05/2020	21/06/2020
Preparación de preguntas para relevamiento	01/05/2020	21/05/2020
Análisis de la información relevada, investigación de elementos necesarios	21/05/2020	30/06/2020
Preparación de nuevas preguntas de relevamiento en base a los analisis realizados previamente (información relevada inicialmente y requerimientos)	15/06/2020	15/07/2020
Analisis de requerimientos y definición de alcance	15/08/2020	15/09/2020
Evaluaciones		
Examenes de Julio	15/07/2020	15/08/2020

Figura 20. Proceso real de la etapa inicial del proyecto.

7.2. Planificación Final

Luego de la presentación de propuestas hubo un periodo de 3 semanas en las que quedamos a la espera de una respuesta por parte del cliente. Durante estas semanas nos dedicamos a avanzar con la implementación de un prototipo de Contract Testing ya que fue una de las propuestas que más le había interesado al cliente, sin embargo este fue descartado en una etapa temprana, ya que una vez obtenida la respuesta del cliente se nos presentó el nuevo objetivo del proyecto.



Actividad
Análisis
Analisis de requerimientos
Preparación de posibles preguntas,Investigación sobre proyectos similares, herramientas , arquitecturas y tecnologías.
Investigar posibles modelos y defeinir modelo a utilizar
Definir Preguntas
Reuniones de consulta y validación con Pyxis
Definición de alcance
Diseño
Diseño y arquitectura
Validación de diseño y arquitectura
Implementación
Implementación y testing del prototipo
Documentación
Elaboración de la Documentación
Presentación Final
Licencias
Licencia

Figura 21. Planificación Final.

En la figura 21 se puede observar la planificación realizada en el comienzo de la etapa final. Tras el cambio de objetivo se intentó mantener acotada la duración de las distintas etapas para evitar un excesivo aumento de duración en el proyecto, pero debido a que se debían repetir tareas de investigación y análisis el aumento de duración del proyecto fue inevitable.

En primera instancia, lo que sucedió realmente en el proyecto coincide con la planificación, pero como se puede observar en la figura 22, la etapa de documentación se extendió mucho más de lo esperado. Durante la etapa de documentación también hubo que hacer algunos

arreglos de bugs encontrados y mejoras solicitadas sobre la aplicación. También se debió dedicar tiempo a la configuración de un nuevo ambiente de testing debido a que se venció la licencia de Mi Nube Antel.

Análisis
Analisis de requerimientos
Preparación de posibles preguntas,Investigación sobre proyectos similares, herramientas , arquitecturas y tecnologías.
Reuniones de consulta y validación con Pyxis
Definición de alcance
Diseño
Diseño y arquitectura
Validación de diseño y arquitectura
Implementación
Implementación y testing del prototipo
Documentación
Elaboración de la Documentación
Preparacion de la presentación
Presentación Final
Evaluaciones
Parciales y Examenes
Licencias
Licencia

Figura 22. Proceso real de la etapa final del proyecto.

7.3. Metodología de Trabajo

En cuanto a la forma de trabajo, durante su mayor parte se buscó orientarla a una metodología ágil. Tanto nosotros como el cliente consideramos conveniente mostrar avances cada cierto tiempo estipulado, por lo que se propuso utilizar el concepto de sprint que se utiliza en Scrum, cuya duración puede variar dependiendo de la etapa del proyecto. Cabe destacar que a pesar de que se utilizó una metodología de trabajo inspirada en Scrum, no se utilizó Scrum de forma estricta, ya que no se cumplían todas las ceremonias y características que lo identifican.

Para nuestro caso se contaba con un backlog de tareas, se trabajaba con modalidad de sprint, seleccionando al comienzo el alcance del mismo como en una reunión de planificación (planning), se realizaban reuniones de revisión (review) con el cliente al final de cada sprint, pero no se contaba con reuniones diarias (daily), o reuniones de retrospección (retrospective), tampoco se utilizaba un sistema de puntuación de tareas ya que consideramos que todos estos elementos podían quitarnos tiempo valioso y no se ajustaban a nuestras necesidades.

7.3.1. Análisis

Debido al cambio de objetivo del proyecto, la etapa de análisis se extendió más de lo deseado, ya que hubo dos etapas de análisis, una para la etapa inicial y otra para la etapa final.

Para la etapa inicial se propuso un periodo inicial de cuatro semanas para investigar sobre microservicios y formular el primer conjunto de preguntas. A partir de ese momento se realizó el mismo procedimiento más de una vez, pero con duración de dos semanas.

Mirándolo en términos de la metodología de trabajo descrita, podemos decir que hubo un sprint inicial de cuatro semanas y luego se realizaron sprints de dos semanas. Para este caso

podría considerarse como reunión de revisión (review) el momento en el que se presentaban las preguntas, el cual se hizo en dos modalidades dependiendo de la necesidad y disponibilidad de los involucrados. Las dos modalidades fueron videollamadas e intercambio de correos.

Para la etapa final se realizaron sprints de dos semanas y las reuniones de revisión pasaron a ser solamente mediante videollamadas.

7.3.2. Diseño e Implementación

Para las etapas de diseño e implementación se utilizaron sprints de dos semanas, las reuniones de revisión se utilizaron también como reuniones de planificación (planning) para obtener en el momento la validación del cliente sobre las tareas que se deberían incluir en el siguiente sprint. Estas reuniones se realizaron en modalidad de videollamada.

