UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Ambiente para la Formalización, Animación y Evaluación de Patrones de Microservicios

Informe de Proyecto de Grado presentado por

Gastón Comba, Bruno Ravera

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisores

Laura González
Maximiliano Arcia
Sebastián Vergara

Montevideo, 13 de marzo de 2025

Resumen

El desarrollo de aplicaciones con arquitectura de microservicios ha ganado popularidad debido a sus ventajas en términos de escalabilidad y flexibilidad en el desarrollo, y mantenimiento. A pesar de estas ventajas, la arquitectura de microservicios presenta diversos desafíos, los cuales motivaron el surgimiento de patrones de microservicios. La descripción de estos patrones es en lenguaje natural, lo cual puede generar ambigüedades. Para solucionar esto se han propuesto soluciones basadas en la formalización de patrones.

Este trabajo extiende una de estas soluciones que utiliza Event-B como método para formalizar patrones. En particular, esta solución utiliza pruebas basadas en modelos para proveer mecanismos que permiten evaluar la conformidad de implementaciones de patrones de microservicios con respecto a la formalización de los patrones en Event-B.

Primero se analizó la solución en la que se basa este trabajo para evaluar la conformidad de las implementaciones de patrones de microservicios. Esta solución requiere gran cantidad de conocimiento previo y herramientas para su ejecución, lo cual es una barrera para que usuarios con diferentes perfiles la puedan utilizar.

Luego, se realizó una propuesta de una nueva solución, tomando en cuenta el análisis anterior, las necesidades de automatización para el sistema de evaluación de la conformidad de implementaciones de patrones y funcionalidades identificadas durante el análisis. La solución brinda un entorno con automatizaciones para la evaluación de la conformidad, y sirve como repositorio centralizado de formalizaciones, implementaciones, animaciones y reportes de pruebas de evaluaciones de conformidad.

Por último, se desarrolló una implementación de referencia de la solución propuesta. La plataforma está construida utilizando Java y Python para la lógica del backend, PostgreSQL para la gestión de datos, y Docker para permitir a los usuarios probar implementaciones en distintos lenguajes de programación. Para la interfaz de usuario se utilizó React.

La implementación de referencia permitió validar la factibilidad técnica de la solución. Además, mediante una instancia de validación con usuarios expertos se comprobó la utilidad de la propuesta.

Palabras clave: Microservicios, patrones de microservicios, Event-B, modelo formal, automatización de pruebas.

Índice general

Capítulo 1

Introducción

En este informe se presenta el proyecto de grado denominado "Ambiente para la Formalización, Animación y Evaluación de Patrones de Microservicios", el cual se basa en la tesis de maestría de Sebastián Vergara (Vergara, 2021a). En este capítulo se presenta el contexto, motivación, objetivos, y aportes del proyecto, y la organización del documento.

1.1. Contexto y Motivación

En los últimos años, la arquitectura de microservicios ha ganado popularidad debido a su capacidad para permitir el desarrollo de aplicaciones escalables, flexibles y mantenibles. Esta arquitectura se caracteriza por basar el desarrollo aplicaciones en componentes pequeños, centralizados como servicios y con un conjunto de responsabilidades definidas, los cuales se comunican entre sí (Fowler, 2014). Sin embargo, la naturaleza distribuida de los microservicios introduce desafíos, por ejemplo, su desarrollo, prueba y despliegue.

Para facilitar la construcción de aplicaciones basadas en microservicios han surgido un gran números de patrones de microservicios. Por ejemplo, el Circuit Breaker y Service Registry son ampliamente utilizados para mejorar la resiliencia y la disponibilidad de los sistemas, respectivamente. No obstante, su implementación y validación son complejas y propensas a errores, debido a la ambigüedad en sus definiciones y en la gran cantidad de implementaciones de estos patrones.

En esta línea Vergara propuso en su tesis de maestría (Vergara, 2021a), titulada "Implementation Compliance of Microservices Architectural Patterns", una forma para abordar la ambigüedad en la definición de patrones de microservicios mediante su formalización, con el método Event-B (EventB, 2014). Sin embargo, dicha solución requiere una intervención manual significativa y depende de una interfaz técnica poco amigable para usuarios no especializados, donde también se necesitan diversas herramientas instaladas y poder de cómputo para poder realizar correctamente la evaluación de la conformidad de las implementaciones.

La motivación detrás de este proyecto es extender el trabajo de Vergara, utilizando su sistema de evaluación de conformidad, y construyendo una nueva plataforma capaz de automatizar la generación y ejecución de pruebas de patrones, con una interfaz de usuario amigable y accesible desde internet, que además funcione como repositorio de formalizaciones, implementaciones, animaciones y reportes de los patrones.

1.2. Objetivos

El objetivo general de este proyecto consiste en desarrollar una plataforma para la gestión de formalizaciones de patrones de microservicios, con el fin de ejecutar pruebas sobre sus implementaciones. Se propone una plataforma que permite a los usuarios gestionar las formalizaciones de cada patrón, para luego ejecutar pruebas sobre sus implementaciones.

Se propone que la plataforma actúe como un repositorio centralizado para los patrones de microservicios, facilitando su documentación y permitiendo múltiples formalizaciones para cada patrón. Esto busca proporcionar a los usuarios una plataforma centralizada donde se pueda comparar distintas implementaciones del mismo patrón, evaluando cuál se adhiere más a distintas formalizaciones. De este modo, el sistema no solo automatizará las pruebas de conformidad, sino que también servirá como una herramienta de análisis, permitiendo seleccionar las implementaciones adecuadas según los resultados obtenidos. Para cumplir con el objetivo general, se plantean los siguientes objetivos específicos:

- **Entender la problemática:** Estudio de temas vinculados, identificando los desafíos asociados a la prueba y gestión de implementaciones de microservicios basados en formalizaciones.
- **Analizar los requerimientos:** Definir los requisitos funcionales y no funcionales del sistema para contemplar las necesidades de los usuarios y los objetivos del proyecto.
- **Proponer y diseñar una solución conceptual:** Diseñar una arquitectura conceptual que describa cómo se gestionarían las formalizaciones, implementaciones y pruebas dentro del sistema.
- **Implementar el sistema:** Desarrollar el sistema siguiendo la solución conceptual definida en el punto anterior, seleccionando tecnologías y herramientas adecuadas para la solución.
- **Validar la solución:** Realizar instancias de validación para conocer si la solución propuesta cumple con las necesidades de potenciales usuarios.
- **Documentar y presentar los resultados:** Generar documentación detallada del sistema, incluyendo manuales de usuario, y presentar los resultados del proyecto a los interesados.

1.3. Aportes

Los contribuciones más significativas de este proyecto son las siguientes:

- Se definieron los requisitos funcionales y no funcionales del sistema.
- Se diseñó una solución conceptual que permite la gestión de formalizaciones, implementaciones y pruebas, de forma integrada. El diseño contempló principios de modularidad y reutilización, apuntando a promover la extensibilidad del sistema.
- Se desarrolló un sistema funcional que permite la gestión centralizada de patrones de microservicios, facilitando su documentación y comparación.
- Se validó el sistema con un grupo de expertos en el área, que permitió evaluar en qué medida se cubren las necesidades de potenciales usuarios.
- Se elaboró documentación del sistema, incluyendo manuales de usuario y guías de implementación, facilitando su adopción y uso.

1.4. Organización del documento

El resto del documento se organiza de la siguiente manera.

En el Capítulo 2 se presenta el marco teórico, explicando los principales conceptos sobre los que se basa este trabajo.

En el Capítulo 3 se detalla el análisis previo a la implementación del proyecto, describiendo la solución actual, y detallando los casos de uso, casos de estudio y escenarios motivadores del presente trabajo.

En el Capítulo 4 se presenta la arquitectura de la solución propuesta en el presente trabajo, de forma genérica, agnóstica a la tecnología, y se detalla el funcionamiento de los flujos principales.

En el Capítulo 5 se presenta el detalle de la implementación del sistema, describiendo sus módulos y tecnologías.

En el Capítulo 6 se presenta cómo se validan y verifican las soluciones propuestas.

Finalmente, en el Capítulo 7 se presentan las conclusiones del proyecto, sus contribuciones, y posibles mejoras o extensiones a futuro.

Capítulo 2

Marco Teórico

En este capítulo se presentan los conceptos necesarios para el entendimiento del trabajo realizado.

2.1. Microservicios

En esta sección se presenta una descripción general de microservicios y se describen dos patrones de microservicios relevantes para la tesis.

2.1.1. Descripción general

La arquitectura de microservicios, a pesar de ser globalmente utilizada y contar con gran popularidad en la actualidad, no tiene un definición formal aceptada. Según Chris Richardson en (Richardson, 2018), una arquitectura de microservicios a alto nivel es un estilo arquitectónico que descompone funcionalmente una aplicación en un conjunto de servicios. Para Fowler y Lewis (Fowler, 2014), aunque no existe una definición precisa de este estilo arquitectónico, hay ciertas características comunes, como por ejemplo: la organización basada en capacidades de negocio, el despliegue automatizado, la inteligencia en los *endpoints* y el control descentralizado de lenguajes y datos.

Según Richardson, la arquitectura de microservicios presenta los siguientes beneficios (Richardson, 2018):

- Permite la entrega y el despliegue continuo de aplicaciones grandes y complejas.
- Los servicios son pequeños y fáciles de mantener.
- Los servicios se pueden desplegar de manera independiente.
- Los servicios son escalables de forma independiente.
- Permite que los equipos de trabajo sean autónomos.

- Facilita la experimentación y la adopción de nuevas tecnologías.
- Tiene mejor aislamiento de fallos.

Por otro lado, Richardson también identifica las siguientes desventajas (Richardson, 2018):

- Encontrar el conjunto adecuado de servicios es un desafío.
- Los sistemas distribuidos son complejos, lo que hace que el desarrollo, las pruebas y el despliegue sean difíciles.
- Desplegar características que abarcan múltiples servicios requiere una coordinación cuidadosa.
- Decidir cuándo adoptar la arquitectura de microservicios puede no ser trivial.

Para facilitar el desarrollo de aplicaciones basadas en este estilo arquitectónico varios autores han propuesto patrones de microservicios. Un patrón es una solución reutilizable para un problema común que surge en un contexto específico. En el caso particular de la arquitectura de microservicios, un patrón representa una solución concreta para un problema específico dentro de este modelo arquitectónico (Richardson, 2018). En las siguientes subsecciones se presentan los patrones de arquitectura de microservicios Circuit Breaker y Service Registry, ya que son los patrones que se tomaron como ejemplo para la realización del trabajo.

2.1.2. Circuit Breaker

Cuando un servicio invoca sincrónicamente a otro, existe la posibilidad de que el otro servicio no esté disponible o tenga una latencia tan alta que sea inutilizable. Esto puede consumir recursos valiosos, como hilos, en el servicio que llama, lo que podría llevar a un agotamiento de recursos e impedir que maneje otras solicitudes. El fallo de un servicio podría propagarse a otros en toda la aplicación (Chris Richardson, 2024a).

Para evitar que un fallo en la red o en un servicio afecte a otros servicios, el cliente debe invocar el servicio remoto a través de un proxy que funcione como un interruptor de circuito. Si el número de fallos consecutivos supera un umbral, el interruptor se dispara y, durante un tiempo determinado, todos los intentos de invocar el servicio fallarán de inmediato. Tras el período de espera, se permiten un número limitado de solicitudes de prueba. Si tienen éxito, el interruptor vuelve a la operación normal; de lo contrario, el tiempo de espera comienza nuevamente. En la figura 2.1 se presenta el diagrama de Circuit Breaker.

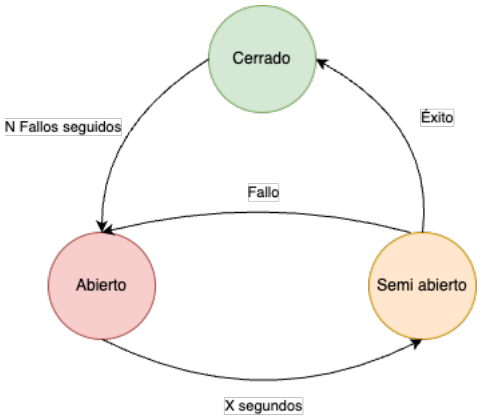

Figura 2.1: Diagrama de Circuit Breaker (adaptación de (Redhat, 2024))

Una de las implementaciones del patrón Circuit Breaker está dada por *Hystrix*, desarrollada por Netflix en tecnología Java, que es "es una biblioteca de latencia y tolerancia a fallos diseñada para aislar puntos de acceso a sistemas remotos, servicios y bibliotecas de terceros, detener fallos en cascada y habilitar la resiliencia en sistemas distribuidos complejos donde el fallo es inevitable" (Netflix, 2024b). Actualmente no se encuentra en desarrollo activo pero sigue en fase de mantenimiento.

Otra implementación del Circuit Breaker está dada por *Resilience4j* que es una biblioteca ligera desarrollada en Java , la cual implementa funcionalidades como Circuit Breaker, Rate Limiter, Retry y Bulkhead (en forma de decoradores), las cuales pueden combinarse según las necesidades específicas de la aplicación. Una de sus principales ventajas es que permite seleccionar únicamente los decoradores necesarios, optimizando así el rendimiento y reduciendo la complejidad del sistema (resilience4j, 2024).

2.1.3. Service Registry

Cuando los clientes de un servicio quieren encontrar la localización de una instancia a la cual enviar peticiones, usan "Client-side discovery" o "Server-side discovery", lo cual genera el problema de saber cuáles instancias están disponibles. En el caso de "Client-side discovery" el cliente está encargado de encontrar la instancia a la cual enviar peticiones y en "Server-side discovery" el servidor es el encargado de resolver este problema. Para esto se implementa un service registry, que es una base de datos de servicios con sus instancias y ubicaciones. Las instancias se registran al iniciar y se desregistran al terminar su ejecución. De esta forma los clientes o enrutadores pueden consultar dicha base para encontrar servicios disponibles (Chris Richardson, 2024b).

Un ejemplo de implementación de este patrón es *Eureka*, que es un servi-

cio RESTful[1] diseñado principalmente para entornos en la nube. Su propósito principal es facilitar el descubrimiento de servicios ("Client-side discovery"), el balanceo de carga y la conmutación por error (asegurar la continuidad del sistema al redireccionar el tráfico a otra instancia del servicio que esta fallando) en el middleware de las aplicaciones. Este servicio es especialmente importante en la infraestructura utilizada por Netflix para garantizar el correcto funcionamiento de sus sistemas distribuidos (Netflix, 2024a). Eureka está desarrollada en Java y sigue siendo mantenida en la actualidad. En la figura 2.2 se presenta el diagrama de Service Registry para el caso de "Cient-side discovery", el cual es el caso de *Eureka*.

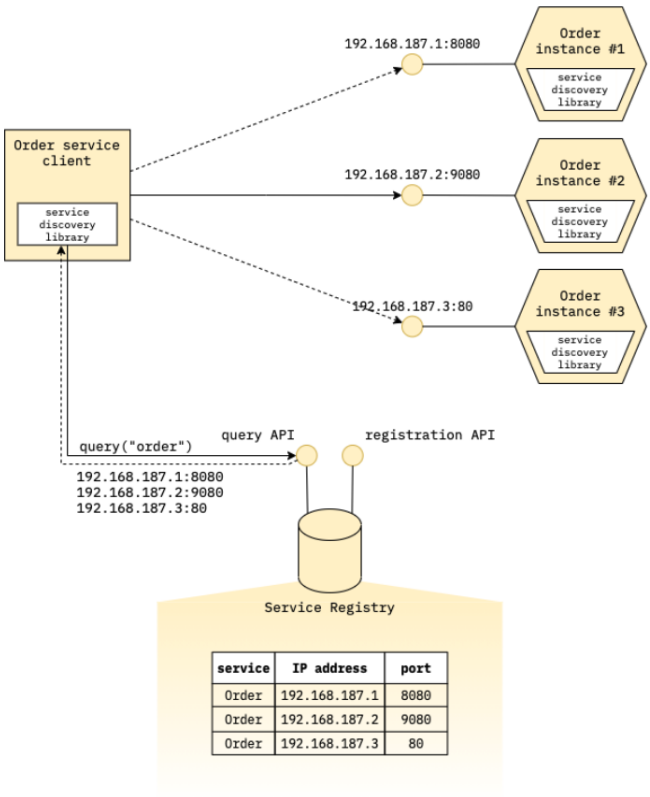

Figura 2.2: Diagrama de Service Registry "Client-side discovery" (Vergara, 2021a)

[1]Transferencia de Estado Representacional

2.2. Formalizaciones en Event-B

En esta sección se presentan se presenta el método Event-B para formalizar sistemas y algunas herramientas asociadas.

2.2.1. Descripción general

Event-B es un método de modelado para formalizar y desarrollar sistemas cuyos componentes pueden ser modelados como un sistema de transiciones discreto. Es una evolución del método B (CLEARSY safety solutions designer, 2024) clásico pero centrado en eventos, como también lo hacen otros modelos formales (Romanovsky y Thomas, 2013).

Las dos piezas fundamentales del método son los contextos y las máquinas. Los contextos representan la parte estática del sistema, incluyendo los parámetros y sus propiedades, mientras que las máquinas representan la parte dinámica del sistema. El sistema de transiciones, donde el estado está determinado por un conjunto de variables, y sus transiciones modeladas por un conjunto de eventos que incluyen una guarda, la cual es una condición que se debe cumplir para que un evento sea ejecutado. Una de las características más importantes de Event-B es que se basa en la teoría de conjuntos para modelar los sistemas, los cuales se pueden refinar en distintas capas de abstracción. Se pueden además realizar pruebas matemáticas para asegurar la coherencia entre dichas capas (EventB, 2014).

2.2.2. ProB

ProB es una herramienta para generar animaciones, resolver restricciones y verificar modelos para el método B (CLEARSY safety solutions designer, 2024) (Heinrich-Heine-University, 2024a). Puede ser utilizado para generar animaciones, encontrar modelos y la generación de casos de pruebas, entre otras cosas. En particular es de interés su capacidad de generación de casos de prueba, ya que es utilizado por Vergara, y en este trabajo. La generación de casos de prueba es el proceso de generar pruebas para un sistema en particular, en este caso a partir de un modelo formal. Esta técnica se utiliza en el enfoque *Model-based Testing (MBT)*. ProB provee dos algoritmos principales para la generación de pruebas (Heinrich-Heine-University, 2024b):

- **Basado en la verificación del modelo:** donde se utiliza el verificador del modelo para generar un grafo del mismo, desde su estado inicial, hasta que se cumpla cierto criterio de cobertura, utilizando un mecanismo de búsqueda en amplitud para la búsqueda de casos de prueba.

- **Basado en restricciones:** donde el solucionador se utiliza para generar posibles secuencias de eventos, también por amplitud, hasta que se cumpla cierto criterio de cobertura. Este método explora posibles caminos de ejecución.

2.3. Jupyter Notebook

Jupyter Notebook (Project Jupyter, 2024) es una aplicación simplificada para la creación de *notebooks*. Pertenece al proyecto Jupyter, el cual tiene como objetivo proveer herramientas y estándares para la computación interactiva con *notebooks* computacionales.

Una *notebook* computacional es un documento colaborativo que combina código computacional, descripciones en lenguaje natural, datos y diferentes tipos de visualizaciones como modelos en 3D, gráficos, diagramas y controles interactivos. Jupyer Notebook brinda un entorno para edición y ejecución de código de manera rápida, y también visualización de datos.