7.3.3. Documentación

Para la etapa de documentación se propuso inicialmente utilizar un proceso similar, en el cual se ajustaría la duración del sprint para realizar entregas secuenciales de los avances al tutor. Sin embargo, al comienzo de esta etapa nos dimos cuenta de que los requerimientos mínimos del tutor para la correcta revisión del documento eran más costosos de lo que parecían en un principio, y de que la velocidad de avance en la tarea de documentación era menor a la esperada. Esto dificultó mucho seguir un proceso como el que se había propuesto y no se pudo llevar a cabo. Lo que terminó sucediendo fue que se proponía una tarea por parte del tutor sobre lo que se debería avanzar en la documentación para una correcta revisión y esta se entregaba una vez que se finalizaba, siempre buscando terminar lo antes posible.

7.4. Dificultades encontradas

Al momento de planificar detectamos algunas características del proyecto que resultaron especialmente difícil de estimar y planificar.

La primera dificultad que encontramos fue la de estimar las distintas etapas de un proyecto que nos causaba mucha incertidumbre. Resultó difícil estimar cuánto llevaría relevar las arquitecturas caracterizadas por distintos elementos de los cuales no contábamos con conocimientos ni capacitación. Pero resultó aún más difícil realizar una justa estimación para la búsqueda de mejoras sobre arquitecturas que aún no habían sido relevadas. De la misma forma que planificar la implementación de un prototipo de estas propuestas, dado que aún no sabíamos cuáles serían ni cual sería su dificultad.

Otra dificultad que se presentó fue planificar la etapa final. Luego del cambio de objetivos nos vimos obligados a realizar una nueva planificación para el proyecto, en la cual se deberían repetir tareas ya realizadas. Dado el tiempo ya transcurrido para la primera etapa, resultó difícil llegar a una planificación que nos otorgara el tiempo necesario para cada una de las etapas que se debían realizar sin extender demasiado la duración total del proyecto.

Una dificultad que se presentó durante el final de la etapa de implementación fue generar la información necesaria para el prototipo. Este prototipo necesita una gran cantidad de datos, y la búsqueda o generación de una parte de estos datos no fue contemplada en la planificación. Por un lado el prototipo necesita cierto conjunto de datos que le permite ejecutar su lógica de negocio, la generación de estos datos se contempló como parte de la etapa de implementación. Por otro lado tenemos los datos informativos que se le presentan al usuario dentro del prototipo, como las descripciones, imágenes y fuentes de información que proveen los componentes. La búsqueda y recolección de estos datos no fue contemplada en la planificación, lo cual dificultó la ejecución de la planificación, ya que debido a la cantidad de componentes que tiene el prototipo, esta tarea requirió un gran esfuerzo. A causa de que esta tarea no fue contemplada, debió ser realizada dentro de la etapa de documentación, lo cual impactó directamente en la velocidad en la que se avanzó durante esta etapa.

8. Conclusiones y trabajo futuro

En este capítulo se presentan las conclusiones a las que se llegaron una vez concluido el proyecto de grado y el trabajo que se podría realizar a futuro para la mejora del prototipo implementado.

8.1. Conclusiones

Como conclusión general del proyecto se destaca el cumplimiento de sus objetivos. Se logró realizar el relevamiento de las arquitecturas utilizadas por Pyxis, el cual fue validado. Asimismo, se realizó un estudio de la industria y del gap técnico con respecto a las arquitecturas relevadas, y se realizaron propuestas para mejorarlas. Luego de esto, surgió una readecuación del alcance del proyecto, generando como nuevo objetivo, diseñar e implementar una herramienta que asesore a los usuarios en la construcción de un sistema con una arquitectura de microservicios. Este objetivo también fue cumplido, replanificando el proyecto e investigando posibles modelos de toma de decisiones. Tras un análisis de los modelos investigados, se concluyó que ninguno de ellos resolvía todos los aspectos de la problemática, por lo que se procedió a construir un modelo propio. Por último se realizó la implementación del prototipo deseado utilizando el modelo de toma de decisiones construido.

Tras el análisis de las arquitecturas relevadas, llegamos a la conclusión de que estas resuelven correctamente los problemas a los que se enfrentan, y que cuentan con una sólida base de patrones y tecnologías.

Las propuestas de mejora presentadas consistieron en incluir ciertos patrones que no se estaban tomando en cuenta y tecnologías que podrían mejorar el área de testing. En particular se propusieron los patrones Contract Testing y Transactional Outbox con implementación de tipo Transactional Log Tailing. Como tecnología se propuso utilizar Chaos Monkey For Spring Boot para introducir un tipo de testing que no se estaba utilizando, como lo es la ingeniería del caos.

Con respecto a la herramienta implementada, se logró una solución web responsive que ayuda al usuario a diseñar una arquitectura en microservicios. Esto se logra mediante preguntas que éste debe responder de forma obligatoria, evitando de esta forma, que olvide algún aspecto importante de la solución. La herramienta además de recomendar componentes, patrones y tecnologías, presenta las posibles incompatibilidades entre estos elementos, permitiendo al usuario tomar la decisión final de cuales de ellos incluir. También permite ingresar la descripción e imágenes que desea que se muestre sobre cada componente y cuales son las fuentes de información que tiene asociadas.

Para la toma de decisiones se construyó un modelo que resulta extensible, ya que permite agregar nuevos componentes, patrones y tecnologías que puedan surgir. También resulta flexible, ya que permite a la persona a cargo ingresar el conjunto de datos que desee, incluyendo categorías, preguntas, respuestas, dimensiones, componentes y fuentes de

información. A través de los datos ingresados la persona a cargo puede configurar las decisiones que se toman, es decir que puede configurar como las respuestas ingresadas recomiendan componentes.

Mirando en retrospectiva, consideramos que para realizar un relevamiento como el realizado en la etapa inicial, es más que conveniente contar con experiencia o conocimientos previos sólidos de los conceptos más importantes. En nuestro caso no teníamos experiencia con microservicios, lo cual puede haber incrementado considerablemente la dificultad del proyecto. Las primeras preguntas generadas resultaron superficiales debido a la cantidad de conocimientos que pudimos adquirir en un periodo de tiempo limitado, una persona con experiencia previa en el tema quizás podría haber comenzado con preguntas más específicas y relevantes que facilitarían la posterior comprensión de las arquitecturas. Otro aspecto que nos lleva a sacar esta conclusión es el nivel de conocimiento de los clientes, los clientes de este proyecto tienen mucha experiencia en el área, por lo que mantener el hilo de la conversación en las reuniones era un reto. Se necesitó realizar mucha investigación en las primeras etapas, no solo para la formulación de preguntas y análisis del relevamiento, también para la completa comprensión de las conversaciones con los clientes.

En lo personal, elegimos este proyecto porque estábamos interesados en microservicios y queríamos aprender más al respecto. Si bien nos habría gustado implementar algo en microservicios para verlo en la práctica, estamos conformes con lo aprendido a nivel teórico y cabe destacar que estos conocimientos adquiridos fueron de ayuda en el ámbito laboral.

Si bien el rumbo del proyecto cambió, consideramos que se cumplió el objetivo que este tenía desde un principio. Inicialmente se buscaba construir un prototipo con mejoras para aplicar sobre las arquitecturas, pero esto era un medio para cumplir un objetivo más general, el cual era mejorar la calidad de las soluciones propuestas por Pyxis. Este objetivo se mantuvo a pesar del cambio sucedido en el proyecto, ya que el prototipo implementado permitirá mejorar los procesos de trabajo dentro de la empresa y facilitará la documentación de sus proyectos, permitiendo a los referentes delegar parte de su trabajo y enfocarse en tareas de mayor importancia. Además, tras la correcta configuración de las reglas del prototipo, la empresa contará con una herramienta que permitirá mantener la consistencia entre las distintas arquitecturas a construir, minimizando las diferencias causadas por distintas opiniones de los distintos arquitectos.

8.2. Trabajo Futuro

En este capítulo se presenta el trabajo que se podría realizar a futuro para la mejora del prototipo implementado.

Acceso al sistema

Incluir la posibilidad de que los usuarios se registren y accedan al sistema, para poder así filtrar la información que se presenta y mostrar solamente la que resulte relevante para dicho usuario. Se deberían reconocer dos tipos de usuarios en el sistema, los usuarios

administradores, encargados del mantenimiento y control del mismo, y los usuarios finales que utilizarán el sistema para realizar relevamientos de arquitecturas.

Reportar Fuentes de información

Incluir la posibilidad de que los usuarios que utilizan la herramienta puedan reportar fuentes de información por diferentes motivos, por ejemplo si la información se encuentra desactualizada. Mediante esta funcionalidad el usuario podría seleccionar el motivo por el cual reporta la fuente de información e ingresar una explicación. Esta información luego sería envíada a los administradores a cargo del sistema para que puedan hacer algo al respecto.

Funcionalidades de Backoffice

El prototipo implementado durante este proyecto solo muestra una cara de lo que sería un sistema completo, para que el sistema esté completo se deben ofrecer las funcionalidades que permiten a un administrador tener el control del sistema y de cómo este funciona. Entre estas funcionalidades se debe permitir a los usuarios administradores crear, modificar y eliminar dimensiones, componentes, fuentes de información, categorías, preguntas, respuestas y decisiones. Además debe existir una funcionalidad que permita configurar como todos estos elementos se relacionan para otorgar un resultado a los usuarios finales que responden las preguntas. Para obtener información más detallada de estos requerimientos puede acceder al anexo 3.

9. Referencias

[1] "Why You Can". Consultado: dic. 05, 2021. [En línea]. Disponible en: https://smartbear.com/blog/why-you-cant-talk-about-microservices-without-ment/

[2] "What are Microservices? | IBM". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://www.ibm.com/cloud/learn/microservices

[3] M. Fowler, "Microservices". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://martinfowler.com/articles/microservices.html

[4] C. H. Gammelgaard, *Microservices in .NET Core, with examples in Nancy*. Shelter Island, NY: Manning Publications Co, 2017.

[5] P. Mikaela, M. Romina, y G. Nicolás, "Aplicación de Microservicios sobre una arquitectura SOA con restricciones de calidad de servicio.pdf". Abril 2016.

[6] "What is a Container? | App Containerization | Docker". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://www.docker.com/resources/what-container

[7] "Virtual Machine". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://www.vmware.com/topics/glossary/content/virtual-machine

[8] B. Scholl, "Microservices and Containers: Patterns for Scalable Agility in the Age of Digital Disruption." [En línea]. Disponible en: https://www.oracle.com/technetwork/articles/dsl/microservices-containers-pattern-wp.pdf

[9] C. Richardson, "API Gateway". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://microservices.io/patterns/apigateway.html

[10] "Circuit Breaker". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://martinfowler.com/bliki/CircuitBreaker.html

[11] C. Richardson, "Service Discovery in a Microservices Architecture", oct. 12, 2015. https://www.nginx.com/blog/service-discovery-in-a-microservices-architecture/ (consultado nov. 28, 2021).