2.4. Docker

Docker (Docker Inc., 2024) es una plataforma abierta para desarrollar, desplegar y correr aplicaciones que permite separar las aplicaciones de la infraestructura, y así manejar la infraestructura de la misma forma que una aplicación. Para esto se utilizan contenedores, que son procesos aislados para cada componente de la aplicación, lo que permite correr, por ejemplo un frontend y un backend, en ambientes aislados a todo lo demás que se encuentre en la computadora que lo contenga. Utilizar contenedores de Docker, tiene varias ventajas, como por ejemplo:

- Auto contenidos, es decir, cada contenedor tiene todo lo que necesita para funcionar sin necesitar de ningún componente pre-instalado en la computadora que lo contenga.
- Aislados, como los contenedores corren en procesos aislados tienen una influencia mínima en la computadora donde corre y en otros contenedores, por lo tanto se obtiene más seguridad en la aplicación.
- Independientes, cada contenedor se maneja de manera independiente.
- Portables, los contenedores pueden correr en casi cualquier computadora y van a correr de la misma manera en cualquier computadora que lo contenga.

Esto a diferencia de una máquina virtual representa una gran ventaja, ya que estas necesitan de una cantidad grande de recursos como programas, aplicaciones y hardware, entre otras, mientras que los contenedores pueden compartir estos mismos recursos sin grandes complicaciones.

Generalmente estos contenedores están definidos en imágenes, que son una manera estandarizada de empaquetar archivos, binarios, librerías y configuraciones que se necesitan para correr el contenedor. Una de las dos grandes características de las imágenes es la inmutabilidad: una vez que esta creada no se puede modificar, es decir, si se necesita cambiar una imagen, es necesario crear una nueva o agregar cambios sobre la imagen. La otra característica es que las

imágenes están creadas por capas y cada una de las capas representa un conjunto de cambios al sistema de archivos que pueden agregar, borrar o actualizar archivos.

2.5. Servicios en la nube

En esta sección se describen servicios de la nube que fueron utilizados en el desarrollo de este proyecto.

2.5.1. Elastic Cloud

Elastic Cloud es una plataforma PaaS(*Plataform as a Service*) perteneciente Antel, que provee servicios en la nube para sistemas informáticos (Antel, 2024a). Cuenta con herramientas para el despliegue, configuración, escalamiento y operación de dichos sistemas, brindando soluciones preconfiguradas. Cuenta con servicios como servidores, por ejemplo de Java o Python, servicios de almacenamiento no estructurado, como OwnCloud, entre otros, permitiendo tener estos recursos funcionales de manera integral en la nube en pocos pasos.

2.5.2. Virtual Private Server

ANTEL provee también servicios de Virtual Private Server (VPS), que según su propia web es una partición virtualizada ejecutando sobre un servidor físico, con su propio sistema operativo y recursos, la que se ejecuta de forma asilada (Antel, 2024c). Esto es de gran utilidad ya que permite tener posibilidad de cómputo remoto, sin depender de ningún equipo local lo cual es fundamental para poder alojar servicios.

2.5.3. Dockerhub

Tal y como dice su web "Docker Hub es un registro de contenedores creado para que los desarrolladores y contribuyentes de código abierto encuentren, utilicen y compartan sus imágenes de contenedores. Con Hub, los desarrolladores pueden alojar repositorios públicos que se pueden usar de forma gratuita o repositorios privados para equipos y empresas" (Docker, 2024). Esto es de gran ayuda ya que permite compartir de manera muy sencilla aplicaciones sin tener que hacer grandes transferencias de archivos entre las partes, al ser un repositorio centralizado, con solo el nombre de la imagen y una etiqueta para indicar la versión, se puede disponer de cualquier aplicación pública o privada (si se tienen las credenciales correspondientes) que allí se encuentre.

Capítulo 3

Análisis

En este capítulo se presenta el análisis realizado previo al desarrollo de la solución. Se explica el proyecto de maestría en el que se basa esta tesis, y luego se describen los casos de uso que guiaron el diseño del sistema, capturando las funciones requeridas por los usuarios para la gestión de patrones de microservicios, sus implementaciones, animaciones y formalizaciones, así como la ejecución y gestión de pruebas.

Se incluyen prototipos y explicaciones de la solución a construir. Los prototipos son representaciones visuales preliminares de la interfaz de usuario, utilizadas para obtener retroalimentación, apuntando a que el diseño satisfaga las necesidades de los usuarios.

Finalmente, se presentan ejemplos de casos de estudio y escenarios motivadores que ilustran cómo la solución a construir puede aplicarse en situaciones reales. Estos ejemplos ayudan a validar la funcionalidad del sistema para gestionar y probar implementaciones de patrones de microservicios.

3.1. Solución actual

Este trabajo se basa en la tesis de maestría de Vergara (Vergara, 2021a), titulada *Implementation Compliance of Microservices Architectural Patterns*. El enfoque de Vergara aborda la ambigüedad en la especificación de patrones de arquitectura de microservicios en lenguaje natural, mediante su formalización en Event-B, seguido de la validación de sus implementaciones. Como parte de su trabajo, se tomaron como casos de estudio los patrones Circuit Breaker y Service Registry, formalizándolos y evaluando diversas implementaciones disponibles para estos patrones. Los objetivos principales que se proponen en el trabajo son:

- Examinar y caracterizar los patrones arquitectónicos de microservicios existentes, ya sea en el ámbito de la literatura académica, gris o no académica.

- Definir un método de formalización basado en Event-B para patrones arquitectónicos de microservicios en lenguaje natural.
- Definir y especificar un mecanismo de prueba para evaluar el nivel de cumplimiento de diferentes implementaciones de patrones arquitectónicos de microservicios con respecto a un especificación formal en Event-B.
- Evaluar el enfoque de formalización y cumplimiento de algunos de los patrones arquitectónicos de microservicios más utilizados y sus correspondientes implementaciones.

Se enfoca el análisis específicamente en los últimos dos puntos ya que es la base para poder construir un nuevo sistema de evaluación de la conformidad de las implementaciones de patrones de microservicios. En particular, en la solución desarrollada por Vergara (Vergara, 2021a), los usuarios interactúan con la herramienta a través de Jupyter Notebooks (Project Jupyter, 2024). Esta herramienta permite generar y ejecutar pruebas sobre implementaciones específicas de patrones y sus formalizaciones en Event-B.

La figura 3.1 describe la arquitectura general del sistema, donde se observan los componentes:

- **Componente de formalización**: responsable de traducir el patrón de arquitectura en lenguaje natural, en una especificación formal. La salida de este componente es el patrón en el modelo formal Event-B. Es un componente HITL (Human in the loop), lo cual significa que requiere la intervención humana, dado que el usuario debe proveer el modelo formal utilizando Rodin (Event-b.org, 2024), con la correspondiente extensión de ProB, a lo cual el autor le llama el ambiente de modelado.
- **Componente de generación de pruebas**: responsable de derivar casos de prueba abstractos[1] del modelo formal. Estos son generados con un ejecutable en Python[2]. Este componente es la primera de las partes de la cual Vergara llama ambiente de conformidad.
- **Componente instanciador de casos de prueba abstractos**: responsable de transformar los casos de prueba abstractos en casos de prueba específicos[3] para una implementación del patrón. Está compuesto de un ejecutable en Java llamado *xml2Junit*[4]. La salida de este componente es un proyecto al cual se le denomina Wrapper (ver Anexo A), y debe ser completado por el usuario. Este componente también es parte del llamado ambiente de conformidad.

[1] Se denota como caso de prueba abstracto, a un caso de prueba de alto nivel sin proveer una implementación especifica, valores de entrada, ni resultados esperados.

[2] Ejecutable llamado *probcli_test_generator.py* el cual se encarga de generar una colección de pruebas en XML dado un patrón modelado en Event-B y un archivo de configuración. Ver Anexo B

[3] Se denota como un caso de prueba específico, a la instanciación de un caso de prueba abstracto, para determinado lenguaje, con sus atributos y resultados esperados definidos.

[4] Ejecutable que recibe como entrada un archivo de configuración en formato YAML (ver Anexo C) y utilizando Maven, FreeMarker y JUnit, devuelve los casos de prueba específicos

- **Componente de validación de conformidad**: Es el último componente del ambiente de conformidad, el cual es responsable de completar el proyecto Wrapper generado en el paso anterior, para luego ejecutar los casos de prueba específicos contra la implementación del patrón. Como salida genera un reporte de conformidad, por lo tanto también es un componente HITL. El usuario es el encargado de ejecutar lo mencionado anteriormente manualmente. Una vez de ejecutadas las pruebas, se puede obtener el índice de conformidad calculado según la fórmula 3.1.

$$CI = \frac{\texttt{número de test aprobados}}{\texttt{número de test totales}} \times 100 \tag{3.1}$$

En síntesis se observa que dada una formalización se genera un modelo formal, con el que el sistema genera casos de pruebas abstractos, los cuales se instancian para una determinada implementación del patrón y la generación de pruebas se realiza con ProB, como se describe en "E.1 ProB Test Case Generation" del trabajo de Vergara. Esto permite luego ejecutar las pruebas correspondientes sobre una implementación y poder calcular el índice de conformidad con los resultados obtenidos.

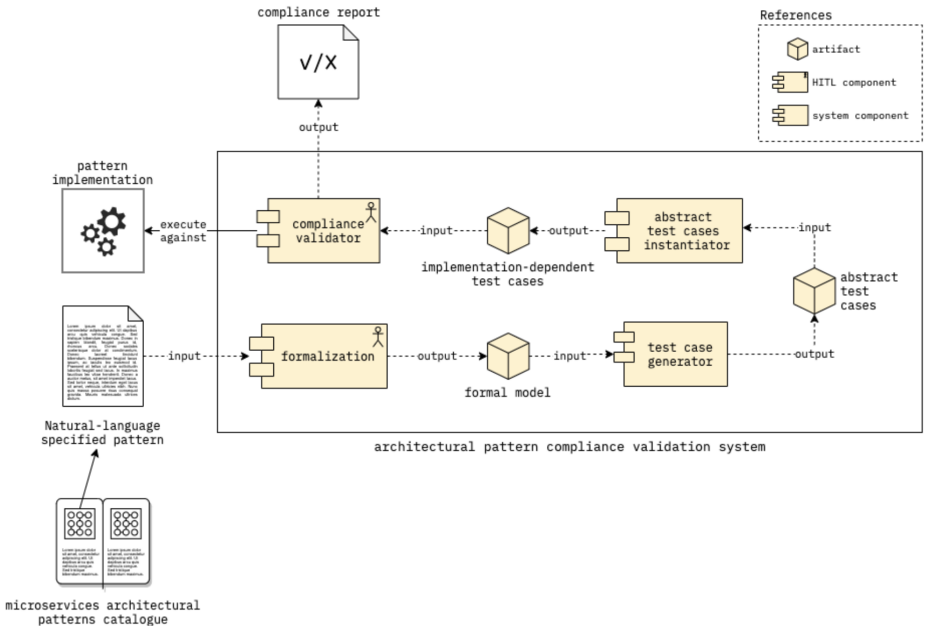

Figura 3.1: Diagrama general de arquitectura (Vergara, 2021a).

3.2. Presentación de casos de uso

El objetivo de esta sección es presentar los casos de usos relevantes que se identificaron en la etapa de análisis de la solución actual, con el propósito de extenderla a modo de que sirva como repositorio de patrones de arquitectura de microservicios y que automatice algunos pasos del sistema para obtener mayor facilidad de uso del sistema de verificación de conformidad.

En este contexto, la tesis aquí presentada busca extender y mejorar la solución propuesta por Vergara, incorporando varias mejoras y nuevas funcionalidades, entre las cuales se destacan:

- **Interfaz Gráfica Web Integrada**: En lugar de depender exclusivamente de Jupyter Notebooks, se propone una aplicación web que permita a los usuarios acceder de forma remota. Por lo tanto, se busca desarrollar una interfaz gráfica de usuario que facilite la realización de los casos de uso detallados en la siguiente sección.

- **Persistencia de Datos**: El sistema almacenará y gestionará los datos de las entidades y los reportes generados, lo cual facilita su reutilización y análisis histórico, sirviendo también como un repositorio centralizado que proporciona trazabilidad y consistencia en las pruebas realizadas.

Teniendo en cuenta las mejoras propuestas para la solución a construir, se procede a definir los casos de uso especificados a continuación. En una primera instancia se identificaron las funcionalidades básicas del sistema tomando como base el sistema actual, y se identifica, por ejemplo, que un usuario puede guardar un Patrón en el sistema, y operar sobre él. Este tipo de funcionalidades se desprenden de analizar el funcionamiento de la solución propuesta por Vergara (Vergara, 2021a), y el objetivo de este trabajo, teniendo así un conjunto de casos de uso que permiten satisfacer los objetivos planteados.

Los casos de uso vinculados a patrones son:

- **Crear Patrón**: Permite a los usuarios del sistema crear un nuevo Patrón, especificando su nombre, descripción y Formalizaciones.

- **Editar Patrón**: Permite a los usuarios del sistema editar el nombre y la descripción de un Patrón.

- **Eliminar Patrón**: Permite a los usuarios del sistema eliminar un Patrón, junto con sus datos relacionados.

- **Listar Patrones**: Permite a los usuarios del sistema listar los Patrones existentes.

- **Ver detalles de un Patrón**: Permite a los usuarios del sistema ver los detalles de un Patrón, lo cual incluye su nombre y descripción, sus Implementaciones, Animaciones y Formalizaciones.

Los casos de uso vinculados a implementaciones son:

- **Crear Implementación**: Permite a los usuarios del sistema crear una Implementación para un determinado Patrón, especificando su nombre, descripción, y su imagen en DockerHub.
- **Editar Implementación**: Permite a los usuarios del sistema editar el nombre, la descripción y la imagen de DockerHub de una Implementación.
- **Eliminar Implementación**: Permite a los usuarios eliminar una Implementación del sistema.
- **Listar Implementaciones de un Patrón**: Permite a los usuarios del sistema listar las Implementaciones existentes para un determinado Patrón.

Los casos de uso vinculados a animaciones son:

- **Crear Animación**: Permite a los usuarios del sistema crear una Animación para un determinado Patrón, especificando su nombre, descripción y archivo de animación.
- **Listar Animaciones de un Patrón**: Permite a los usuarios del sistema listar las Animaciones existentes para un determinado Patrón.
- **Ver Animación**: Permite a los usuarios del sistema visualizar una Animación.
- **Eliminar Animación**: Permite a los usuarios del sistema eliminar una Animación.

Los casos de uso vinculados a formalizaciones son:

- **Crear una Formalización**: Permite a los usuarios del sistema crear una Formalización para un determinado Patrón, seleccionando su archivo de Formalización.
- **Listar Formalizaciones de un Patrón**: Permite a los usuarios del sistema listar las Formalizaciones existentes para un determinado Patrón.
- **Descargar Formalización**: Permite a los usuarios del sistema descargar el archivo de formalización de una determinada Formalización.
- **Eliminar Formalización**: Permite a los usuarios del sistema eliminar una Formalización de un Patrón.

Los casos de uso vinculados a pruebas son:

- **Ejecutar Pruebas**: Permite a los usuarios del sistema ejecutar pruebas sobre una Implementación y una Formalización de un determinado Patrón.
- **Listar Reportes**: Permite a los usuarios del sistema listar los Reportes de las ejecuciones de pruebas de una Implementación determinada.

- **Ver detalles de Reporte**: Permite a los usuarios del sistema ver los detalles de un Reporte determinado.
- **Eliminar Reporte**: Permite a los usuarios del sistema eliminar un Reporte del sistema.

Como se puede apreciar en los casos de uso y en la figura 3.2, la entidad principal del sistema es el Patrón de microservicios. Esta entidad funciona como agrupador del resto de las entidades, dado que una Implementación, Formalización, Animación o Reporte existe en el contexto de un Patrón.

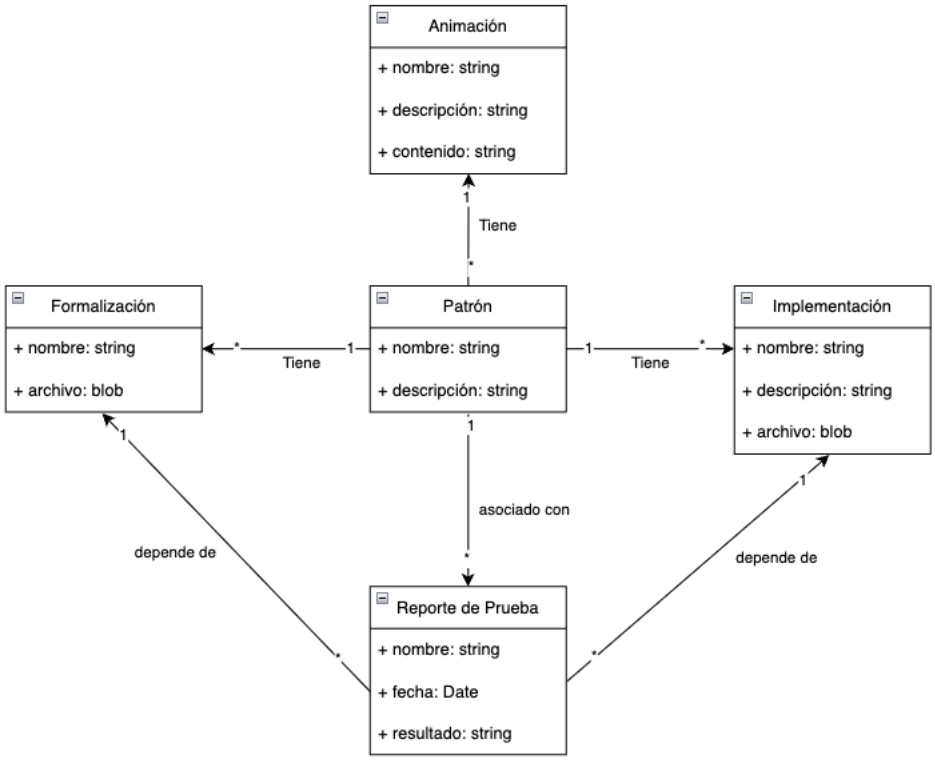

Figura 3.2: Modelo conceptual

Asimismo, tanto una Implementación como una Formalización no puede ser de más de un Patrón a la vez (una Implementación es una entidad que representa una implementación específica a nivel de código de un Patrón, y una Formalización es la definición formal de un Patrón de arquitectura de microservicios en una notación dada, en este caso Event-B). Una Animación es una entidad que representa una animación visual de un Patrón, y un Reporte es el resultado de

la ejecución de pruebas sobre una Implementación con una Formalización dada, por lo tanto está ligado a un Patrón implícitamente.

3.3. Prototipos

En esta sección se presentan los prototipos desarrollados para la aplicación, los cuales representan visualmente las interfaces de usuario. Estos prototipos fueron diseñados utilizando la herramienta Figma (Figma, 2024), y tienen el objetivo de obtener retroalimentación temprana y promover que el diseño cumpla con las expectativas y necesidades de los usuarios.

Los prototipos permiten una visualización preliminar de la interacción del usuario con el sistema, facilitando la identificación y corrección de posibles problemas de usabilidad antes de la implementación final. A continuación, se muestran los prototipos, detallando cada una de las pantallas y las principales características que ofrecen.

Se incluyen prototipos para las pantallas de listado de Patrones, y sus detalles, lo cual incluye la visualización de Animaciones, Implementaciones y Formalizaciones. Se omiten prototipos para los formularios de creación y edición de entidades, y listado y detalles de Reportes, dado que se entiende que su interfaz de usuario es simple y no impacta decisiones de modelado del sistema, ni afecta crear nuevos requerimientos del sistema.