[12] "Implement health check APIs for microservices." Consultado: nov. 28, 2021. [En línea]. Disponible en: https://www.ibm.com/garage/method/practices/manage/health-check-apis/

[13] "Microsoft Docs - Transacciones distribuidas saga". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://docs.microsoft.com/es-es/azure/architecture/reference-architectures/saga/saga

[14] P. Loganathan, "Transactional Outbox Pattern", jul. 16, 2019. https://pradeeploganathan.com/patterns/transactional-outbox-pattern/ (consultado nov. 28, 2021).

[15] A. Yildirim, "Transaction Log Tailing With Debezium", *Trendyol Tech*, mar. 03, 2020. https://medium.com/trendyol-tech/transaction-log-tailing-with-debezium-part-1-aeb968d72220 (consultado nov. 28, 2021).

[16] G. Morling, “Reliable Microservices Data Exchange With the Outbox Pattern”, *Debezium*, feb. 2019. https://debezium.io/blog/2019/02/19/reliable-microservices-data-exchange-with-the-outbox-pattern/ (consultado nov. 28, 2021).

[17] Ma, “What is contract testing?”, *Pactflow Contract Testing Platform*, jul. 19, 2019. https://pactflow.io/blog/what-is-contract-testing/ (consultado nov. 28, 2021).

[18] “How Pact works | Pact Docs”. Consultado: nov. 28, 2021. [En línea]. Disponible en: https://docs.pact.io/getting_started/how_pact_works/

[19] T. Muc, “The Basics of Contract Tests”, *Medium*, ene. 18, 2021. https://codeburst.io/the-basics-of-contract-tests-920ff363c820 (consultado nov. 28, 2021).

[20] “What are the benefits of contract testing?” Consultado: nov. 28, 2021. [En línea]. Disponible en: https://pactflow.io/what-are-the-benefits-of-contract-testing/

[21] “PRINCIPLES OF CHAOS ENGINEERING”. Consultado: nov. 28, 2021. [En línea]. Disponible en: https://principlesofchaos.org/

[22] C. Richardson, “The Microservice Architecture Assessment”. Consultado: nov. 28, 2021. [En línea]. Disponible en: https://microservices.io/platform/microservice-architecture-assessment.html

[23] “IBM Cloud Architectures”. Consultado: nov. 28, 2021. [En línea]. Disponible en: https://www.ibm.com/cloud/architecture/architectures/

[24] S. Farshidi, S. Jansen, y J. Werf, “1 Capturing software architecture knowledge for pattern-driven design”, *J. Syst. Softw.*, vol. 169, p. 110714, jul. 2020, doi: 10.1016/j.jss.2020.110714.

[25] D. Ameller y X. Franch, “Assisting software architects in architectural decision-making using Quark”, *CLEI Electron. J.*, vol. 17, pp. 1–20, dic. 2014, doi: 10.19153/cleiej.17.3.1.

[26] “Kubernetes”. Consultado: nov. 28, 2021. [En línea]. Disponible en: https://kubernetes.io/

[27] “Configure Liveness, Readiness and Startup Probes”. Consultado: nov. 28, 2021. [En línea]. Disponible en: https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/

[28] “Elasticsearch”. Consultado: nov. 28, 2021. [En línea]. Disponible en: https://www.elastic.co/es/

[29] “Elastic APM: Monitoreo de rendimiento de aplicaciones”. Consultado: nov. 28, 2021. [En línea]. Disponible en: https://www.elastic.co/es/apm/

[30] “Kibana: Explora, visualiza y descubre datos”. Consultado: nov. 28, 2021. [En línea]. Disponible en: https://www.elastic.co/es/kibana

[31] C. Richardson, “microservices.io”. Consultado: nov. 28, 2021. [En línea]. Disponible en: https://microservices.io/

[32] "Service Discovery: Eureka Clients". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://cloud.spring.io/spring-cloud-netflix/multi/multi__service_discovery_eureka_clients.html

[33] "Spring Cloud". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://spring.io/projects/spring-cloud

[34] "Apache Kafka". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://kafka.apache.org/

[35] "Introduction | Pact Docs". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://docs.pact.io/

[36] "Spring Cloud Contract". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://spring.io/projects/spring-cloud-contract

[37] P. Kruchten, *Planos Arquitectónicos: El Modelo de "4+1" Vistas de la Arquitectura del Software*. [En línea]. Disponible en: http://materias.fi.uba.ar/7510/practica/zips/Modelo4_1Krutchen.pdf

[38] P. Avgeriou y U. Zdun, "Architectural Patterns Revisited – A Pattern Language".