3.3.1. Lista de Patrones

En esta pantalla se muestra la lista de Patrones disponibles en el sistema, como se puede ver en la figura 3.3, representados en una tabla que incluye el nombre y la descripción de cada uno, así como las acciones que se pueden realizar sobre ellos. Se puede ver el detalle, agregar una Implementación, editar y eliminar. El Patrón se identifica como la entidad principal y agrupadora del resto de las entidades del sistema. Por esta razón, la lista de Patrones es la pantalla principal del sistema, permitiendo a los usuarios tener un listado centralizado de Patrones. Esto facilita a los usuarios la selección de un Patrón de interés sobre el cual realizar operaciones.

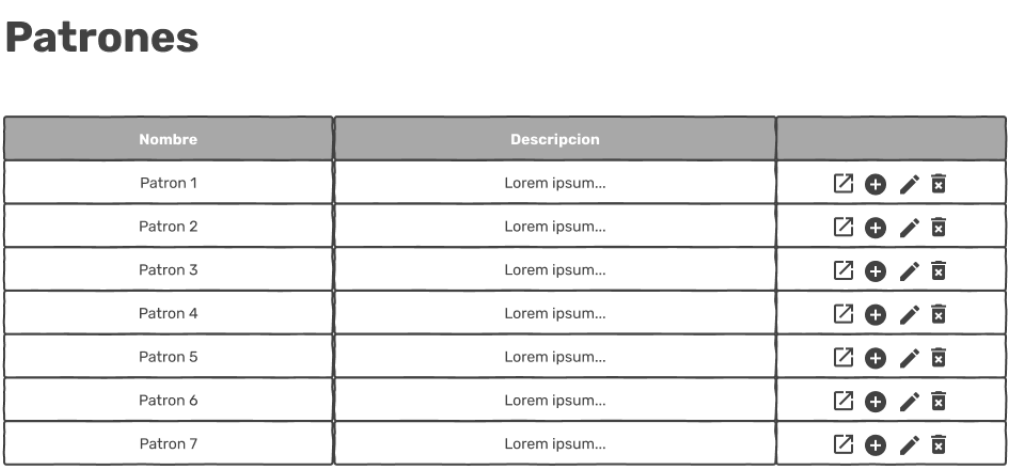

Figura 3.3: Prototipo de diseño - Lista de Patrones

3.3.2. Detalles de un Patrón

En esta pantalla se muestra el Detalle de un Patrón, como se puede ver en la figura 3.4. Cuenta con tres secciones principales: listado de implementaciones del Patrón seleccionado, con opciones de editar, ejecutar pruebas, ver reportes y eliminar la Implementación, listado de animaciones del Patrón seleccionado con opciones de visualizar o eliminar la Animación, y el listado de Formalizaciones del Patrón seleccionado, con opciones para descargar la Formalización o eliminarla.

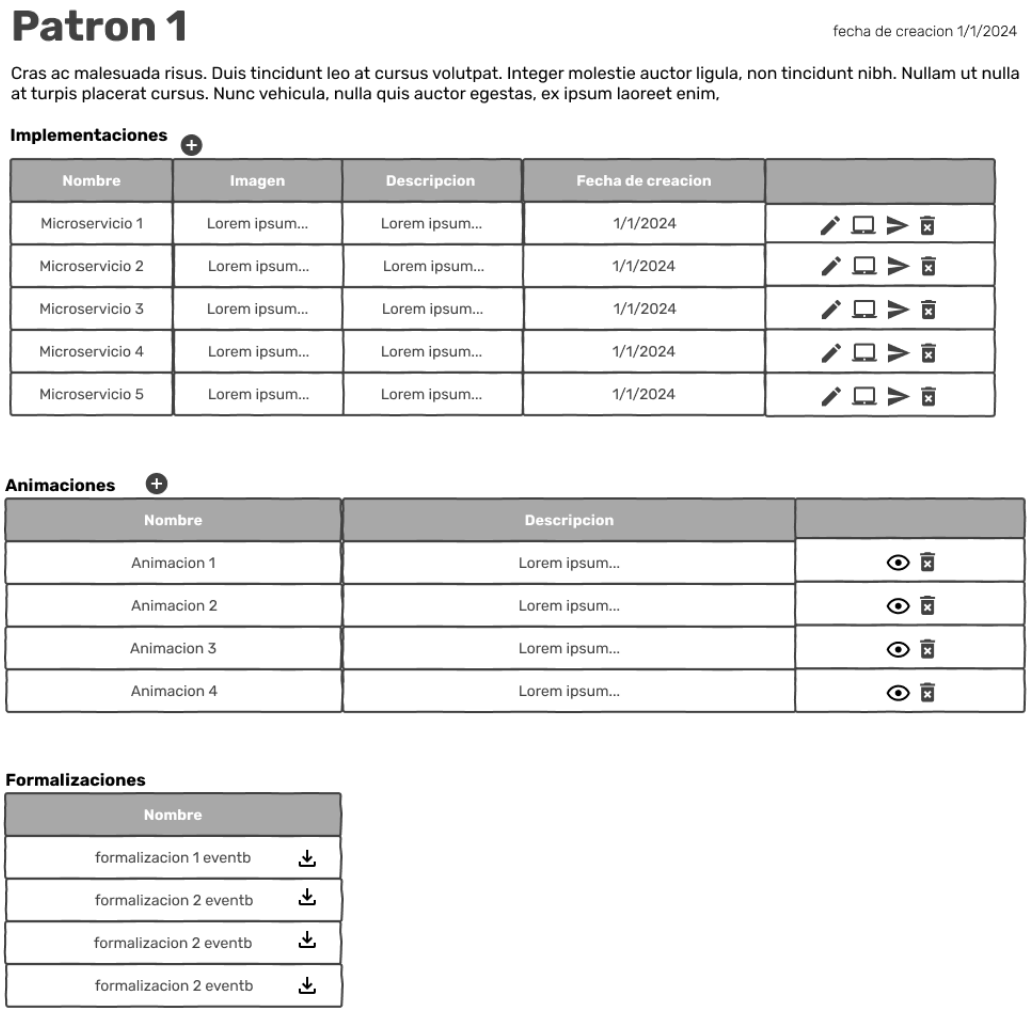

Figura 3.4: Prototipo de diseño - Detalles de un Patrón

3.4. Casos de estudio y escenarios motivadores

Este trabajo apunta a brindar una interfaz web integrada que permita la generación y ejecución de pruebas de patrones de microservicios, de acuerdo a lo propuesto por Vergara. El propósito principal de este trabajo es la generación y ejecución de pruebas sobre patrones de microservicios formalizados en notación Event-B, con una implementación específica determinada. Se toma como ejemplo el análisis realizado por Vergara (Vergara, 2021b) donde se generan y ejecutan pruebas del patrón de microservicios Circuit Breaker con la implemen-

tación de *Hystrix*.

A continuación se describe cómo se realizaría la generación y ejecución de pruebas para el patrón Circuit Breaker tomando como base los prototipos descritos en la Sección 3.3

- Se crea el patrón Circuit Breaker con su formalización en Event-B.
- Se accede a los detalles del patrón recientemente creado, seleccionándolo desde el listado de Patrones de la figura 3.3.
- Se le asocia la implementación de *Histrix*, desde el detalle de patrón, como se observa en la figura 3.4.
- Se procede a generar y ejecutar las pruebas desde la misma pantalla. Se obtienen los resultados correspondientes.

Teniendo en cuenta lo descrito anteriormente, se observa como escenario motivador los siguientes aspectos:

- Se logra la automatización de la carga de información. Una vez creada la entidad del Patrón Circuit Breaker con su formalización asociada en Event-B y su implementación, no es necesario repetir el proceso para correr pruebas nuevamente (y si se desea, con distintas configuraciones) sobre el mismo Patrón. Además, se pueden agregar más Formalizaciones e Implementaciones, manteniendo la persistencia de datos históricos.
- La nueva propuesta de solución descentraliza los datos al ser una aplicación web, permitiendo acceso remoto y facilitando la colaboración entre distintos usuarios.
- La capacidad de gestionar y comparar múltiples Implementaciones y Formalizaciones para un mismo Patrón en un solo lugar, lo cual busca facilitar la evaluación y selección de opciones.
- La persistencia de datos históricos permite revisar pruebas anteriores y seguir el progreso de las diversas implementaciones de los patrones a lo largo del tiempo.

3.5. Resumen

En este capítulo se presentó el detalle la solución actual, y los casos de uso motivadores que justifican el desarrollo de una nueva propuesta. Se presentaron prototipos y ejemplos que ilustran cómo se aplicaría la solución planteada. Los casos de uso y los prototipos desarrollados evidencian la importancia de una interfaz gráfica web integrada.

Capítulo 4

Solución planteada

En este capítulo se presenta la solución propuesta para la gestión y evaluación de patrones de microservicios. Se describen los distintos módulos que componen el sistema, así como las decisiones arquitectónicas tomadas. El enfoque se mantiene agnóstico a las tecnologías específicas, centrándose en una descripción teórica y conceptual de la arquitectura del sistema. Esto incluye la estructura de los componentes, las interacciones entre ellos y los principios de diseño en los que se basa la implementación.

4.1. Descripción de la solución

Se propone un sistema modular, ilustrado en la figura 4.1, donde se tienen los siguientes módulos y componentes:

- **Aplicación Web**: Este módulo es la interfaz principal del sistema, proporcionando una plataforma para que los usuarios interactúen con las diferentes funcionalidades del sistema. La aplicación web permite a los usuarios gestionar patrones de microservicios, generar y ejecutar pruebas, visualizar reportes y acceder a resultados históricos. Está construida para ser accesible de manera remota, lo que permite a los usuarios interactuar con el sistema desde cualquier navegador. Este módulo actúa como un punto central de comunicación entre los usuarios y los otros módulos del sistema, enviando solicitudes al Módulo Controlador y recibiendo respuestas para presentarlas de manera comprensible a los usuarios.

- **Módulo Controlador**: Este módulo consiste en una API central que maneja las solicitudes entrantes desde la Aplicación Web. Se denomina *Backend* a la API del Módulo Controlador. El *Backend* se encarga de delegar las tareas a los módulos correspondientes, como el procesamiento de pruebas y la ejecución de pruebas, y también es responsable de la gestión de la base de datos que almacena los datos estructurados del sistema. Aunque el *Backend* y la base de datos son componentes diferentes, se agrupan en

un solo módulo debido a que la base de datos es exclusivamente accesible a través del *Backend*.

- **Módulo de Procesamiento**: Este módulo consiste en una API encargada de la generación de pruebas para las formalizaciones de los patrones. El Módulo Controlador se comunica con este módulo con los datos necesarios para generar archivos los cuales serán utilizados para ejecutar pruebas. El módulo toma como base el componente de generación de pruebas y el instanciador de pruebas abstractas propuesto por Vergara, los cuales se encuentran utilizados por Vergara como se observa en la figura 3.1. Este módulo es de gran importancia ya que logra resumir en un solo conjunto de interacciones con el usuario, lo que antes se hacia en dos. De este modo, se minimiza la interacciones de los usuarios para obtener una mayor automatización.

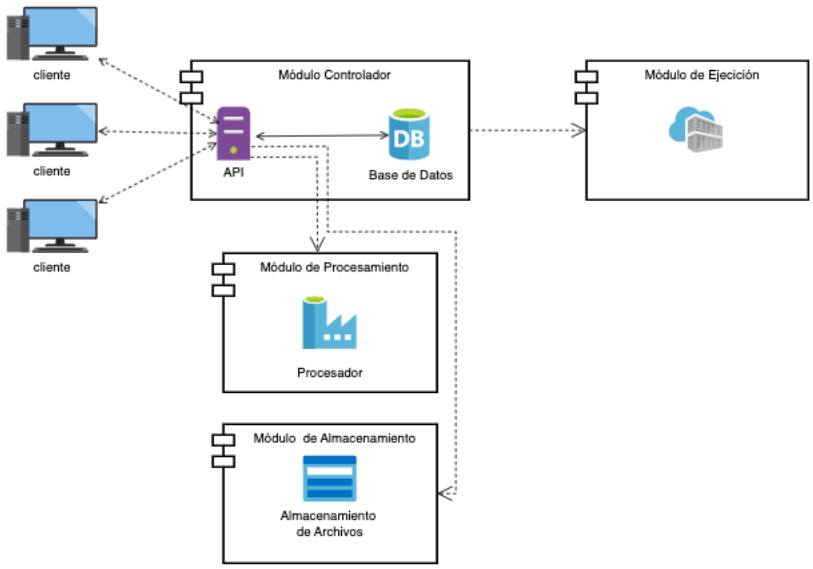

Figura 4.1: Solución Planteada - Diagrama general de arquitectura.

- **Módulo de Ejecución**: Este módulo consiste en una API encargada de ejecutar las pruebas generadas por el módulo de procesamiento, sobre una implementación específica del patrón. El equivalente en el sistema de verificación de conformidad presentado por Vergara sería parte del componente "compliance validator". Sigue siendo necesario un componente HITL debido a que el usuario tiene que completar la salida del módulo anterior para

luego ejecutarla, y la ejecución de las pruebas contra la implementación con su respectivo reporte. El módulo obtiene entradas desde el Módulo Controlador la referencia a un contenedor a descargar que contenga la implementación, que se encuentra en una plataforma de repositorio de contenedores, y una referencia al Wrapper completo, que se encuentra en el módulo de almacenamiento. Con las entradas, ejecutaría las pruebas del Wrapper completo contra la implementación que se encuentra en el contenedor y obteniendo como resultado el reporte de conformidad de la implementación para el patrón de arquitectura de microservicios.

- **Módulo de Almacenamiento**: Repositorio de datos no estructurados, para almacenar archivos de configuración, resultados intermedios, resultados de pruebas, entre otros.

4.2. Comparativo conceptual entre la Solución Actual y la Solución Planteada

En la sección 3.1 se presenta la solución actual y en la sección 4.1 se presenta el nuevo sistema, con sus módulos y responsabilidades. A continuación se traza un paralelismo entre ambos, mencionando sus equivalencias, diferencias, y toma de decisiones para la construcción del nuevo sistema. En la figura 4.2 se superponen algunos de los módulos de ambas soluciones, y se detalla a continuación.

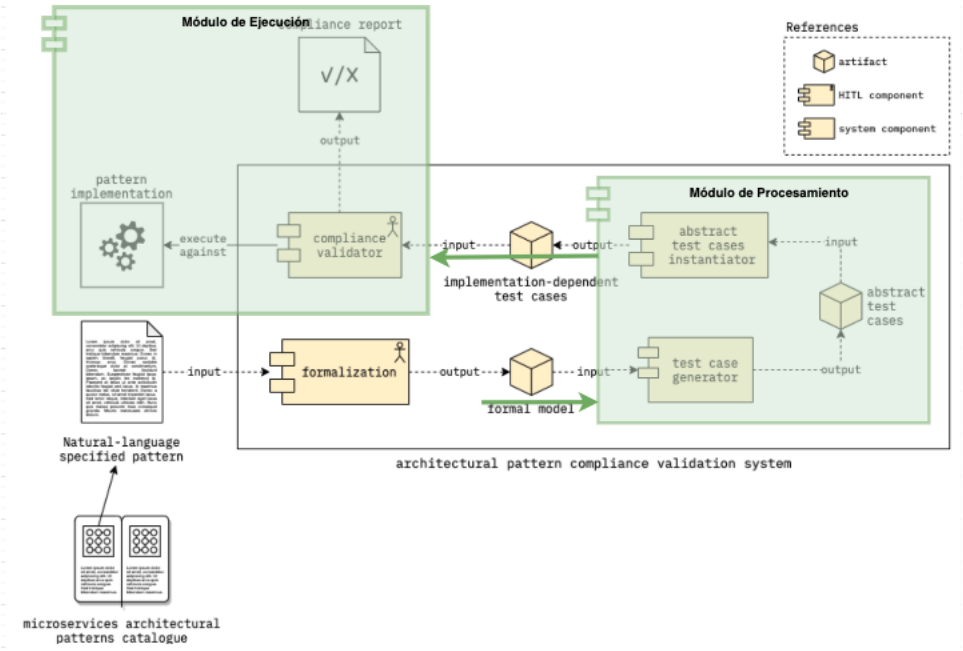

Figura 4.2: Equivalencias y comparativo entre soluciones.

Se observan las siguientes equivalencias y diferencias:

- El flujo de la Solución Actual comienza con una especificación en lenguaje natural, que debe ser formalizada por el usuario, es decir, transformada en un modelo formal. La Solución Propuesta comienza con el modelo formal dado.
- Los componentes *test case generator* y *abstract test case instantiator* de la Solución Actual se abstraen como un solo módulo denominado Módulo de Procesamiento en la Solución Planteada. Estos módulos son equivalentes, dado que, teniendo como entrada un modelo formal en Event-B, y estrategias de generación e instanciación de pruebas, la salida del Módulo de Procesamiento es equivalente a ejecutar ambos pasos con las mismas entradas en la Solución Actual.
- El componente *compilance validator* de la Solución Actual es equivalente al Módulo de Ejecución de la Solución Planteada, donde su propósito es recibir casos de prueba específicos, y generar un reporte de conformidad para una implementación específica.
- La Solución Propuesta cuenta con otros módulos, como se mencionó anteriormente, los cuales no tiene un paralelismo o equivalencia con la Solución

Actual. A continuación se pasa a detallar su propósito y motivo por el cual no hay tal equivalencia:

- Aplicación Web: Módulo de la Solución Propuesta el cual está encargado de servir como interfaz gráfica para el usuario, permitiendo la interacción con el sistema, sin necesidad de tener conocimientos técnicos, ni ningún software adicional instalado más que un navegador web. En la Solución Actual dicha interacción es mediante Jupyter Notebooks, y es necesario tener conocimientos técnicos, y ambiente local preparado para la ejecución.
- Módulo Controlador: Módulo de la Solución Propuesta, el cual, como se mencionó anteriormente consta de un *Backend* y una base de datos, los cuales no presentan correspondencia con la Solución Actual dado que:
  - El *Backend* es el encargado de manejar las peticiones del usuario a través de la Aplicación Web, e invocar los módulos correspondientes ante cada petición. En la Solución Actual esto no ocurre dado que el usuario ejecuta comandos en la Jupyter Notebook.
  - La base de datos almacena los datos ingresados por el usuario, los generados por el sistema, y todo dato necesario a modo de presentar datos históricos, o cualquier otra información útil para el usuario. En la Solución Actual esto no ocurre dado que no se persisten los datos, ya que queda a criterio del usuario si quiere almacenarlos en su propia computadora, o repositorio, de forma manual.
- Módulo de Almacenamiento: Módulo de la Solución Propuesta el cual, como se mencionó anteriormente, se encarga de almacenar datos no estructurados, como por ejemplo, formalizaciones en Event-B. Esto permite al sistema poder cargar este tipo de datos una vez sola al sistema, y reutilizarlos las veces que sea necesario. También permite compartir datos entre usuarios, ya que un usuario puede utilizar o acceder los datos de un Patrón generado por otro usuario, permitiendo así la colaboración. En la Solución Actual esto no ocurre, el usuario debe subir los archivos para cada ejecución, por lo tanto, no se puede reutilizar, y al ser ejecutado en ambiente local, tampoco permite la colaboración.

4.3. Flujo principal - Generación y Ejecución de pruebas

En la figura 4.3 se muestra un diagrama el cual representa el flujo de interacciones entre los módulos para la generación y ejecución de pruebas, el cual es el flujo principal del sistema. Por simplicidad del diagrama se omiten interacciones con el Módulo de Almacenamiento, en el cual se almacenan archivos de

configuración, y resultados de las pruebas. Se parte de la base que en el sistema existe un Patrón creado, con su Formalización e Implementación creadas. Se detallan las interacciones entre los módulos:

1. El usuario del sistema desea generar y ejecutar pruebas. La Aplicación Web solicita al usuario que especifique el Patrón, con su Formalización e Implementación sobre el cual proceder, junto con el archivo de estrategia. La Aplicación Web envía la petición al *Backend*.

2. El *Backend* recibe la solicitud, y solicita al Módulo de Procesamiento que genere los casos de prueba abstractos.

3. El Módulo de Procesamiento genera los casos de prueba abstractos, y genera el Wrapper genérico. Almacena dichos archivos en el Módulo de Almacenamiento y le responde a la API indicando la ubicación de los mismos en el Módulo de Almacenamiento.

4. El *Backend* responde a la Aplicación Web indicando la generación exitosa de los casos de prueba abstractos y el Wrapper.

5. El usuario completa el Wrapper genérico y lo envía al *Backend*. La Aplicación Web no espera una respuesta inmediata, sino que permite al usuario seguir utilizando la aplicación, ya que este proceso puede demorar unos minutos, y no se bloquea al usuario. Es por esto que se representa en el diagrama como una transición en ambos sentidos.

6. El *Backend* solicita a la API del Módulo de Ejecución que ejecute las pruebas sobre la Implementación dada, utilizando el Wrapper completo.

7. La API del Módulo de Ejecución le responde al *Backend* indicando el resultado de la ejecución de las pruebas.

8. El *Backend* notifica a la Aplicación Web la finalización de la ejecución de las pruebas, utilizando Web Sockets.

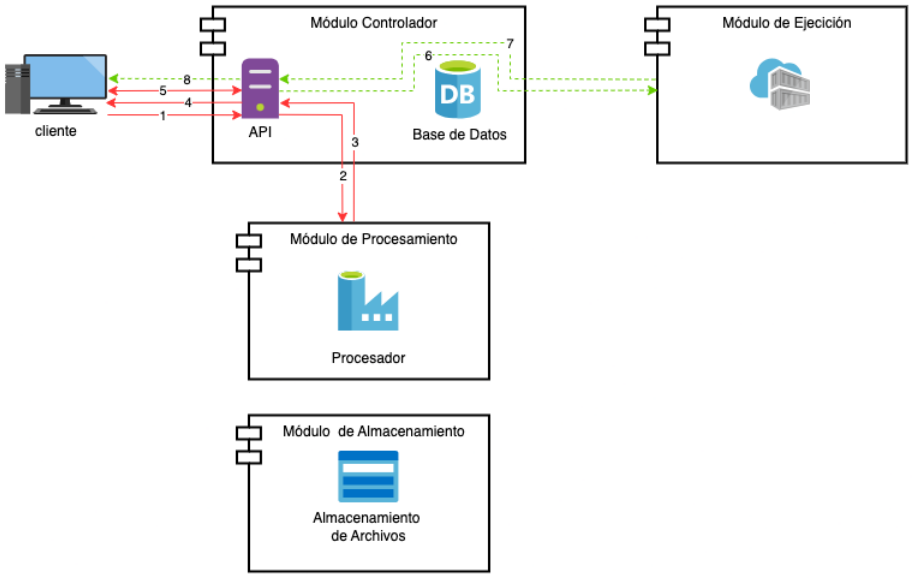

Figura 4.3: Flujo principal - Generación y Ejecución de pruebas.

4.4. Sistema modular y toma de decisiones

La arquitectura del sistema fue diagramada de forma modular teniendo en cuenta cambios a futuro, ya sean cambios en la operativa de los módulos, escalabilidad del sistema, forma de manejo o representación de datos, y lenguajes utilizados. Se tomaron las siguientes decisiones:

- **Aplicación Web**: Se optó por utilizar una aplicación web debido a la necesidad de un sistema accesible y disponible para los usuarios de forma remota. Las aplicaciones web, al ser accesibles a través de internet mediante un navegador, ofrecen una solución flexible y multiplataforma, permitiendo que los usuarios accedan desde distintos dispositivos sin la necesidad de instalar software adicional. Además, las aplicaciones web facilitan el mantenimiento y la actualización del sistema, ya que cualquier cambio se puede implementar en el servidor sin necesidad de intervención en los dispositivos de los usuarios.

- ***Backend***: Este módulo actúa como el orquestador del sistema, encargado de procesar las peticiones de los usuarios a través de la aplicación web. Es responsable de la creación, modificación y eliminación de entidades y es el único módulo con acceso directo a la base de datos, lo que garantiza la integridad de los datos. Además, este módulo interactúa con otros módulos para la generación y ejecución de pruebas. Las decisiones detrás de esta arquitectura incluyen:
  - Centralizar el flujo de información entre los componentes del sistema, apuntando a facilitar la gestión y monitoreo.
  - Centralizar el acceso a la base de datos, permitiendo la implementación de medidas de control de acceso y autenticación.
  - Delegar la responsabilidad de la generación y ejecución de pruebas a módulos especializados, apuntando a facilitar su implementación, reemplazo y escalabilidad.
  - Habilitar el escalamiento horizontal, permitiendo la distribución de la carga entre varias instancias del *Backend* a través de balanceadores de carga, con el fin de mejorar la disponibilidad y capacidad de respuesta del sistema.
- **Base de Datos Relacional**: El sistema utiliza una base de datos relacional para almacenar los datos estructurados, como las entidades principales del dominio (por ejemplo, patrones, implementaciones, formalizaciones, reportes) y las relaciones entre ellas. La elección de una base de datos relacional se basa en la necesidad de manejar datos con relaciones definidas y la posibilidad de realizar consultas de manera eficiente, y las bases de datos relacionales permiten la definición de restricciones y claves foráneas para garantizar la integridad en las referencias entre las entidades. Además, para los datos no estructurados, como archivos de configuración o resultados de pruebas, se utiliza un módulo de almacenamiento separado que se adapta mejor a este tipo de información, permitiendo una gestión flexible para ambos tipos de datos.
- **Módulo de Procesamiento**: Módulo encargado de la generación de casos de prueba abstractos, y los Wrapper genéricos ya mencionados. El Módulo se presenta como una API, y encapsula la lógica presentada por Vergara. Su principal objetivo es unir los pasos del *test case generator* y el *test case instantiator* en uno solo, en busca de poder automatizar lo más posible el proceso y reducir las interacciones con el usuario. Por lo tanto, se reutilizan las herramientas ya implementadas para la tesis de Vergara, como los son el "prob_cli_generator" (ejecutable Python) en conjunto con el "Custom Strategy" (proyecto Java) para el caso de generador de pruebas abstractas y de la herramienta "Xml2Junit" (proyecto Java) para el caso del instanciador de pruebas abstractas. Este módulo será utilizado por la API del Módulo Controlador para iniciar la generación de casos de prueba. Se opta por crear este módulo dado que:

- Centraliza la lógica de generación de casos de prueba, permitiendo extender, modificar o reemplazar esta lógica sin afectar el resto del sistema, lo que facilita la evolución del sistema.
- Facilita la escalabilidad del sistema al separar la generación de casos de prueba en un módulo independiente, dado que sería posible escalar este componente de manera autónoma en función de la carga de trabajo, sin necesidad de escalar todo el sistema.
- Modularidad y reutilización: al encapsular la lógica de generación de casos de prueba, este módulo puede ser reutilizado en otros proyectos.

- **Módulo de Almacenamiento**: Este módulo actúa como un repositorio de datos no estructurados, tales como archivos de configuración, resultados intermedios de procesos, y cualquier otro tipo de dato que no se ajuste al esquema relacional de la base de datos. Se opta por crear este módulo dado que:

  - Permite almacenar y gestionar datos no estructurados, que no se ajustan a la estructura de una base de datos relacional.
  - Proporciona un punto de acceso centralizado para los datos, lo que facilita su acceso y gestión por parte de otros módulos del sistema.

- **Módulo de Ejecución**: Módulo encargado de la ejecución de pruebas sobre implementaciones específicas de patrones de microservicios. Se decide crear este módulo por los mismos motivos a los mencionados para el Módulo de Procesamiento, acerca de la modularidad y reutilización. Además se agrega la necesidad de tener un componente especializado que pueda ejecutar cualquier implementación específica, sin importar su tecnología o lenguaje, esto se logra con los contenedores ya mencionados.

- **Medidas de seguridad**: A continuación se listan medidas de seguridad que se tuvieron en cuenta a la hora de especificar el sistema:

  - Comunicación cifrada entre componentes: Se utiliza el protocolo HTTPS para que la comunicación entre los diferentes componentes del sistema, garantizando que los datos transmitidos estén cifrados y protegidos.
  - Validación de orígenes: La API del sistema implementa políticas de *Cross-Origin Resource Sharing (CORS)* (MDN Web Docs, 2024a) para restringir los orígenes permitidos. Esto apunta a que solo solicitudes desde dominios de confianza puedan interactuar con la API.
  - Autorización: Los módulos de procesamiento y ejecución están protegidos mediante el uso de *API Keys*. Estas claves de autorización apuntan a que solo usuarios y servicios autenticados y autorizados puedan acceder y ejecutar funciones críticas del sistema, limitando el acceso no autorizado.

- **Consideraciones**: Dado que el sistema realiza operaciones complejas y de larga duración, como la ejecución de pruebas sobre implementaciones de microservicios, se ha decidido utilizar *WebSockets* (MDN Web Docs, 2024b) para la comunicación en tiempo real. Esta tecnología permite que la Aplicación Web reciba notificaciones instantáneas cuando una operación costosa haya finalizado, evitando así que los usuarios experimenten bloqueos o esperas prolongadas. De esta manera, se mejora la experiencia del usuario al mantener la interfaz interactiva y se les informa cuando sus datos están listos para ser consultados.

Capítulo 5

Implementación

En este capítulo se describe la implementación de la arquitectura descrita en la sección 4.1. Se utilizaron los servicios en la nube de Mi Nube de Antel (Antel, 2024b) Elastic Cloud 2.5.1 y una VPS 2.5.2 donde se alojan los servicios necesarios para el funcionamiento del sistema. La decisión fue motivada por el hecho de que Antel colabora con la Facultad de Ingeniera de la Universidad de la República, brindando la facilidad de utilizar dichos servicios sin cargo, a través de códigos promocionales, con propósitos académicos. Asimismo, Elastic Cloud brinda un servicio moderno e intuitivo, facilitando el despliegue de aplicaciones. Por otra parte, la VPS se utiliza para alojar los servicios donde Elastic Cloud no cubre con las necesidades del sistema, por ejemplo, el manejo dinámico de imágenes de Docker. El detalle de la VPS seleccionada es:

- Sistema Operativo: Linux Ubuntu
- Cantidad de núcleos: 4
- Memoria RAM: 8 Giga Bytes
- Almacenamiento: 150 Giga Bytes

A continuación se describe la arquitectura del sistema con sus tecnologías y lenguajes de programación específicos, utilizando los servicios mencionados. La figura 5.1 muestra el diagrama de arquitectura. Se observa que se el Módulo Controlador, Owncloud y Docker se encuentran en Elastic Cloud, y el Módulo de Procesamiento y el Módulo de Ejecución se encuentran en una VPS de Antel, expuestos mediante NGINX.

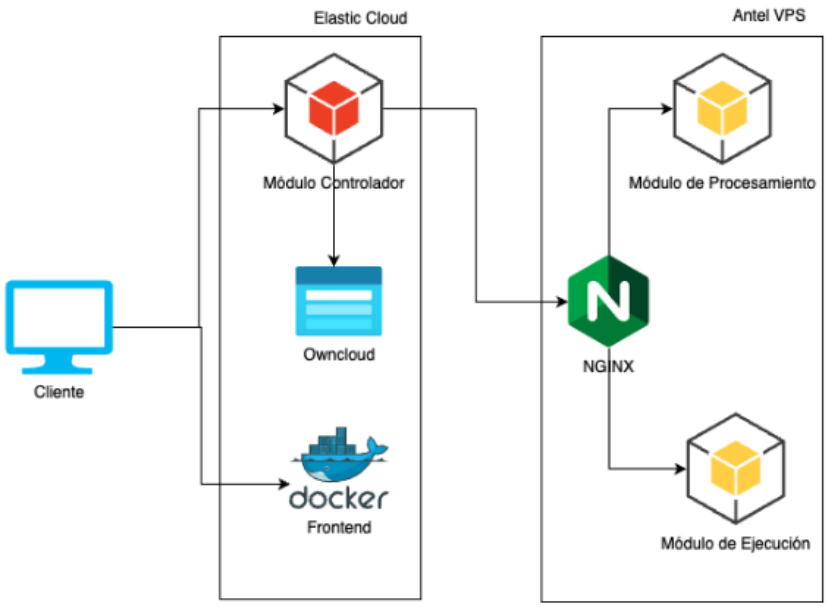

Figura 5.1: Diagrama arquitectura.

5.1. Módulos

En esta sección se describen los diferentes módulos de la solución y sus correspondientes componentes.

5.1.1. Aplicación Web

La Aplicación Web se implementó en React versión 18 (Meta, 2024) utilizando el framework RedwoodJS (Tom Preston-Werner, 2024) debido a su simplicidad y eficacia para desarrollar aplicaciones web desde cero, ya que provee diversas funcionalidades y configuraciones para acelerar el desarrollo. Además, se utilizó TypeScript (Microsoft Corporation, 2024), un lenguaje fuertemente tipado que permite identificar errores en tiempo de compilación.

RedwoodJS se eligió principalmente por su estructura de proyecto bien definida que incluye características como el enrutamiento, gestión de formularios y sencillamente listo para usar. Estas funcionalidades permiten que el desarrollo sea más rápido y que el equipo pueda concentrarse en los objetivos del proyecto sin preocuparse por las configuraciones del entorno.

Chakra UI (Segun Adebayo, 2024) se utilizó como biblioteca de componentes debido a su enfoque en la accesibilidad y la facilidad para construir interfaces de usuario modernas. Ofrece componentes pre construidos y extensibles, lo que permite centrarse en la funcionalidad y la experiencia de usuario sin tener que preocuparse por los detalles de estilo y diseño desde cero.

El propósito principal de la Aplicación Web es servir como la interfaz de usuario del sistema, permitiendo a los usuarios interactuar con el *Backend*. A través de esta interfaz, los usuarios pueden gestionar las Formalizaciones, Implementaciones, y Reportes de Patrones de microservicios, de manera remota y centralizada. La aplicación se comunica con el *Backend* a través de una API REST, utilizando peticiones HTTP.

Las siguientes librerías se utilizaron para mejorar la funcionalidad y el rendimiento de la aplicación:

- **Ably** (Ably Realtime Ltd., 2024): Utilizada para la implementación de WebSockets, permitiendo la comunicación en tiempo real con el *Backend*.
- **Axios** (Axios, 2024): Maneja la configuración y ejecución de peticiones HTTP.
- **Formik** (Formik, 2024): Facilita la gestión y validación de formularios, mejorando la interacción del usuario.
- **i18next** (i18next, 2024): Proporciona soporte multilenguaje, permitiendo adaptar la aplicación a diferentes idiomas.
- **Zustand** (Zustand, 2024): Se encarga del manejo del estado de la aplicación.
- **React Query** (TanStack, 2024): Gestiona las peticiones y el *cache* de datos, optimizando la eficiencia en la comunicación con el *Backend*.

La infraestructura de la Aplicación Web está totalmente alojada en Elastic Cloud, como se puede ver en la figura 5.5, y es una imagen de Docker que contiene la aplicación la cual está alojada en Dockerhub. El despliegue consiste en crear y subir una nueva imagen, generando un nuevo TAG en Dockerhub, para luego obtenerla y actualizarla desde Elastic Cloud.

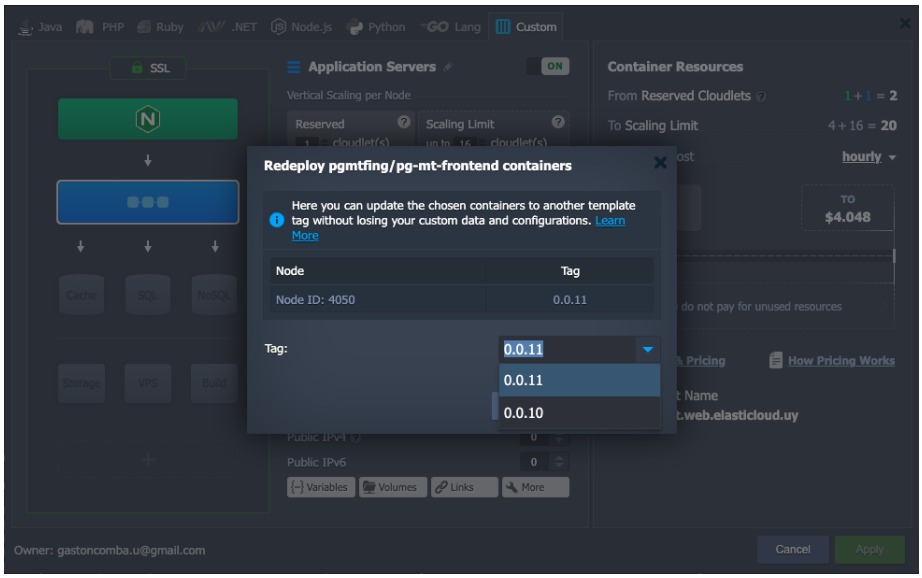

Figura 5.2: Infraestructura de la Aplicación Web.

El código fuente de la aplicación Web se encuentra en `https://gitlab.fing.edu.uy/tesis-testing-microservicios/microservices-testing-frontend`.

5.1.2. Módulo Controlador

En esta subsección se describen los componentes del módulo Controlador, los cuales son el *Backend* y la base de datos.

Backend

El *Backend* se implementó en Java (Oracle, 2024) utilizando Maven (Apache, 2024) como herramienta de gestión de proyectos y dependencias. Se eligió Maven porque es compatible con Elastic Cloud, lo que facilita el despliegue de la aplicación, ya que Elastic Cloud soporta aplicaciones Maven de manera nativa, simplificando el proceso de configuración y despliegue.

Para las migraciones de base de datos, se utilizó Flyway (Redgate, 2024), ya que permite gestionar las versiones y cambios en la estructura de la base de datos de forma sencilla. Java fue seleccionado como lenguaje de programación dado que es un lenguaje que hemos aprendido durante la carrera.

Para la implementación se seleccionaron diversas librerías, entre ellas se encuentran:

- **Spring Boot**: (Spring Framework, 2024) Se utilizó *Spring Boot* como el framework principal para la creación de la API REST.

- **Flyway**: (Redgate, 2024) Se empleó para la gestión de migraciones de la base de datos, lo que permite un control de versiones e integridad de los datos durante los cambios en el esquema de la base de datos.
- **PostgreSQL Driver**: (PostgreSQL Global Development Group, 2024) Se eligió el controlador de PostgreSQL para la interacción con la base de datos.
- **ModelMapper**: (ModelMapper, 2024) Herramienta utilizada para convertir objetos de una clase a otra, simplificando el proceso de conversión entre diferentes capas del sistema.
- **Mockito**: (Mockito, 2024) Se utilizó para la creación de pruebas unitarias, permitiendo la simulación de dependencias y facilitando su implementación.
- **Ably**: (Ably Realtime Ltd., 2024) Se utilizó para la comunicación en tiempo real mediante WebSockets, permitiendo actualizaciones instantáneas entre el *Backend* y los clientes (en este caso la Aplicación Web).
- **Jacoco**: (JaCoCo, 2024) Se utilizó para medir la cobertura de las pruebas.