[39] M. Fowler, "Writing Software Patterns". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://martinfowler.com/articles/writingPatterns.html

[40] "PostgreSQL". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://www.postgresql.org/

[41] "Apache Tomcat". Consultado: nov. 28, 2021. [En línea]. Disponible en: http://tomcat.apache.org/

[42] "Mi nube de Antel". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://minubeantel.uy/

[43] "React". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://reactjs.org/

[44] "Redux". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://es.redux.js.org/docs/introduccion/motivacion.html

[45] "JavaScript". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://www.javascript.com/

[46] "Java Software". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://www.oracle.com/java/

[47] "Spring Boot". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://spring.io/projects/spring-boot

[48] "Git". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://git-scm.com/

[49] "GitLab". Consultado: nov. 28, 2021. [En línea]. Disponible en: https://about.gitlab.com/

[50] "Apache Maven". Consultado: nov. 28, 2021. [En línea]. Disponible en:

https://maven.apache.org/

[51] “NPM”. Consultado: nov. 28, 2021. [En línea]. Disponible en: https://www.npmjs.com/

[52] “Swagger”. Consultado: nov. 28, 2021. [En línea]. Disponible en: https://swagger.io/

[53] “ESLint”. Consultado: nov. 28, 2021. [En línea]. Disponible en: https://eslint.org/

[54] “PMD”. Consultado: nov. 28, 2021. [En línea]. Disponible en: https://pmd.github.io/

[55] “JUnit”. Consultado: nov. 28, 2021. [En línea]. Disponible en: https://junit.org/junit5/

10. Glosario

Backlog: En metodologías ágiles se refiere a lista de requerimientos o tareas del producto o sistema a desarrollar que se encuentran por implementar.

***Dimensión*:** Identifica un aspecto importante en una arquitectura como lo puede ser seguridad, observabilidad, comunicación, etc.

***Componente*:** Identifica un elemento de la arquitectura, puede ser un componente arquitectónico, un patrón o una tecnología.

Fuente de información: Puede ser un vínculo a una página externa o archivo con información relevante sobre un componente.

Modelo: Se define como modelo al conjunto de datos del sistema que son necesarios para su utilización (preguntas, respuestas, decisiones, componentes, etc.)

Categoría: Se utiliza para la categorización de preguntas, todas las preguntas que se le presenten al usuario deben pertenecer a una categoría.

Timeout: Al realizar una consulta a un servicio remoto, se le llama timeout a una consulta que no provee una respuesta antes de que se agote el límite de tiempo máximo que tiene definido.

11. Anexos

11.1. Anexo 1: Requerimientos dentro del alcance

A continuación en la figura 23, se presentan los requerimientos de la aplicación cliente de Architecture Adviser. Estos conforman todos los requerimientos que se definieron dentro del alcance del proyecto, los requerimientos que fueron quitados del proyecto al momento de definir el alcance se pueden encontrar en el anexo 3.

Ref.	Nombre	Descripción
RA1	Crear proyecto	Se debe poder crear un proyecto indicando nombre, descripción, nombre del autor, correo del autor, versión de modelo y url.
RA2	Eliminar proyecto	Se debe poder realizar una eliminación de tipo lógico de un proyecto del sistema.
RA3	Visualizar y acceder proyectos	Se debe poder acceder a una lista de los proyectos creados en el sistema. Al seleccionar uno de ellos se debe mostrar un formulario que contendrá las preguntas del mismo. Este formulario contendrá las categorías del sistema y se mostrarán solamente las preguntas de la categoría que se encuentre actualmente seleccionada.
RA4	Responder y validar preguntas	Una vez accedido un proyecto se debe poder responder las preguntas que el sistema presenta. Debe haber un botón que al presionar valide las respuestas ingresadas y actualice el formulario como corresponda (generar nuevas preguntas, o borrar aquellas que no dejen de corresponder).
RA5	Navegación de categorías	Dentro del formulario de un proyecto se debe poder seleccionar una categoría, al hacerlo se deben mostrar las preguntas correspondientes a la misma. También se debe auto-seleccionar la categoría siguiente a la actual una vez que se responden todas las preguntas de la categoría actual y se presiona el botón para validar el formulario. Se debe contar con un segundo botón que al presionar seleccione la categoría anterior a la actual. La categoría actualmente seleccionada debe ser visualmente distinguible.
RA6	Visualizar categorías completas	Dentro del formulario de un proyecto, si se responden todas las preguntas de una categoría, se presiona el botón validar, y el sistema no detecta nuevas preguntas (dentro de esa categoría), se considera que la categoría fue completada y debe presentarse esa información al usuario de forma visual con una marca distintiva.

		En caso de que al validar una categoría se generen nuevas preguntas en otra categoría ya completa, se debe quitar la marca visual de la misma.
RA7	Generar dimensiones y componentes	Una vez que se respondan todas las preguntas, el sistema debe analizar las respuestas ingresadas y generar las dimensiones y componentes de la arquitectura resultante. Una vez hecho esto el proyecto se considera finalizado y se bloquean las preguntas del formulario.
RA8	Ver y editar respuestas de un proyecto	Una vez que un proyecto finaliza y las respuestas del mismo se bloquean se debe mostrar un botón que permite editar las respuestas ingresadas. Una vez ingresadas las nuevas respuestas el usuario debe presionar el botón de validación lo cual analiza y valida las nuevas respuestas, en caso de que no se generen nuevas preguntas el sistema actualiza los componentes del proyecto.
RA9	Visualizar dimensiones	Una vez que el usuario finaliza el formulario del proyecto y el sistema genera sus dimensiones y componentes, se debe presentar un botón con el texto “Siguiente” que permite al usuario acceder a la lista de dimensiones del proyecto. Aquellas dimensiones que no cuenten con componentes deben identificarse con un color más tenue y no deben poder ser accedidas.
RA10	Visualizar componentes de una dimensión	Cuando el usuario se encuentra en el listado de dimensiones, este puede acceder a cualquiera de ellas y al hacerlo se le presentan los componentes del proyecto que fueron generados y que además se encuentran dentro de la dimensión seleccionada. Los componentes son presentados por tipo de componente (componente arquitectónico, patrón o tecnología), esta separación se realiza mediante pestañas.
RA11	Selección y de-selección de componente	Dentro del listado de componentes el usuario puede desmarcar el componente que desee, una vez hecho esto el mismo se muestra en un color más tenue. En caso de volver a marcarlo vuelve a su estado original.
RA12	Visualizar componentes recomendados	El sistema puede generar componentes que considere no son estrictamente necesarios para el proyecto pero aun así se recomiendan utilizar, en este caso los mismos deben mostrar un texto que indique que son componentes recomendados.
RA13	Visualizar incompatibilidades	El sistema puede presentar componentes que generen incompatibilidades entre sí, estos componentes deben presentar un texto en rojo que indique que generan una incompatibilidad.