El sistema se organizó siguiendo una arquitectura basada en capas, lo cual facilita la separación de responsabilidades, la escalabilidad, y el mantenimiento del código. Se definen la siguientes capas:

- **Capa de Controladores:** Los controladores son responsables de manejar las peticiones HTTP que llegan al *Backend*. Cada entidad del sistema cuenta con su propio controlador, como por ejemplo *PatternCtrl*, que maneja las peticiones relacionadas con los Patrones. Estos controladores exponen los puntos de acceso necesarios para las operaciones básicas como: Crear, Leer, Actualizar y Eliminar, y se encargan de invocar los servicios apropiados para procesar cada solicitud.
- **Capa de Servicios:** La lógica de negocio del sistema está encapsulada en la capa de servicios. Esta capa contiene clases como *PatternService* que implementan las reglas de negocio y coordinan la interacción entre los controladores y la capa de repositorios.
- **Capa de Repositorios:** La capa de repositorios es responsable de la interacción con la base de datos. Utiliza *Spring Data JPA* para realizar operaciones sobre las entidades persistidas. Por ejemplo, *PatternRepository* proporciona los métodos necesarios para acceder y manipular datos de los Patrones.
- **Integración con OwnCloud:** El *Backend* integra funcionalidad para la gestión de archivos mediante OwnCloud, lo que permite almacenar y recuperar archivos, como Formalizaciones, y Estrategias. Esta integración se realizó a través de una API que permite interactuar con OwnCloud.

- **Integración con los Módulos de Procesamiento y Ejecución:** El *Backend* actúa como intermediario entre la Aplicación Web y los Módulos de Procesamiento y Ejecución. Estos módulos se encargan de generar y ejecutar las pruebas sobre las Implementaciones de microservicios. El *Backend* envía las solicitudes a estos módulos, maneja las respuestas, y proporciona la información necesaria a la Aplicación Web para su presentación al usuario. Dichos módulos exponen sus servicios mediante una API definida, lo que permite que sean reemplazables, por lo que a futuro se puede integrar o reemplazar el módulo, y sus proveedores.

En la figura 5.3 se muestra el diagrama de arquitectura en capas del Módulo Controlador.

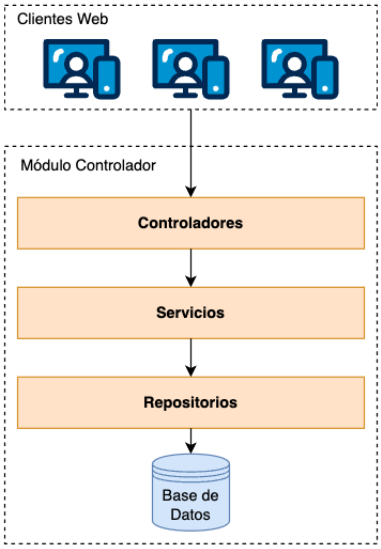

Figura 5.3: Arquitectura en capas del Módulo Controlador.

Se desarrollaron pruebas unitarias para el *Backend* con el objetivo de mejorar la calidad y la robustez del código. Estas pruebas cubren los controladores, servicios y repositorios, alcanzando una cobertura del 91 %, como se muestra en la figura 5.4. Para medir la cobertura de las pruebas, se utilizó JaCoCo (JaCoCo, 2024), el cual está configurado en el proyecto mediante la extensión **jacoco-maven-plugin** (EclEmma Development Team, 2024), y se configura en el archivo de configuración **pom.xml** la ejecución y generación de reportes de cobertura utilizando el comando **mvn verify**. Las pruebas unitarias permiten identificar errores de manera anticipada, facilitando la detección de potenciales problemas cuando se realizan cambios en el código, se agregan nuevas funcionalidades o se modifican las existentes.

api

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
pg.microservicesTesting.api		92%		n/a	1	5	2	9	1	5	0	2
pg.microservicesTesting.api.ably		80%		n/a	1	4	2	12	1	4	0	3
pg.microservicesTesting.api.animations.controllers		100%		n/a	0	3	0	8	0	3	0	1
pg.microservicesTesting.api.animations.models		100%		n/a	0	28	0	28	0	28	0	3
pg.microservicesTesting.api.animations.services		100%		100%	0	5	0	19	0	4	0	1
pg.microservicesTesting.api.configurations		66%		0%	6	13	11	25	5	12	0	5
pg.microservicesTesting.api.files		100%		n/a	0	6	0	8	0	6	0	2
pg.microservicesTesting.api.formalizations.controllers		82%		n/a	2	9	5	23	2	9	0	2
pg.microservicesTesting.api.formalizations.models		100%		n/a	0	26	0	27	0	26	0	3
pg.microservicesTesting.api.formalizations.services		90%		50%	2	9	8	55	0	7	0	1
pg.microservicesTesting.api.implementations.controllers		100%		n/a	0	7	0	17	0	7	0	2
pg.microservicesTesting.api.implementations.models		100%		n/a	0	42	0	43	0	42	0	4
pg.microservicesTesting.api.implementations.services		100%		n/a	0	6	0	18	0	6	0	1
pg.microservicesTesting.api.own_cloud		78%		55%	10	21	28	121	3	11	0	1
pg.microservicesTesting.api.patterns.controllers		100%		n/a	0	13	0	31	0	13	0	5
pg.microservicesTesting.api.patterns.models		100%		n/a	0	31	0	34	0	31	0	4
pg.microservicesTesting.api.patterns.services		84%		100%	2	9	6	25	2	8	1	2
pg.microservicesTesting.api.process_module		100%		100%	0	3	0	25	0	2	0	1
pg.microservicesTesting.api.reports.controllers		83%		n/a	1	3	2	8	1	3	0	1
pg.microservicesTesting.api.reports.models		100%		n/a	0	34	0	35	0	34	0	3
pg.microservicesTesting.api.reports.services		100%		n/a	0	5	0	13	0	5	0	1
pg.microservicesTesting.api.server_module		95%		50%	1	4	3	29	0	3	0	1
Total	221 of 2,622	91%	14 of 34	58%	26	286	67	613	15	269	1	49

Figura 5.4: Cobertura de pruebas unitarias con JaCoCo.

La infraestructura del *Backend* está totalmente alojada en Elastic Cloud, representada en la figura 5.5, donde se seleccionó: una aplicación Springboot para contener al *Backend*, una base de datos PostgreSQL y una aplicación Maven para encargarse de la compiliación y despliegue. El despliegue de la aplicación está hecho totalmente con Elastic Cloud, con la aplicación Maven. En este despliegue la plataforma se encarga de tomar el código del repositorio en GitLab, compilarlo y dejarlo disponible en la correspondiente URL al accionar el botón *Build-and-Deploy* de la aplicación Maven. Esta extensión la ofrece Elastic Cloud bajo el nombre de *Git-Push-Deploy* y permite configurar de manera muy sencilla la totalidad del despliegue del *Backend*. Elastic Cloud también ofrece SSL de manera gratuita, la cual se configura sencillamente desde el menú de configuraciones de la aplicación, por lo cual se agrega este mecanismo de seguridad.

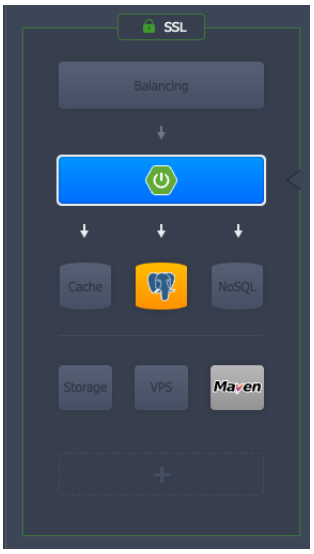

Figura 5.5: Infraestructura del Backend.

El código fuente del backend se encuentra en https://gitlab.fing.edu.uy/tesis-testing-microservicios/api

Base de datos

Se utilizó PostgreSQL 16.2 (The PostgreSQL Global Development Group, 2023) para la base de datos. El diagrama de la base de datos se presenta en la figura 5.6.

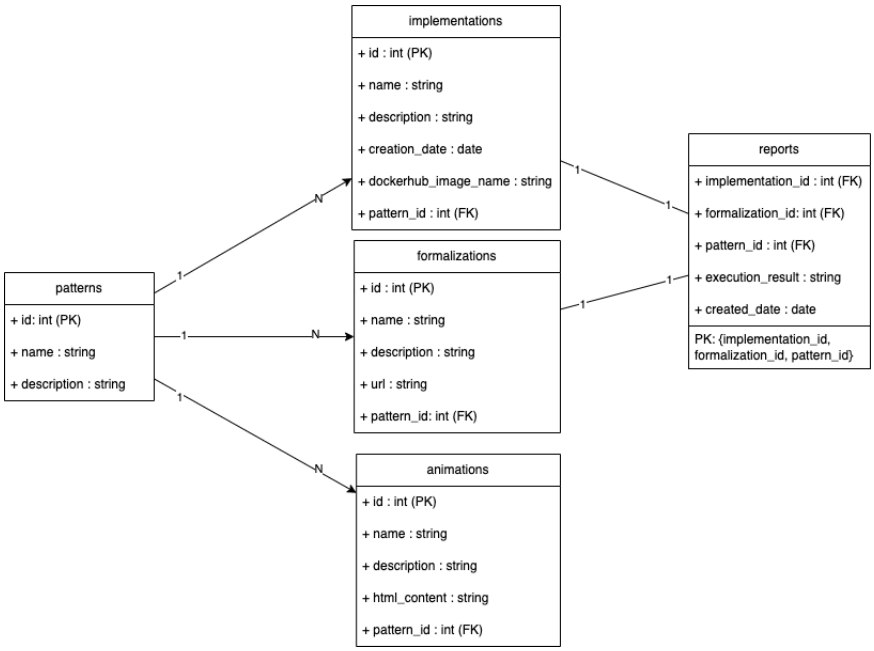

Figura 5.6: Diagrama de base de datos.

En este diagrama, se observa que la entidad principal es el Patrón, que actúa como núcleo central alrededor del cual se organizan las demás entidades. Cada Patrón puede tener múltiples Implementaciones, Formalizaciones y Animaciones asociadas.

Los Reportes generados están vinculados tanto a una Implementación como a una Formalización, y ambas deben corresponder al mismo Patrón.

Este diseño de base de datos permite almacenar y gestionar la información del sistema de la siguiente manera:

- **Consistencia del Patrón:** Cada Patrón tiene un nombre y una descripción, lo que establece su identidad dentro del sistema. Esto apunta a que todas las Formalizaciones, Implementaciones y Animaciones asociadas estén correctamente vinculadas a su Patrón correspondiente.
- **Multiplicidad de Formalizaciones:** Un Patrón puede tener varias Formalizaciones, permitiendo que un mismo Patrón pueda ser expresado en distintas notaciones formales, como Event-B.
- **Diversidad de Implementaciones:** Cada Patrón puede tener múltiples Implementaciones, cada una de ellas desarrollada en diferentes tecnologías o lenguajes de programación.
- **Relación entre Formalizaciones e Implementaciones:** Los Reportes generados, que contienen los resultados de las pruebas, están directamente asociados a una Implementación y una Formalización del mismo Patrón.

5.1.3. Módulo de Procesamiento

El Módulo de Procesamiento se implementó en Python (Python Software Foundation, 2024) utilizando el framework Flask (Pallets, 2024) para la creación de la API. Se decidió utilizar esta tecnología dado que permite crear una API rápidamente con baja configuración y fácil ejecución de comandos. Ademas se tuvo en cuenta que este módulo esta basado en la lógica propuesta por Vergara (Vergara, 2021a) para la generación de pruebas, la cual utiliza módulos y librerías en Python, lo cual facilita su integración.

Este módulo se encarga de la generación de pruebas unitarias en Java utilizando JUnit, siguiendo un flujo similar al descrito por Vergara. El proceso integra dos pasos principales: el *test case generator* y el *test case instantiator*. La decisión de combinar estos pasos en una única operación se toma para reducir la cantidad de interacciones necesarias por parte del usuario, simplificando así el proceso de pruebas. El usuario sólo necesita proporcionar los archivos de estrategia para que el flujo completo se ejecute automáticamente.

Se utilizó la librera *pyocclient* (owncloud, 2024b), la cual facilitó enormemente la integración con OwnCloud ya que provee una interfaz amigable para la gestión de los datos no estructurados. Se utiliza para la carga y descarga de archivos. Por ejemplo, cargar el proyecto Java completado por el usuario.

La API expone un único punto de acceso, el cual es invocado por el *Backend*. Luego de la autenticación con API-Key[1], recibe tres referencias a archivos en OwnCloud: uno para el archivo de Formalización en Event-B, otro para el archivo de Estrategia de generación de casos de prueba abstractos B, y el último para el archivo de Estrategia de instanciación de casos de prueba C. El módulo utiliza tres proyectos realizados por Vergara, y una herramienta externa:

- **probcli_test_generator** (Python): Encargado de la creación de los casos de prueba abstractos.
- **CustomStrategy** (Java): Referenciado por el proyecto **probcli_test_generator** para instanciar los casos de prueba.
- **ProB CLI** (Heinrich-Heine-University, Institut für Software und Programmiersprachen, 2024b): Herramienta utilizada por **probcli_test_generator** para la generación de pruebas.
- **Xml2Junit** (Java): Encargado de crear el proyecto Java con los casos de prueba generados.

El flujo de operación del módulo es el siguiente:

1. Descarga de archivos de OwnCloud: Se descargan los archivos de estrategia y la formalización en Event-B desde OwnCloud y se guardan en una carpeta temporal.

[1] Cadena de caracteres única con el propósito de identificar y autenticar a una aplicación o a un usuario

2. Ejecución de **probcli_test_generator**: Utiliza los archivos descargados para generar un archivo XML con los casos de prueba abstractos, que se guarda en una carpeta temporal.

3. Ejecución de **CustomStrategy**: Tomando la salida del paso anterior y la Estrategia de instanciación para generar un proyecto Java, el cual se guarda en la carpeta temporal.

4. Guardado en OwnCloud: El proyecto Java generado se sube a OwnCloud, se elimina la carpeta temporal, y se devuelve la ruta del proyecto guardado.

El diagrama que se observa en la figura 5.7 representa el flujo mencionado previamente.

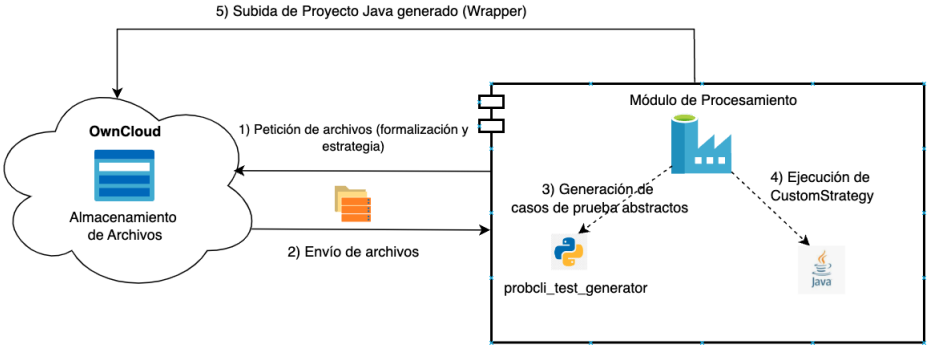

Figura 5.7: Diagrama de de flujo del Módulo de Procesamiento.

En caso de que alguno de los pasos anterior falle, se devuelve una respuesta de error, en este caso al *Backend* quien fue el que invocó a la API.

El Módulo de Procesamiento está desplegado en una VPS de Antel, utilizando Nginx como servidor web. El despliegue se realiza mediante un simple comando *git pull* desde la VPS. Se decidió utilizar la VPS de Antel debido a la necesidad de manejar proyectos Java dinámicamente. El módulo se encuentra disponible en `https://processor.pg-mt-runner.cloud`, y su acceso es encriptado mediante SSL, y asegurado con API-Key.

El código fuente del Módulo de Procesamiento se encuentra en `https://gitlab.fing.edu.uy/tesis-testing-microservicios/mircroservices-testing-process`.

5.1.4. Módulo de Ejecución

El Módulo de Ejecución fue implementado en Python (Python Software Foundation, 2024), utilizando el framework Flask (Pallets, 2024) para la creación

de la API. La elección de esta tecnología siguió los mismos criterios mencionados para el Módulo de Procesamiento.

El propósito de este módulo es ejecutar las pruebas generadas por el Módulo de Procesamiento tras completadas por el usuario. La API ofrece un punto de acceso protegido por autenticación mediante API-Key. Recibe dos parámetros: la ruta del proyecto Java completo, es decir el proyecto Wrapper, almacenado en OwnCloud, y el nombre de la imagen de Dockerhub con la implementación. Una vez recibidos, el módulo se encarga de ejecutar la imagen de Docker, y correr sobre esta las pruebas correspondientes, devolviendo los resultados. Al igual que en el Módulo de Procesamiento, se utilizó la librería *pyocclient* (owncloud, 2024b), la cual facilitó la integración con OwnCloud, dado que provee fácil acceso para la gestión de los datos no estructurados que se necesitan.

El uso de imágenes de Docker para contener las implementaciones se debe a la flexibilidad que ofrece, permitiendo ejecutar aplicaciones sin importar su lenguaje o tecnología, ni la necesidad de instalar software adicional en el servidor. Esto busca proveer un entorno de ejecución aislado para las pruebas.

El módulo opera de la siguiente forma:

1. Descarga de archivos de OwnCloud: El módulo recibe las rutas del proyecto Java y la imagen en Dockerhub, descarga el proyecto Java y lo almacena temporalmente en el servidor.

2. La imagen de Docker correspondiente a la Implementación se descarga y se ejecuta en un contenedor aislado.

3. Se ejecuta el proyecto Java, realizando peticiones a la implementación la cual es ejecutada en un contenedor de Docker.

4. Una vez completadas las pruebas, se elimina la carpeta temporal, y se devuelven sus resultados.

El diagrama que se observa en la figura 5.8 representa el flujo mencionado previamente.

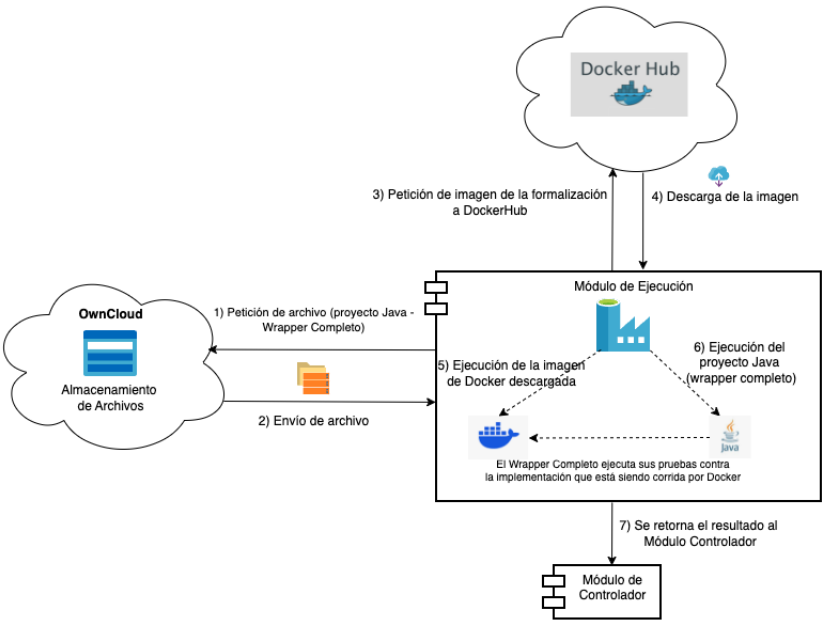

Figura 5.8: Diagrama de de flujo del Módulo de Ejecución.

En caso de que alguno de los pasos anteriores falle, se devuelve una respuesta de error, en este caso al *Backend* quien fue el que hizo la llamada a la API.

El Módulo de Ejecución esta desplegado en una VPS de Antel, utilizando Nginx como servidor web. El despliegue se realiza mediante el comando *git pull* en el servidor, sin necesidad de compilación previa debido a que Python es un lenguaje interpretado, actualizando así el código fuente del módulo. Se decidió alojar este módulo en una VPS debido a la necesidad de manejar imágenes Docker y ejecutar proyectos Java de manera dinámica. El módulo se encuentra disponible en `https://server.pg-mt-runner.cloud`, y su acceso es encriptado mediante SSL, y asegurado con API-Key.

El código fuente del Módulo de Ejecución se encuentra en `https://gitlab.fing.edu.uy/tesis-testing-microservicios/pg-mt-env-runner`.

5.1.5. Módulo de Almacenamiento

El Módulo de Almacenamiento utiliza la tecnología OwnCloud (owncloud, 2024a), la cual es una herramienta para el almacenamiento y gestión de archivos en la nube, tal como dice su sitio: “ownCloud representa un almacenamiento en la nube de archivos gratuito y de código abierto. Se puede utilizar fácilmente para compartir y sincronizar datos, así como simplemente para almacenar documentos”. OwnCloud se encuentra en Elastic Cloud, y se puede instanciar desde el MarketPlace, dado que tiene una integración nativa con la herramienta. Este es el principal motivo para su elección, y además, brinda una API para gestionar

los archivos.

5.2. Problemas encontrados

En esta sección se describen ciertos problemas que se han detectado en la etapa de implementación del proyecto, se explica su origen y su solución.

5.2.1. Procesos de larga duración

Al realizar pruebas en el sistema, se observó en la implementación de *Eureka* del patrón Service Registry, que la ejecución de las mismas demoraba aproximadamente 5 minutos. Al considerar este tiempo demasiado extenso para mantener al usuario esperando, se buscó una solución a este inconveniente. Finalmente se optó por integrar WebSockets a modo de que durante estas operaciones costosas, el usuario pueda seguir utilizando la aplicación, y recibir una notificación cuando el proceso concluya. Esto a punta a una mejora en la experiencia de usuario al usar la aplicación.

5.2.2. Limitación de recursos en la nube

Al seleccionar como proveedor de servicios en la nube a Antel, y luego de realizar prototipos mínimos funcionales de los módulos, se comenzó a trabajar en desplegarlos en la nube. En particular, para los Módulos de Procesamiento 5.1.3 y Ejecución 5.1.4, se observa que se necesita utilizar una VPS, y no un servicio en Elastic Cloud. Esto se debe a que, para el Módulo de Procesamiento, es necesario correr servicios de Python y Java, lo cual no es posible en un servicio predeterminado de Elastic Cloud. Para el Módulo de Ejecución, a pesar de que Elastic Cloud soporta Docker, no soporta la carga de diferentes imágenes a demanda, lo cual es necesario para correr las pruebas. Se optó por correr ambos módulos en una VPS para poder tener todos los recursos necesarios. Sin embargo buscando optimizar recursos, se optó por colocar ambos módulos en la misma VPS. Idealmente, y como se muestra en el diagrama de arquitectura de la figura 4.1 cada uno debería correr en su propia infraestructura aislada.

5.2.3. Patrones con varios servicios

Durante la etapa de implementación se encontró el problema de que uno de los proyectos Wrapper que se obtuvo de la tesis de maestría de Vergara, en concreto de Service Registry, inicializaba aplicaciones en distintos puertos para poder llevar acabo las pruebas. Esto es esperable ya que la naturaleza del patrón de arquitectura de Service Registry consiste en registrar diversos servicios. El problema surge en el proyecto Wrapper, a la hora de terminar los procesos generados en las pruebas, ya que, para poder probar esta implementación, se ejecutan varios servicios en varios puertos. Estos consumen recursos e imposibilitan hacer otras pruebas en los puertos utilizados hasta que se reinicie la VPS, dado que

quedan ocupados. Para resolver este problema, se implementó un proceso de limpieza de los procesos que estén en el rango conocido. Este proceso consiste en buscar todos los procesos que estén asociados a puertos abiertos dentro de dicho rango, para terminarlos y así liberar los recursos y puertos de la VPS, habilitándolos para ser reutilizados en posteriores pruebas.

5.2.4. Visualización de animaciones

En un principio, se había tomado la decisión de utilizar BMotionWeb (Heinrich-Heine-University, Institut für Software und Programmiersprachen, 2024a) para el manejo y visualización de animaciones. Luego de varios intentos, y pruebas de concepto, esto no fue posible debido a la falta de documentación alrededor de la herramienta. Pese a esta limitación, se encontró la forma de almacenar las animaciones en formato HTML, permitiendo a los usuarios generar sus animaciones en sus propios ambientes, y subirlos en este formato a la plataforma, pudiendo así, servir de repositorio centralizado de animaciones.

Capítulo 6

Verificación y validación

En este capítulo se describe cómo se verificó y validó el sistema propuesto. En primer lugar, se presenta la ejecución del flujo principal, lo que permite mostrar cómo obtener resultados en el sistema propuesto. En segundo lugar, se generan los resultados sobre las mismas implementaciones de patrones de microservicios presentadas por Vergara y se verifica que el sistema propuesto obtiene los mismos resultados. Por último, se presentan los resultados y conclusiones obtenidos tras realizar una presentación y encuesta a usuarios expertos.

6.1. Ejecución flujo principal

En esta sección se presenta la ejecución del flujo principal del sistema con el fin de mostrar como obtener resultados en el sistema propuesto, para luego poder compararlos con los resultados obtenidos por Vergara y verificar lo construido. El flujo principal consta de tres interacciones con el usuario:

1. **Creación del Patrón**: El usuario crea un Patrón, donde se le solicita al usuario ingresar su nombre, descripción y al menos una formalización en Event-B.
2. **Creación de la Implementación**: El usuario crea la Implementación, donde se le solicita ingresar el nombre, el nombre de una imagen en Dockerhub que contiene la implementación y una descripción.
3. **Creación y ejecución de las pruebas**: Se crean y se ejecutan las pruebas para el Patrón, donde se le solicita al usuario ingresar el nombre de la prueba, y una Implementación y Formalización del Patrón.

6.1.1. Creación del Patrón

El usuario inicia el flujo al pulsar el botón “Crear”, donde se le solicita ingresar el nombre, descripción y una Formalización, tal como se ve en la figura 6.2.

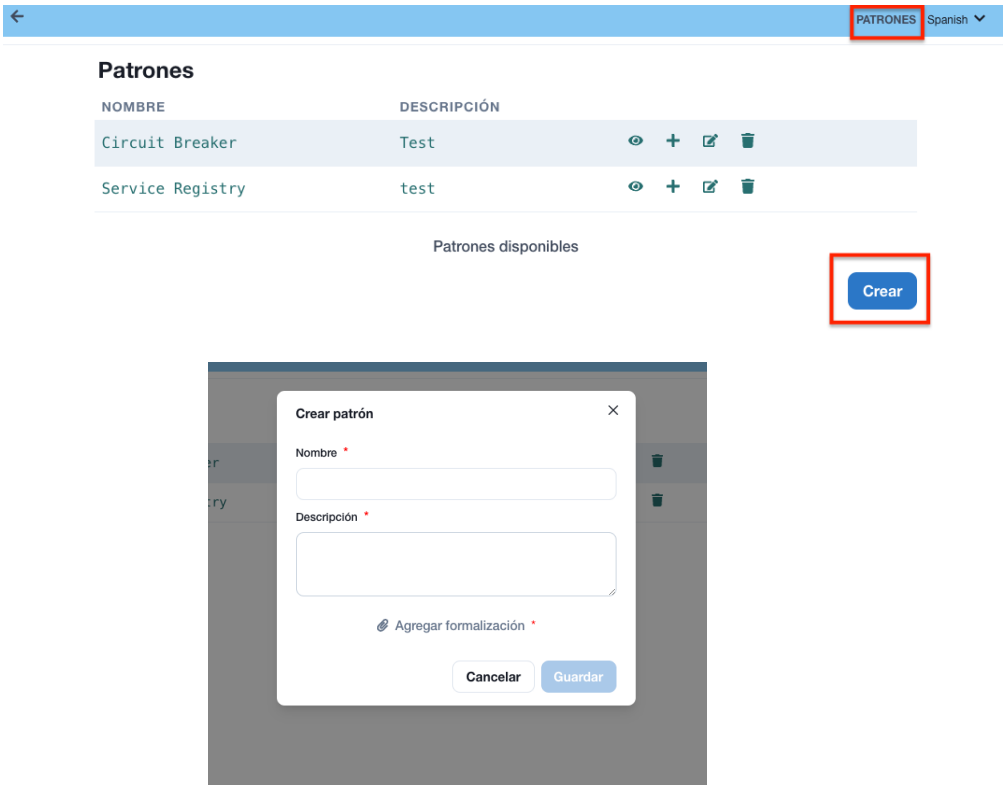

Figura 6.1: Flujo principal - Creación del Patrón.

6.1.2. Creación de la implementación

El usuario inicia esta parte del flujo al pulsar el botón “Agregar implementación” donde se le solicita al usuario ingresar el nombre, el nombre de una imagen en Dockerhub conteniendo la implementación y una descripción, tal como se ve en la figura 6.1. Es importante notar que para esta parte, se le hace saber al usuario que dicha implementación alojada en Dockerhub debe ser pública y estar expuesta en el puerto 8090.

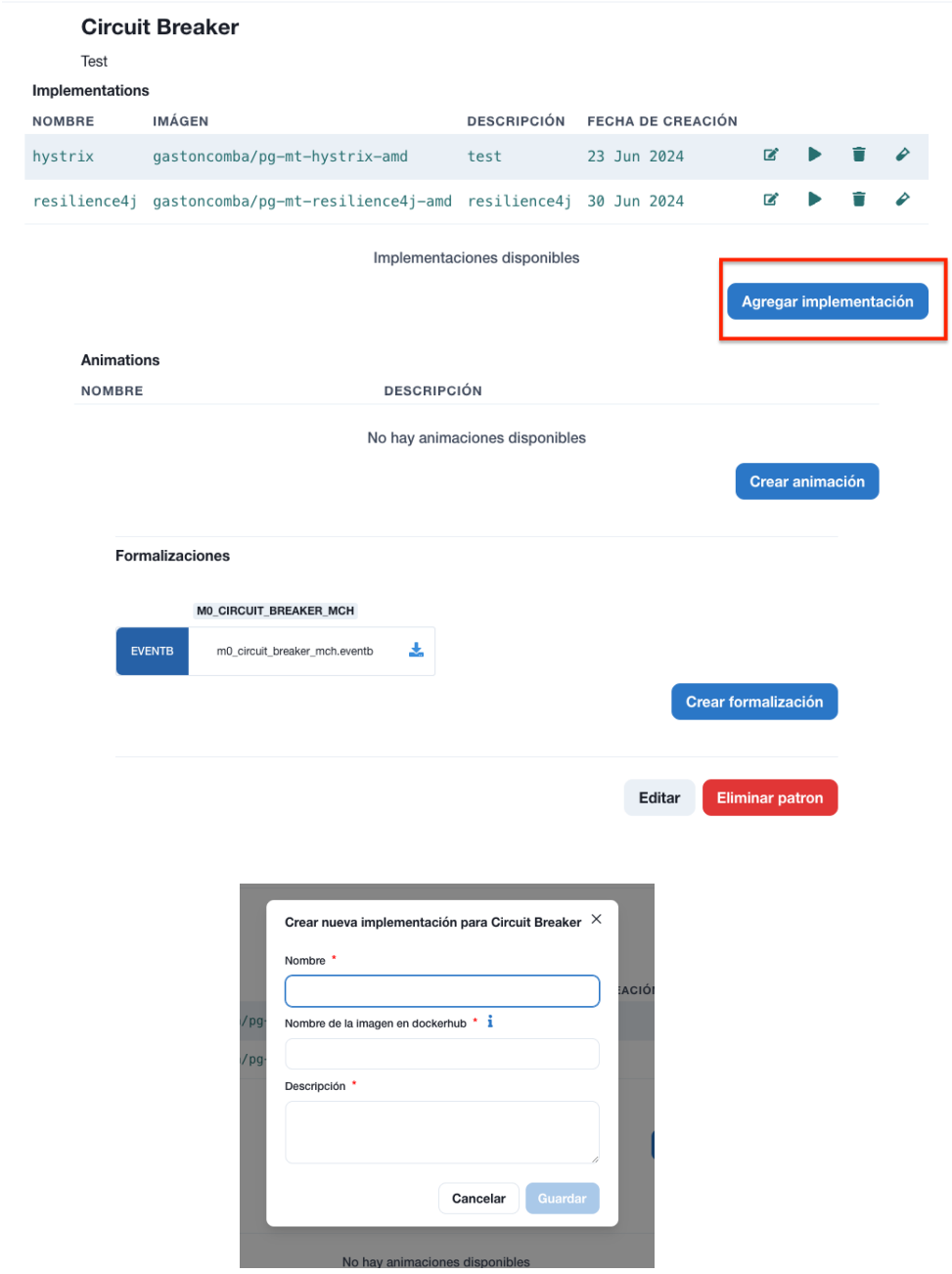

Figura 6.2: Flujo principal - Creación de la Implementación

6.1.3. Creación y ejecución de pruebas

La primera parte de este proceso incluye seleccionar el icono desde una Implementación a probar y la carga de archivos de configuración. Esto requiere nombrar la prueba, seleccionar una Formalización, y cargar una Estrategia de pruebas abstractas (ver Anexo B), y Estrategia de configuración para la instanciación de pruebas abstractas (ver Anexo C).

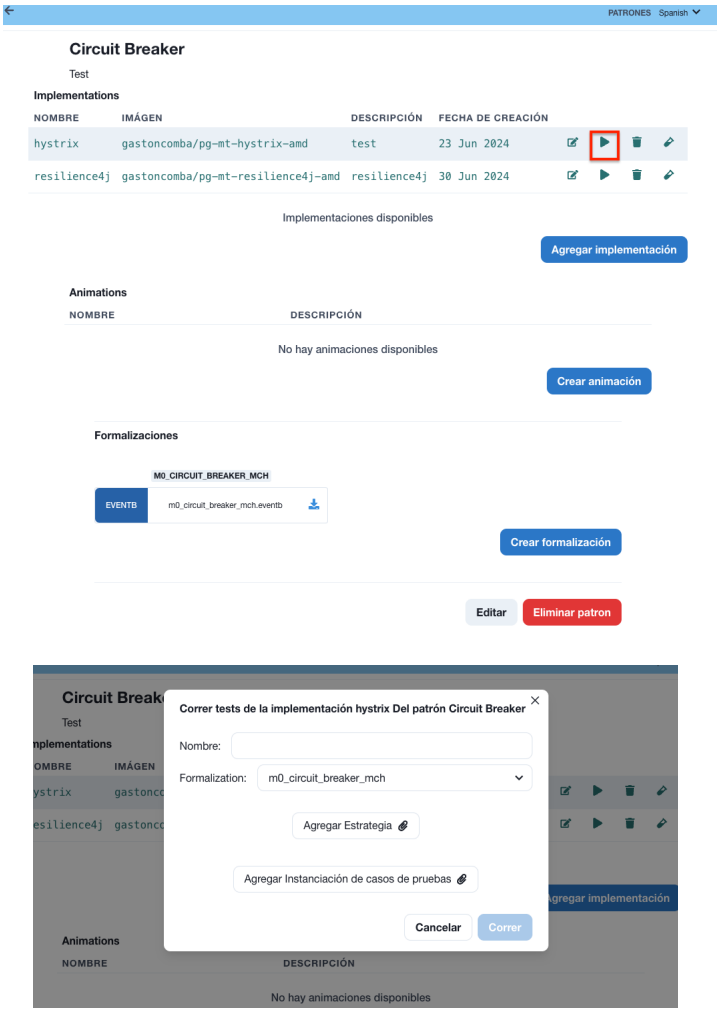

Figura 6.3: Flujo principal - Creación de las pruebas 1

Figura 6.4: Flujo principal - Creación de las pruebas 2

Este paso crea un proyecto en Java, el cual se denomina Wrapper, y el usuario debe descargar y completar, a modo de que dichas pruebas puedan ejecutarse contra la Implementación previamente definida. Como se ve en la figura 6.5, se comienzan a ejecutar las pruebas y, al finalizar, el usuario recibe una notificación de "Nuevo reporte disponible", donde los resultados se pueden ver en la figura 6.6.

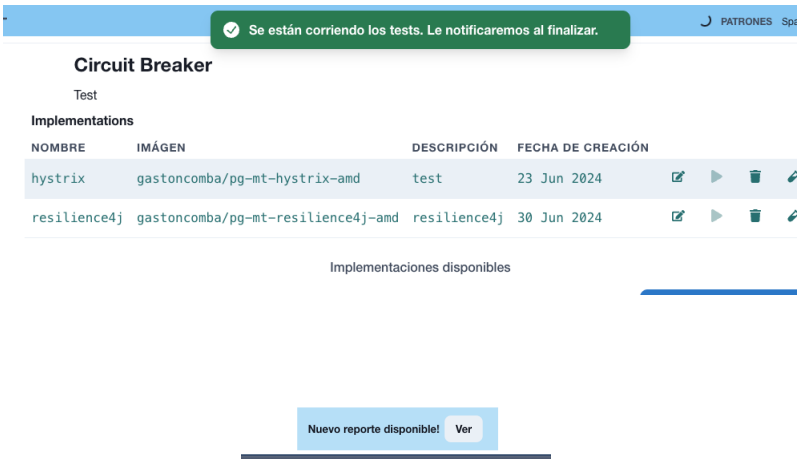

Figura 6.5: Flujo principal - Ejecución de pruebas

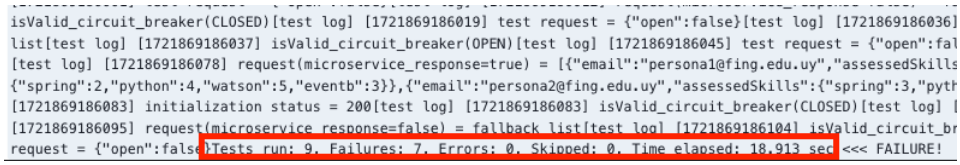

Figura 6.6: Flujo principal - Resultados

6.2. Patrones de microservicios

En esta sección se verifica que los resultado obtenidos en la plataforma son consistentes con los resultados obtenidos por Vergara. Se toman los siguientes patrones e implementaciones de los mismos para realizar la verificación.