RA14	Visualizar relación de incompatibilidad	Cuando un componente presenta una incompatibilidad se debe agregar sobre él, un icono de información. Al colocar el ratón sobre este icono se debe mostrar la lista de componentes con los cuales se generó la incompatibilidad.
RA15	Acceder a componente	Al acceder a un componente se debe mostrar una página con información sobre el mismo. Esta página puede contener texto e imágenes.
RA16	Visualizar fuentes de información	Al acceder a un componente se deben mostrar las distintas fuentes de información que este contiene junto con una breve descripción. Si la fuente de información es de tipo enlace debe presentar un botón que al presionar acceda al enlace en una nueva pestaña. Si la fuente de información es de tipo pdf debe presentar un botón que permite la descarga del mismo.
RA17	Versionado de modelo	El sistema debe poder convivir con más de una versión de un modelo de datos.
RA18	Aplicación responsive	La aplicación debe ser responsive para poder ser utilizada desde distintos dispositivos.
RA19	Facilidad de uso	La aplicación debe ser intuitiva y sencilla de utilizar.
RA20	Modelo extensible	Aunque no se cuente con una interfaz para realizar la funcionalidad, el sistema debe posibilitar la modificación y/o extensión del modelo de datos. El diseño debe permitir ingresar nuevos componentes con sus preguntas y respuestas correspondientes. Se debe poder añadir la lógica de decisión que estas generan sobre los componentes.

Figura 23: Tabla de requerimientos de Architecture Adviser.

11.2. Anexo 2: Diagramas de interacción

Este anexo contiene los diagramas de secuencia de los casos de uso “Responder y validar preguntas” y “Generar dimensiones y componentes” que muestran las interacciones y completan la vista lógica.