- **Circuit Breaker**
  - **Hystrix**
  - **Resilience4j**
- **Service Registry**
  - **Eureka**

6.2.1. Circuit Breaker

Hystrix

Para el patrón Circuit Breaker, con la implementación *Hystrix* de Netflix, Vergara obtiene un índice de conformidad del 22 % con dos pruebas aprobadas de las nueve totales, lo cual se puede observar en la fórmula 6.1.

$$CI(Hystrix) = \frac{\texttt{número de test aprobados}}{\texttt{número de test totales}} \times 100 = \frac{2}{9} \times 100 \approx 22\,\% \quad (6.1)$$

Al ejecutar las mismas pruebas en la nueva plataforma, se obtienen los resultados que se encuentran en la figura 6.7. Por lo tanto, se obtiene el mismo valor de índice de conformidad que Vergara, verificando los resultados obtenidos.

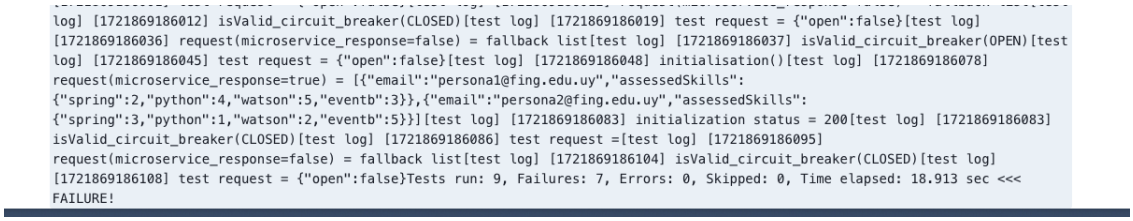

Figura 6.7: Resultado de ejecución de pruebas en implementación *Hystrix*

Resilience4j

Para el patrón Circuit Breaker, con la implementación *Resilience4j*, Vergara obtiene un índice de conformidad del 67 % con seis pruebas aprobadas de las nueve totales, lo cual se puede observar en la fórmula 6.2.

$$CI(Resilience4j) = \frac{\texttt{número de test aprobados}}{\texttt{número de test totales}} \times 100 = \frac{6}{9} \times 100 \approx 67\,\% \tag{6.2}$$

Al ejecutar las mismas pruebas en la nueva plataforma, se obtienen los resultados que se encuentran en la figura 6.8. Por lo tanto, se obtiene el mismo valor de índice de conformidad que Vergara, verificando los resultados obtenidos

```
testCase_9(uy.edu.fing.svergara.WrapperTest) Time elapsed: 3.281 sec <<< FAILURE! junit.framework.AssertionFailedError at
junit.framework.Assert.fail(Assert.java:47) at junit.framework.Assert.assertTrue(Assert.java:20) at
junit.framework.Assert.assertTrue(Assert.java:27) at uy.edu.fing.svergara.WrapperTest.testCase_9(WrapperTest.java:538) Results :
Failed tests: testCase_7(uy.edu.fing.svergara.WrapperTest) testCase_8(uy.edu.fing.svergara.WrapperTest)
testCase_9(uy.edu.fing.svergara.WrapperTest) Tests run: 9, Failures: 3, Errors: 0, Skipped: 0 [ [1;34mINFO [m]  [1m--------------
------------------------------------------------------------ [m [ [1;34mINFO [m]  [1;31mBUILD FAILURE [m [ [1;34mINFO [m]  [1m---------
------------------------------------------------------------------------ [m [ [1;34mINFO [m] Total time: 33.900 s [ [1;34mINFO [m] Finished
at: 2024-06-30T20:04:30Z [ [1;34mINFO [m]  [1m------------------------------------------------------------------------ [m
[ [1;31mERROR [m] Failed to execute goal  [32morg.apache.maven.plugins:maven-surefire-plugin:2.12.4:test [m  [1m(default-test) [m on
project  [36mjunit-circuit-breaker-resilience4j [m:  [1;31mThere are test failures. [m [ [1;31mERROR [m]  [1;31m [m [ [1;31mERROR [m]
 [1;31mPlease refer to /usr/server-runner/pg-mt-env-runner/temporal_files/junits/d86b88e4-8c33-4542-9f66-9eb59fb9cf05/junit-
circuit-breaker-resilience4j/target/surefire-reports for the individual test results. [m [ [1;31mERROR [m] ->  [1m[Help 1] [m
[ [1;31mERROR [m]  [ [1;31mERROR [m] To see the full stack trace of the errors, re-run Maven with the  [1m-e [m switch.
[ [1;31mERROR [m] Re-run Maven using the  [1m-X [m switch to enable full debug logging. [ [1;31mERROR [m]  [ [1;31mERROR [m] For more
information about the errors and possible solutions, please read the following articles: [ [1;31mERROR [m]  [1m[Help 1] [m
http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
```

Figura 6.8: Resultado de ejecución de pruebas en implementación *Resilience4j*

6.2.2. Service Registry

Para el patrón Service Registry, con la implementación *Eureka*, Vergara obtiene un índice de conformidad del 100 % con tres pruebas aprobadas de las tres totales, lo cual se puede observar en la fórmula 6.3.

$$CI(Eureka) = \frac{\texttt{número de test aprobados}}{\texttt{número de test totales}} \times 100 = \frac{3}{3} \times 100 = 100\,\% \tag{6.3}$$

Al ejecutar las mismas pruebas en la nueva plataforma, se obtienen los resultados que se encuentran en la figura 6.9. Por lo tanto, se obtiene el mismo valor de índice de conformidad que Vergara, verificando los resultados obtenidos

Figura 6.9: Resultado de ejecución de pruebas en implementación *Eureka*

6.3. Juicio de expertos

En esta sección se presentan los resultados obtenidos de una encuesta realizada a personas con experiencia variada en el área, a los cuales se les realizó una presentación mostrando el flujo principal del sistema. La encuesta se realizó para obtener retroalimentación, comentarios y sugerencias de dichos usuarios expertos, a fin de validar la plataforma. Se obtuvieron cuatro respuestas de la encuesta, lo que si bien no llega a la totalidad de los participantes en la reunión (diez) refleja el intercambio que hubo durante dicha instancia.

Como se observa en la figura 6.10, todos los usuarios tuvieron una experiencia positiva con respecto a la plataforma, ya que votaron con más de tres puntos en la escala de uno a cinco. Además, como se puede ver en la figura 6.11, también todos votaron con más de tres puntos al preguntarles si consideran que lo construido facilita las tareas en comparación con la solución de Vergara, por lo que se destaca la utilidad de la plataforma con respecto a la solución anterior.

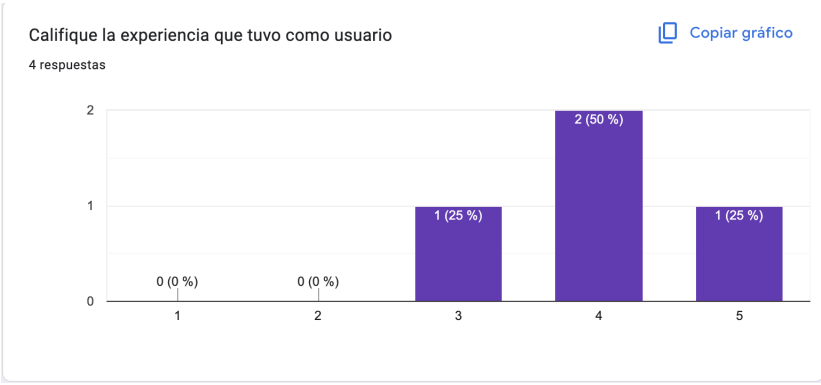

Figura 6.10: Experiencia como usuario.

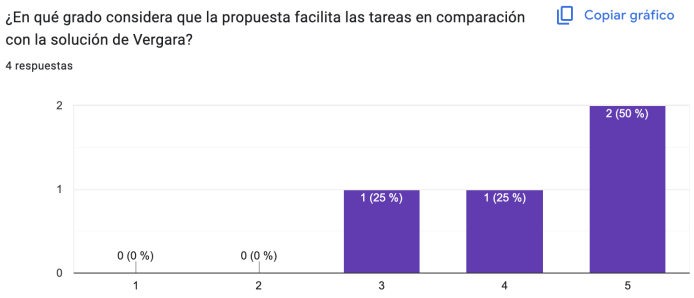

Figura 6.11: Grado de facilidad en comparación con solución de Vergara.

También es de interés destacar que, a partir de las respuestas de la encuesta, se observa que la parte del proceso que perciben más compleja en la práctica es la creación de entidades en el sistema y visualización de resultados, como se observa en la figura 6.12. El resultado se alinea con lo que se pensó previamente, ya que todo lo que refiere a la creación de entidades en el sistema abarca la creación y modelado de formalizaciones, lo cual está por fuera de la plataforma y son tareas no triviales como se puede ver en el trabajo de Vergara (Vergara, 2021a). Por otro lado la visualización de resultados puede ser mejorada ya que se presenta dando la salida de la consola de la ejecución de las pruebas y no es muy amigable para el usuario, como se puede también apreciar en la encuesta.

Figura 6.12: Componentes complejos del sistema.

Al consultar a los usuarios qué parte de la solución consideran mejor con respecto a la solución de Vergara, el aspecto que más se destaca es la interfaz de usuario. Asimismo, los usuarios también destacaron el uso de recursos y disponibilidad de la plataforma. Estos resultados se pueden observar en la figura 6.13. Esto es un resultado satisfactorio, ya que se pasó de usar Jupyter Notebooks con ejecución local, a una plataforma web ejecutada en un servidor remoto.

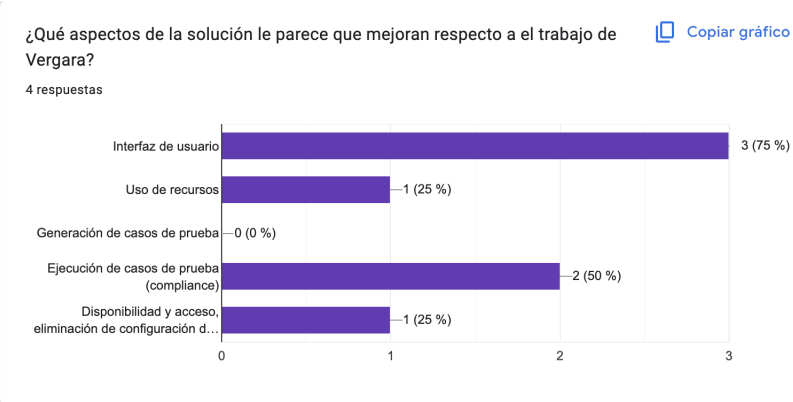

Figura 6.13: Aspectos de mejora.

Luego de la presentación, los participantes propusieron sugerencias para mejorar la plataforma, las cuales no se llevaron a la práctica debido a limitaciones de alcance del proyecto, pero se pueden considerar como trabajo a futuro. Como por ejemplo, reducir la salida de las ejecuciones y presentarla de mejor manera, mejorar la forma en el cual se envían los estrategias utilizando formulario, y proveer un video explicativo en alguna plataforma audiovisual debido a la gran complejidad técnica que requiere ejecutar la solución. También hubo retroalimentación acerca de la utilidad de la plataforma, donde se sugirió que podría ser no solamente para buscar el índice de conformidad de patrones de microservicios si no para arquitecturas en general, lo que es un gran aporte ya que se puede ver la utilidad de la plataforma más allá del contexto de este proyecto. Por último, hubo intercambios de ideas con los participantes que se reflejaron en los resultados de las encuestas.

Por lo expuesto anteriormente, se cumple el objetivo de validar con expertos en el área la plataforma y comprobar de que cumple con las necesidades de los usuarios, ya que todos los expertos tuvieron calificaciones positivas tanto de experiencia de usuario como de mejora con respecto a la solución propuesta por Vergara.

6.4. Resumen

El capítulo presenta un paso a paso de como ejecutar el flujo principal en la nueva plataforma utilizando la formalización del patrón Circuit Breaker y la implementación *Hystrix* de Netflix, con los respectivos archivos de configuraciones utilizados por Vergara, así como una presentación de los resultados del patrón Circuit Breaker, tanto como para la implementación *Hystrix* de Netflix como para *Resilience4j*, y para el patrón Service Registry con la implementación *Eureka* de Netflix.

Por lo tanto, el objetivo de generar y ejecutar pruebas en el sistema se

valida, ya que se obtuvo un índice de conformidad de 22 % y 66 % para las implementaciones *Hystrix* y *Resilience4j* respectivamente, del patrón Circuit Breaker y un índice de conformidad del 100 % para la implementación *Eureka* del patrón Service Registry, resultados que son equivalentes a los obtenidos por Vergara.

Con los resultados expuestos en las secciones anteriores y el juicio de expertos, se concluye que se cumplieron los objetivos de la planteados inicialmente.

Capítulo 7

Conclusiones y trabajo futuro

En este capítulo se exponen las conclusiones alcanzadas a lo largo del desarrollo de este proyecto. En la sección 7.1 se resume el trabajo realizado y se destacan las principales contribuciones de la solución propuesta. Finalmente, en la sección 7.2 se sugieren posibles mejoras y líneas de trabajo futuro que podrían mejorar la solución propuesta.

7.1. Conclusiones

El objetivo principal de este proyecto fue desarrollar un sistema que facilite la gestión, formalización y evaluación de patrones de microservicios, automatizando el proceso de creación y ejecución de pruebas sobre sus implementaciones. A lo largo del trabajo, se lograron implementar diversas funcionalidades que permitieron cumplir con el objetivo de manera satisfactoria.

Se desarrolló una solución que permite a los usuarios gestionar patrones de microservicios, con sus respectivas formalizaciones en Event-B, utilizar diversas implementaciones de los patrones, y realizar pruebas automatizadas para verificar su conformidad.

Los resultados obtenidos validan la efectividad del sistema propuesto. Al replicar las mismas pruebas de la tesis de maestría de Vergara, utilizando la nueva plataforma, se demuestra que el sistema mantiene la consistencia de los resultados, mejorando la usabilidad y accesibilidad de la solución. Además, la arquitectura modular implementada brinda mecanismos para que el sistema sea escalable y adaptable a futuros requerimientos.

Como principales contribuciones del proyecto, se destacan:

- El diseño y desarrollo de una plataforma para la gestión de patrones de microservicios, que centraliza la evaluación de los mismos.

- La automatización del proceso de generación y ejecución de pruebas, lo que reduce significativamente el esfuerzo manual y mejora la eficiencia del proceso de validación.
- La verificación del sistema a través de la replicación de pruebas previas, confirmando que los resultados obtenidos son consistentes con las expectativas iniciales. Además, la validación del sistema se llevó a cabo utilizando el juicio de expertos.

Estos logros demuestran la viabilidad y efectividad de la solución propuesta, cumpliendo con los objetivos planteados al inicio del proyecto y proporcionando una base sólida para futuras mejoras y extensiones.

7.2. Trabajo a Futuro

A continuación se mencionan ciertas limitaciones del sistema que fueron identificadas tanto en el proceso de especificación de requerimientos, como durante la implementación.

- **Autenticación y Autorización**: Implementar o integrar un sistema para la autenticación y autorización de usuarios, para limitar el acceso al sistema.
- **Validación de datos más estricta**: Se identifica que, a fin de facilitar al usuario el uso del sistema, se puede agregar un sistema de validación de los diversos datos introducidos por el usuario, a modo de ejemplo:
  - Validar el formato de archivo de estrategias, y la coherencia sintáctica de sus atributos.
  - Validaciones sobre el proyecto Wrapper completo provisto por el usuario, verificando que se utilicen efectivamente los puertos disponibles por el sistema.
- **Mejora del sistema de mensajes de error**: Proveer mensajes de error específicos a los usuarios cuando falle algún paso en el flujo de generación o ejecución de pruebas. Este punto se complementa con el punto anterior.
- **Permitir distintas notaciones de formalizaciones**: Actualmente se permiten formalizaciones únicamente en formato Event-B. Se propone como trabajo futuro permitir otros tipos de notaciones. Si bien esto requeriría cambios en el código del sistema, el mismo es suficientemente modular para soportarlo fácilmente. El Módulo de Procesamiento trabaja con Event-B, pero se podría agregar otro módulo para otra notación, o extender el mismo.
- **Permitir a los usuarios proveer implementaciones en otros formatos**: Actualmente los usuarios proveen las implementaciones indicando el nombre de la misma en Dockerhub. Esto se debe a que el sistema puede

fácilmente descargar y ejecutar dicha implementación en un contenedor, lo que facilita que se realicen pruebas sobre implementaciones en cualquier lenguaje, con cualquier librería y sistema, dentro de un contenedor. Se propone extender este mecanismo para que el usuario pueda proveer implementaciones sin necesidad de alojarlas en Dockerhub.

- **Mejoras de usabilidad:** Se proponen las siguientes herramientas a integrar:
  - **Rodin:** Integrar Rodin en conjunto con sus extensiones, para poder crear las formalizaciones dentro de la plataforma.
  - **Estrategias:** Permitir a los usuarios crear y/o completar los archivos de Estrategia dentro de la plataforma.
  - **Proyecto Wrapper:** Permitir a los usuarios completar el proyecto Wrapper dentro de la plataforma.
- **Generación de formalizaciones con inteligencia artificial:** Se propone la generación de formalizaciones en formato Event-B mediante inteligencia artificial, donde a partir de un patrón dado, y especificaciones en lenguaje natural, se genere el mismo.