Responder y validar preguntas de un proyecto

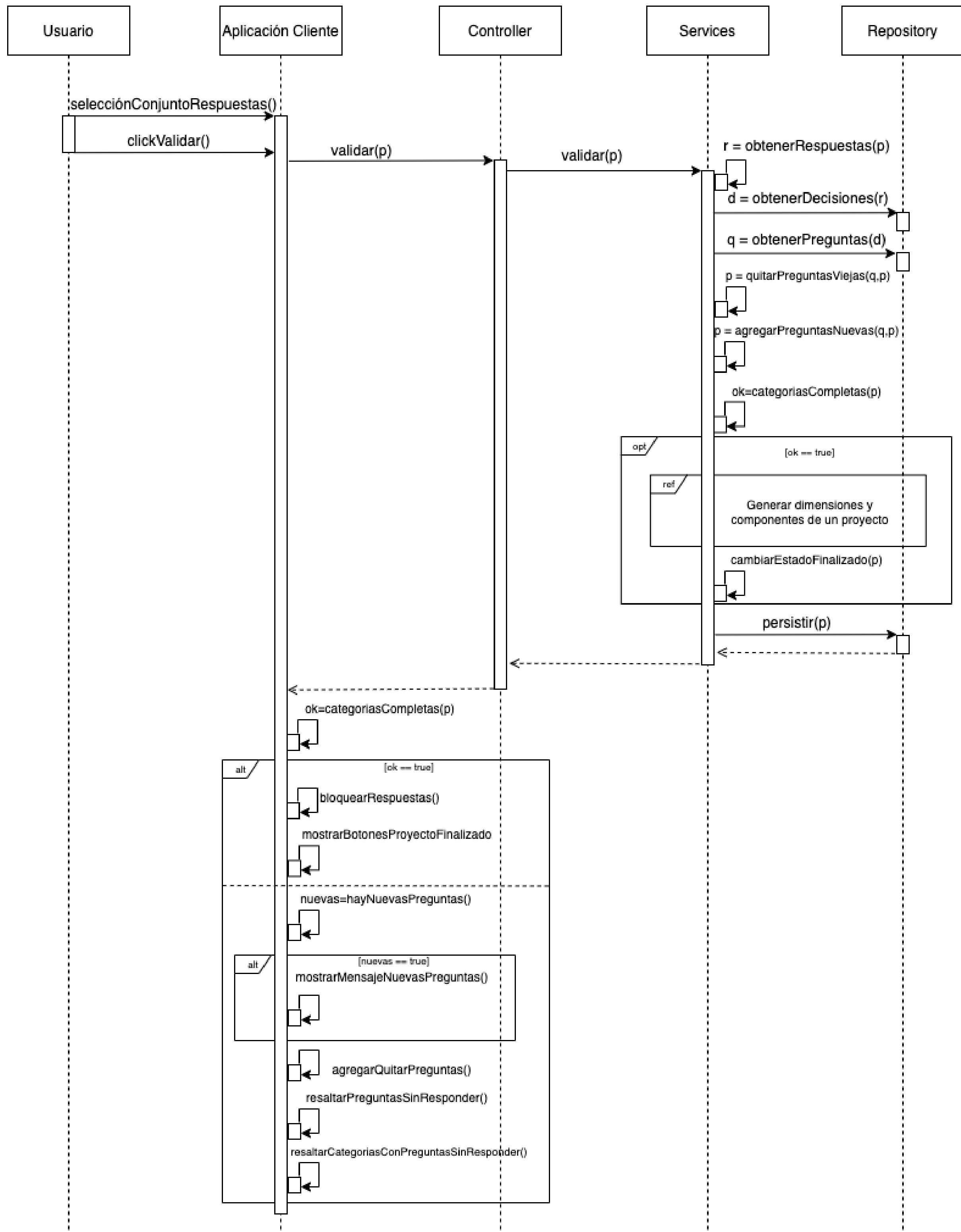


Figura 24. Diagrama de secuencias “Responder y validar preguntas de un proyecto”.

Como se puede ver en el diagrama de secuencia de la figura 24, el flujo comienza cuando el usuario decide responder un conjunto de preguntas de un proyecto y presiona el botón de *“Validar y Guardar”* por lo que comienza a ejecutarse el caso de uso en el componente *“Aplicación cliente”*, una vez que el componente recibe el proyecto con las respuesta se

comunica con el controlador encargado de recibir las solicitudes de validar preguntas de un proyecto, luego este se comunica con el service correspondiente para que valide las respuestas de las preguntas del proyecto. Cuando el service recibe una solicitud de validar las respuestas comienza por obtener las respuestas del proyecto, sigue por obtener las decisiones en base a las respuestas, y con las decisiones obtienen todas las preguntas del proyecto, una vez que se tiene todas las preguntas, el sistema quita las preguntas que no pertenecen al proyecto y agrega las nuevas, luego sigue por verificar si todas las preguntas de las categorías del proyecto tienen una respuesta, en ese caso se pasa a generar las dimensiones y componentes del proyecto para luego cambiar el estado del proyecto a *"Finalizado"*, por último el service se comunica con el repositorio encargado de persistencia de los proyectos en la base de datos. Una vez que el controlador recibe la respuesta de la solicitud, le retorna como respuesta el proyecto a el componente *"Aplicación cliente"*, este componente verifica si las categorias estan completas, en el caso que estén completas bloquea las respuestas de las preguntas para que no se puedan editar y muestra los botones correspondientes para editar el proyecto y visualizar las dimensiones. En caso que no estén completas el sistema verifica si hay preguntas nuevas, si las hay, muestra un mensaje notificando al usuario de que se generaron nuevas preguntas, luego sigue por actualizar las preguntas en cada categoría y por último resalta en rojo las preguntas que están sin responder.

Generar dimensiones y componentes de un proyecto.

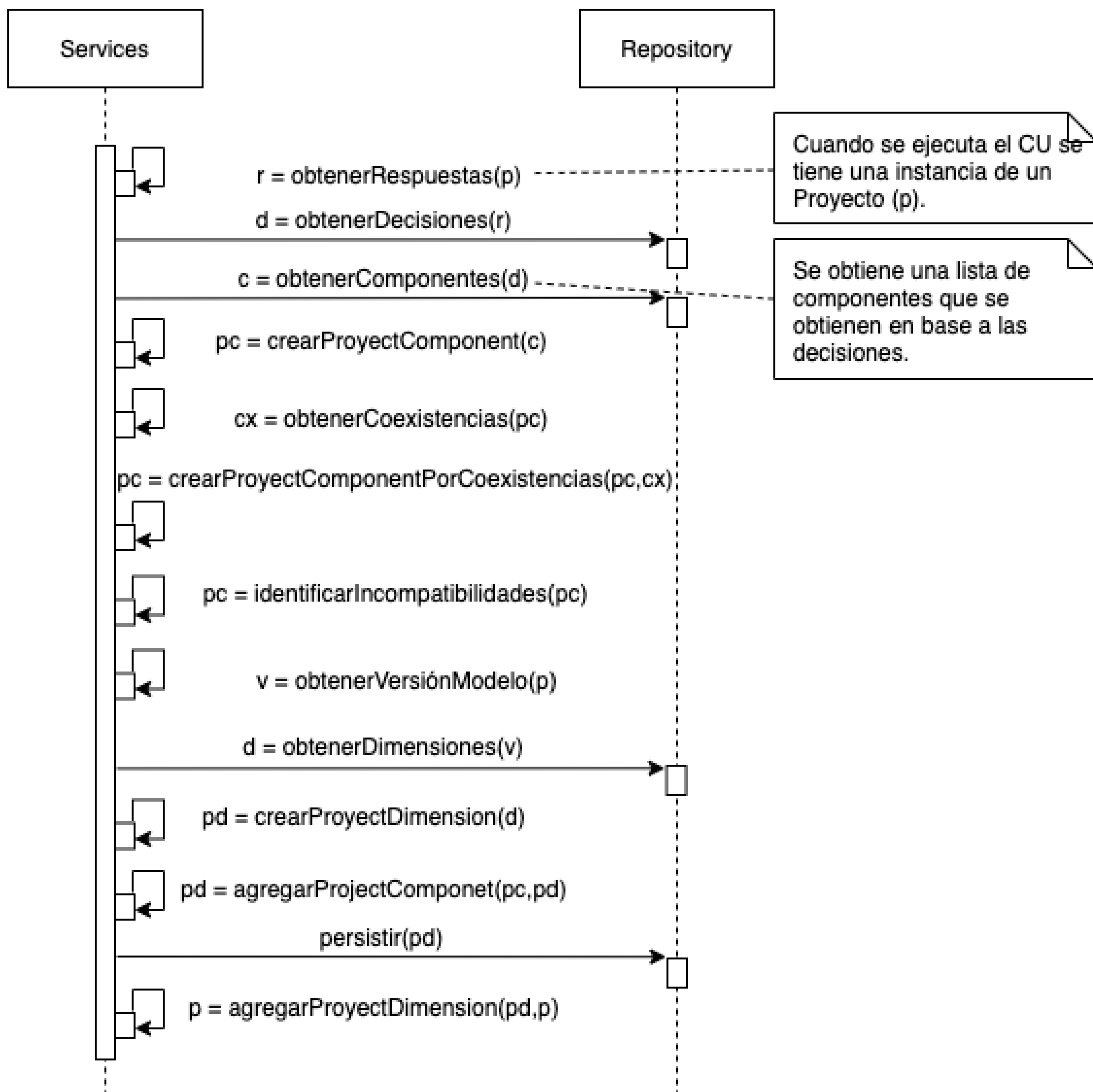


Figura 25. Diagrama de secuencias "Generar dimensiones y componentes de un proyecto".

Como se puede ver en el diagrama de secuencia de la figura 25, el flujo comienza cuando el service de proyectos recibe una solicitud para generar dimensiones y componentes de un proyecto para el cual se respondieron las preguntas de todas las categorías. Primero se comienza por crear los componentes y luego las dimensiones del proyecto. Para poder generar los componentes del proyecto lo primero que tiene que hacer el service es obtener las respuestas del proyecto, luego obtener las decisiones y luego obtener los componentes del modelo que surgen de las decisiones. Una vez que se tienen los componentes del modelo se crean los componentes del proyecto en base a las decisiones, luego el service debe crear los componentes del proyecto que se crean debido a coexistencias, para eso busca las coexistencias de cada componente del proyecto y a partir de ellas se crean los nuevos componentes que surjan. Por último el service busca e identifica si existe alguna incompatibilidad entre los componentes del proyecto creado. El flujo continúa por la creación de las dimensiones del proyecto, para eso se obtiene el modelo y sus dimensiones, a partir de ellas se crean las dimensiones del proyecto para luego asignar los componentes del proyecto a

cada dimensión. Una vez que se generaron las dimensiones y los componentes, el service se comunica con el repositorio encargado de persistir las dimensiones de un proyecto, para que persista en la base de datos. La funcionalidad finaliza cuando se agregan las dimensiones al proyecto para ser retornado con la dimensiones y componentes generados.

11.3. Anexo 3: Requerimientos fuera de alcance

A continuación en la figura 26, se presentan los requerimientos que quedaron fuera de alcance del proyecto.

Ref.	Nombre	Descripción
RB1	Ingreso al sistema	Deben existir dos tipos de usuario (usuario común y administrador) que pueden acceder al sistema ingresando nombre de usuario y contraseña
RB2	Gestión de usuarios	Un usuario de tipo administrador debe poder ver, crear, modificar y eliminar los usuarios del sistema.
RB3	Gestión de preguntas	Un usuario de tipo administrador debe poder ver, crear, modificar y eliminar las preguntas del sistema. Para cada pregunta se debe poder manejar un conjunto de respuestas asociadas. Para cada respuesta se debe poder definir un conjunto de componentes a los cuales afecta y de qué modo.
RB4	Gestión de dimensiones	Un usuario de tipo administrador debe poder ver, crear, modificar y eliminar las dimensiones del sistema. También debe poder definir qué componentes corresponden a cada dimensión.
RB5	Gestión de componentes	Un usuario de tipo administrador debe poder ver, crear, modificar y eliminar un componente. La eliminación de un componente debe ser de tipo lógico.
RB6	Gestión de fuentes de conocimiento	Un usuario de tipo administrador debe poder ver, crear, modificar y eliminar una fuente de información.
RB7	Ver proyectos realizados por usuarios	Un usuario de tipo administrador puede ver un historial de los proyectos realizados por los usuarios.
RB8	Gestión de categorías	Un usuario de tipo administrador debe poder ver, crear, modificar y eliminar una categoría y definir qué preguntas corresponden a ella.
RB9	Notificar solicitud de cambio de estado	Se debe notificar al administrador de las solicitudes de cambio de estado de una fuente de información que sean realizadas por los usuarios o la aplicación.

	para fuente de información	
RB10	Solicitud de cambio de estado automático	Debe haber un parámetro configurable que indique el tiempo de vida de una fuente de información, una vez que una fuente de información cumple este tiempo de vida, el sistema debe solicitarle al administrador un cambio de estado. De ser rechazado el tiempo de vida vuelve a iniciar.

Figura 26: Tabla de requerimientos de la aplicación backoffice de Architecture Adviser.