Referencias

Ably Realtime Ltd. (2024). *Ably realtime messaging.* https://ably.com/. (Accessed: 2024-08-24)

Antel. (2024a). *Elastic cloud.* https://minubeantel.uy/index.php?NAME_PATH=Elastic_Cloud. (Accessed: 2024-08-17)

Antel. (2024b). *Mi nube antel.* https://minubeantel.uy/. (Accessed: 2024-08-17)

Antel. (2024c). *Vps antel.* https://minubeantel.uy/index.php?NAME_PATH=VPS_HOSTING_PATH. (Accessed: 2024-08-17)

Apache. (2024). *Maven.* https://maven.apache.org/. (Accessed: 2024-08-26)

Axios. (2024). *Axios: Promise based http client for the browser and node.js.* https://axios-http.com/. (Accessed: 2024-08-24)

Chris Richardson. (2024a). *Circuit breaker.* https://microservices.io/patterns/reliability/circuit-breaker.html. (Accessed: 2024-09-11)

Chris Richardson. (2024b). *Service registry.* https://microservices.io/patterns/service-registry.html. (Accessed: 2024-09-11)

CLEARSY safety solutions designer. (2024). *B-method.* https://www.clearsy.com/en/thematics/b-method/. (Accessed: 2024-09-10)

Docker. (2024). *Dockerhub.* https://www.docker.com/products/docker-hub/. (Accessed: 2024-09-11)

Docker Inc. (2024). *Get started with docker.* https://docs.docker.com/get-started/. (Accessed: 2024-09-18)

EclEmma Development Team. (2024). *Jacoco maven plugin.* https://www.eclemma.org/jacoco/trunk/doc/maven.html. (Accessed: 2024-08-26)

Event-b.org. (2024). *Event-b and the rodin platform.* https://wiki.eventb.org/index.php/Rodin_Platform. (Accessed: 2024-09-18)

EventB. (2014). *Event-b wiki.* https://wiki.event-b.org/index.php/Main_Page. (Accessed: 2024-09-10)

Figma. (2024). *Figma.* https://www.figma.com/. (Accessed: 2024-08-10)

Formik. (2024). *Formik: Build forms in react.* https://formik.org/. (Accessed: 2024-08-24)

Fowler, M. (2014). *Microservices.* https://martinfowler.com/articles/microservices.html. (Accessed: 2024-07-27)

Heinrich-Heine-University. (2024a). *Prob main.* https://prob.hhu.de/w/index.php?title=Main_Page. (Accessed: 2024-09-10)

Heinrich-Heine-University. (2024b). *Prob test case generator.* https://prob.hhu.de/w/index.php?title=Test_Case_Generation. (Accessed: 2024-09-10)

Heinrich-Heine-University, Institut für Software und Programmiersprachen. (2024a). *Bmotionweb.* https://prob.hhu.de/w/index.php?title=BMotionWeb. (Accessed: 2024-09-18)

Heinrich-Heine-University, Institut für Software und Programmiersprachen. (2024b). *Prob cli.* https://prob.hhu.de/w/index.php/ProB_Cli. (Accessed: 2024-08-17)

i18next. (2024). *i18next: Internationalization for javascript.* https://www.i18next.com/. (Accessed: 2024-08-24)

JaCoCo. (2024). *Jacoco java code coverage library.* https://www.jacoco.org/jacoco/. (Accessed: 2024-08-26)

MDN Web Docs. (2024a). *Cross-origin resource sharing (cors).* https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS. (Accessed: 2024-09-18)

MDN Web Docs. (2024b). *Websockets api.* https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API. (Accessed: 2024-09-18)

Meta. (2024). *React 18.* https://es.react.dev/blog/2022/03/29/react-v18/. (Accessed: 2024-08-24)

Microsoft Corporation. (2024). *Typescript.* https://www.typescriptlang.org/. (Accessed: 2024-08-13)

Mockito. (2024). *Mockito.* https://site.mockito.org/. (Accessed: 2024-08-26)

ModelMapper. (2024). *Modelmapper.* http://modelmapper.org/. (Accessed: 2024-08-26)

Netflix. (2024a). *Eureka.* https://github.com/Netflix/eureka. (Accessed: 2024-09-11)

Netflix. (2024b). *Hystrix.* https://github.com/Netflix/Hystrix. (Accessed: 2024-09-11)

Oracle. (2024). *Java.* https://www.oracle.com/java/. (Accessed: 2024-08-26)

owncloud. (2024a). *owncloud.* https://owncloud.com/. (Accessed: 2024-08-17)

owncloud. (2024b). *pyocclient.* https://pypi.org/project/pyocclient/. (Accessed: 2024-08-25)

Pallets. (2024). *Flask.* https://flask.palletsprojects.com/. (Accessed: 2024-08-17)

PostgreSQL Global Development Group. (2024). *Postgresql jdbc driver.* https://jdbc.postgresql.org/. (Accessed: 2024-08-26)

Project Jupyter. (2024). *Jupyter notebooks.* https://jupyter.org/. (Accessed: 2024-08-05)

Python Software Foundation. (2024). *Python.* https://www.python.org/. (Accessed: 2024-08-17)

Redgate. (2024). *Flyway by redgate.* https://flywaydb.org/. (Accessed: 2024-08-26)

Redhat. (2024). *Istio circuit breaker: When failure is an option.* https://developers.redhat.com/blog/2018/03/27/istio-circuit-breaker-when-failure-is-an-option#breaking__good. (Accessed: 2024-12-01)

resilience4j. (2024). *resilience4j.* https://github.com/resilience4j/resilience4j. (Accessed: 2024-09-11)

Richardson, C. (2018). *Microservices patterns: With examples in java.* Shelter Island, NY: Manning Publications.

Romanovsky, A., y Thomas, M. (2013). *Industrial deployment of system engineering methods.* Eds., Berlin, Heidelberg: Springer, 2013,p.211, ISBN: 978-3-642-33170-1.

Segun Adebayo. (2024). *Chakra ui.* https://v2.chakra-ui.com/. (Accessed: 2024-08-24)

Spring Framework. (2024). *Spring boot.* https://spring.io/projects/spring-boot. (Accessed: 2024-08-26)

TanStack. (2024). *React query: Performant and powerful data synchronization for react.* https://tanstack.com/query/latest. (Accessed: 2024-08-24)

The PostgreSQL Global Development Group. (2023). PostgreSQL 16.2 Release Notes [Manual de software informático]. Descargado de https://www.postgresql.org/docs/release/16.2/ (Accessed: 2024-08-26)

Tom Preston-Werner. (2024). *Redwoodjs.* https://redwoodjs.com/. (Accessed: 2024-08-24)

Vergara, S. (2021a). *Implementation compliance of microservices architectural patterns* (Tesis de Master no publicada). Facultad de Ingeniería - UdelaR.

Vergara, S. (2021b). Implementation compliance of microservices architectural patterns. *Universidad de la República - PEDECIBA Informática*, 139–147. (Circuit Breaker Compilance)

Zustand. (2024). *Zustand: Bear necessities for state management in react.* https://zustand-demo.pmnd.rs/. (Accessed: 2024-08-24)

Anexo A

Manual de usuario

En este anexo de presenta el manual de usuario del sistema, el cual detalla paso a paso cómo utilizar la herramienta a fin de generar y ejecutar pruebas sobre una formalización de un patrón de arquitectura de microservicios, para una implementación dada.

A.1. Crear Patrón

Como se presenta en la figura A.1, en el listado de **Patrones** se selecciona la opción **Crear**, para crear un nuevo Patrón. En caso de desear utilizar un Patrón existente en el sistema, saltear este paso.

Figura A.1: Crear Patrón.

Como se presenta en la figura A.2 se debe completar campos *Nombre* y *Descripción*. Subir Formalización del Patrón en Event-B en un archivo *.eventb*

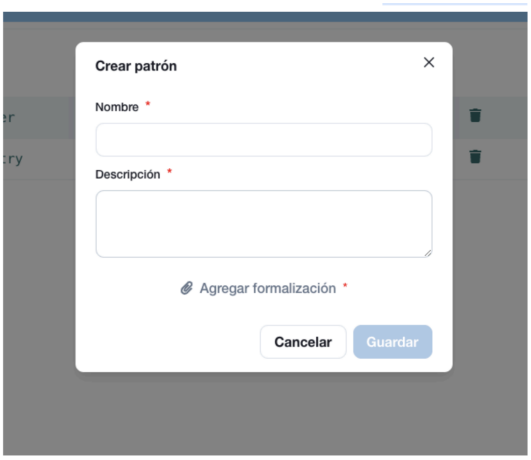

Figura A.2: Detalles del Patrón.

Guardar el Patrón.

A.2. Crear Implementación

Como se presenta en la figura A.3, se puede acceder al detalle del Patrón deseado seleccionando su nombre o icono de detalle.

Figura A.3: Lista de Patrones.

Como se muestra en la figura A.4, en la pantalla de Detalles del Patrón se mostrara la información acerca de las implementaciones, animaciones y formalizaciones del Patrón.

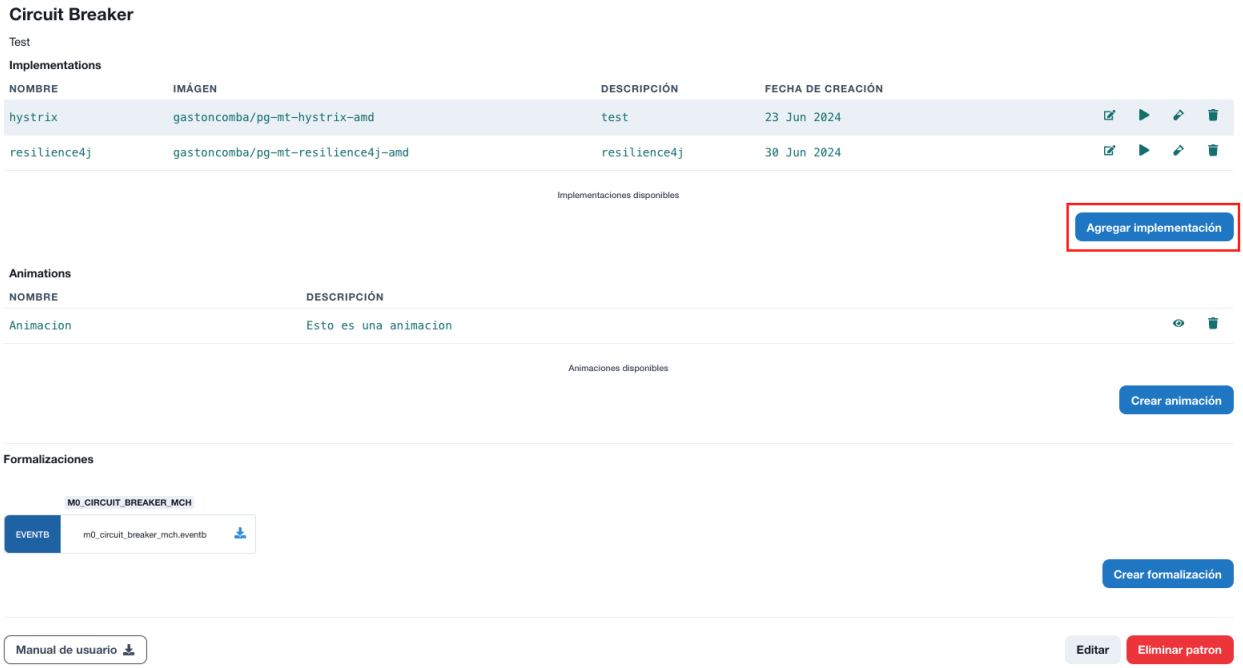

Figura A.4: Agregar Implementación.

Para poder agregar una nueva Implementación para el Patrón se deben completar los campos *Nombre*, *Descripción* y *Nombre de la imagen en Dockerhub*. Este último campo debe ser el nombre de la imagen en Dockerhub,. Se debe utilizar una plataforma *linux/amd64*, y la implementación debe exponerse en el puerto 8090.

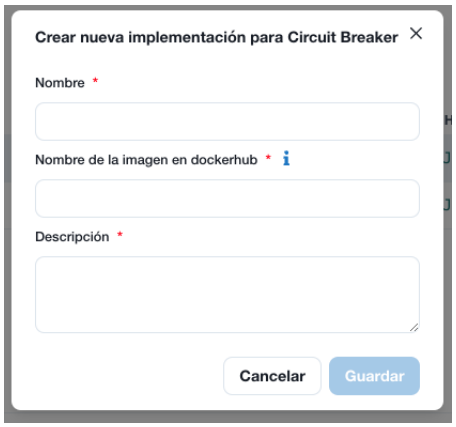

Figura A.5: Crear Implementación.

Guardar la Implementación.

A.3. Ejecutar Pruebas

Para ejecutar pruebas sobre una Implementación, se debe seleccionar el icono de “Ejecutar Pruebas” en la Implementación deseada, como se muestra en la figura A.6.

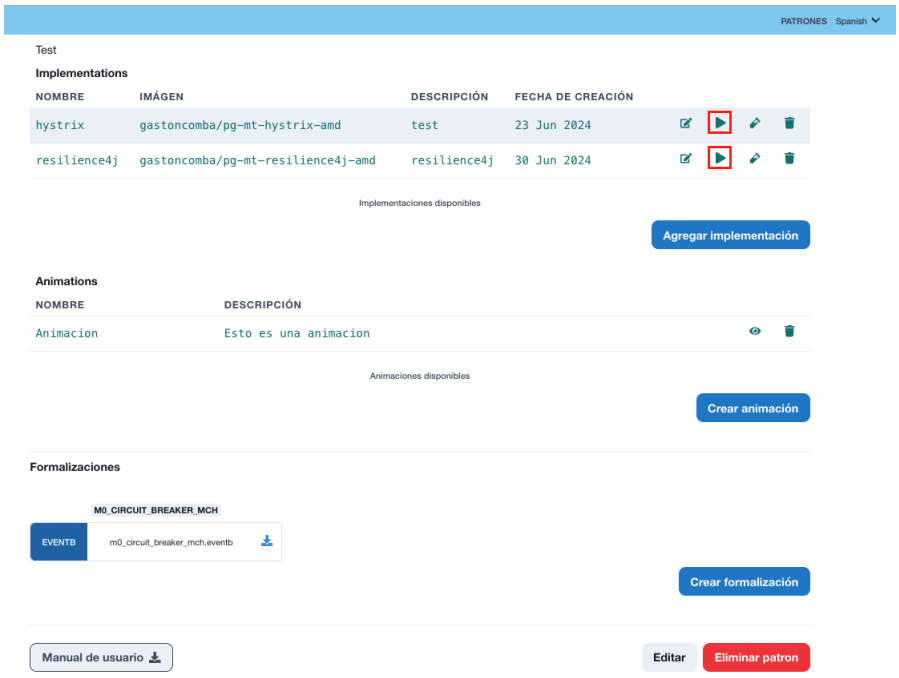

Figura A.6: Ejecutar Pruebas sobre una Implementación.

Como se presenta en la figura A.7, se debe ingresar el *Nombre* de la Prueba y seleccionar una Formalización existente. También se debe proporcionar un archivo de *Estrategia* (ver Anexo B) y un archivo de *Estrategia de instanciación de casos de prueba* (ver Anexo C) ambos en formato YAML.

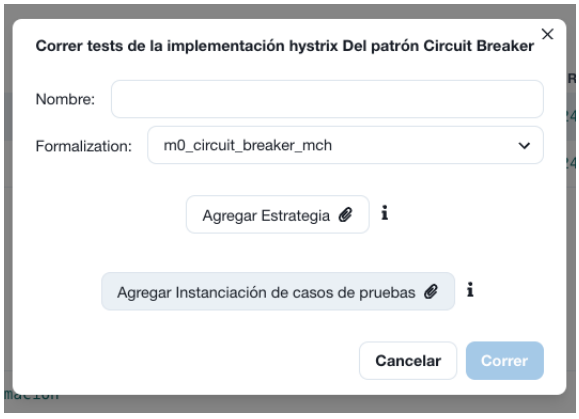

Figura A.7: Configuración de la ejecución de Pruebas.

Seleccionar *Correr* y aguardar a que se complete el proceso. Una vez se complete el proceso, el siguiente paso es descargar el proyecto generado, como se puede ver en la figura A.8

Figura A.8: Descargar proyecto Wrapper.

Al descargar el proyecto, se encontrarán los siguientes archivos y directorios que se presentan en la figura A.9.

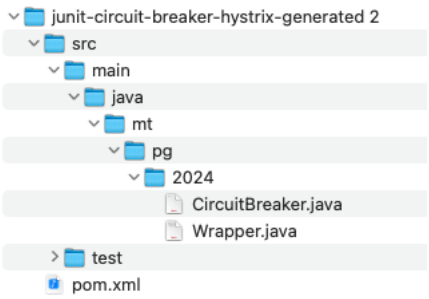

Figura A.9: Estructura proyecto Wrapper.

Para continuar con el proceso es necesario completar el archivo Java Wrapper. Se presenta el Wrapper a completar, y el Wrapper completo generado en el ejemplo:
Wrapper a completar: El Wrapper generado para el patrón Circuit Breaker, tiene siguiente forma:

```
1 public class Wrapper {
2   public void initialisation (Integer AMOUNT_TEST_REQUESTS,
3                               Integer THRESHOLD, Integer
                                    TIMEOUT_PERIOD,
4                               CircuitBreaker circuit_breaker,
                                    Integer consecutive_errors,
5                               Integer test_request_to_go, Integer
                                    time,
6                               Integer timestamp_cb_trips) {
```

```
7       throw new RuntimeException("code not implemented yet");
8     }
9
10    public Boolean isValid (CircuitBreaker circuit_breaker,
11                            Integer consecutive_errors, Integer
                                test_request_to_go,
12                            Integer time, Integer timestamp_cb_trips)
                                 {
13      throw new RuntimeException("code not implemented yet");
14    }
15
16    public void request (Boolean microservice_response) {
17      throw new RuntimeException("code not implemented yet");
18    }
19
20    public void clock() {
21      throw new RuntimeException("code not implemented yet");
22    }
23  }
```

Wrapper completo: En el Listing A.3 se presenta un ejemplo de un Wrapper generado tras ser completado. Es importante observar que se debe utilizar el puerto 8090 para las peticiones.

```
1  public class Wrapper {
2
3    public void initialisation(Integer AMOUNT_TEST_REQUESTS, Integer
         THRESHOLD, Integer TIMEOUT_PERIOD,
4                               CircuitBreaker circuit_breaker,
                                    Integer consecutive_errors,
5                               Integer test_request_to_go, Integer
                                    time, Integer timestamp_cb_trips)
                                    {
6      System.out.println("[test log] [" + System.currentTimeMillis()
           + "] initialisation()");
7      request(true);
8      String requestURL = "http://localhost:8090/reset";
9      OkHttpClient client = new OkHttpClient().newBuilder().build();
10     Request request = new Request.Builder().url(requestURL).method(
           "DELETE", null).build();
11     try {
12       Response response = client.newCall(request).execute();
13       System.out.println("[test log] [" + System.currentTimeMillis
             () + "] initialization status = " + response.code());
14     } catch (IOException e) {
15       throw new RuntimeException("exception when calling rest api
             to reset circuit breaker", e);
16     }
17   }
18
19   public void request(Boolean microservice_response) {
20     String requestURL = "http://localhost:8090/people";
21     if (!microservice_response) {
22       requestURL += "?fail=true";
23     }
24     OkHttpClient client = new OkHttpClient().newBuilder().build();
25     Request request = new Request.Builder().url(requestURL).method(
           "GET", null).build();
26     try {
27       Response response = client.newCall(request).execute();
28       System.out.println("[test log] [" + System.currentTimeMillis
             () + "] request(microservice_response="
29                           + microservice_response + ") = " +
                                response.body().string());
30     } catch (IOException e) {
31       throw new RuntimeException("exception when calling rest api",
              e);
32     }
33   }
34
35   public void clock() {
36     System.out.println("[test log] [" + System.currentTimeMillis()
           + "] starting clock()");
37     try {
38       Thread.sleep(1000);
39     } catch (InterruptedException e) {
40       throw new RuntimeException("exception while trying to sleep",
```

```
            e);
41      }
42      System.out.println("[test log] [" + System.currentTimeMillis()
            + "] ended clock()");
43    }
44
45    public boolean isValid(CircuitBreaker circuit_breaker, Integer
          consecutive_errors,
46                           Integer test_request_to_go, Integer time,
                                 Integer timestamp_cb_trips) {
47      System.out.println("[test log] [" + System.currentTimeMillis()
            + "] isValid_circuit_breaker(" + circuit_breaker + ")");
48      String requestURL = "http://localhost:8090/status";
49      OkHttpClient client = new OkHttpClient().newBuilder().build();
50      Request request = new Request.Builder().url(requestURL).method(
            "GET", null).build();
51      try {
52        Response response = client.newCall(request).execute();
53        String strResponse = response.body().string();
54        Status status = new Gson().fromJson(strResponse, Status.class
              );
55        if (status == null) {
56          return true;
57        } else {
58          if (circuit_breaker.equals(CircuitBreaker.CLOSED)) {
59            return !status.isOpen();
60          } else if (circuit_breaker.equals(CircuitBreaker.OPEN)) {
61            return status.isOpen();
62          } else {
63            return true;
64          }
65        }
66      } catch (IOException e) {
67        throw new RuntimeException("exception when calling rest api
              to validate circuit breaker", e);
68      }
69    }
70  }
```

Una vez agregado el Wrapper completo, seleccionar *Prueba*, como se presenta en la figura A.10.

Figura A.10: Ejecutar pruebas.

A.4. Ver Reporte

Al finalizar el paso anterior, el sistema ejecuta las pruebas y genera un reporte. Esta actividad se realiza en segundo plano, permitiendo al usuario utilizar la aplicación mientras las mismas son ejecutadas. El usuario recibe una notificación cuando el Reporte está listo, o en caso de falla, una alerta de error.

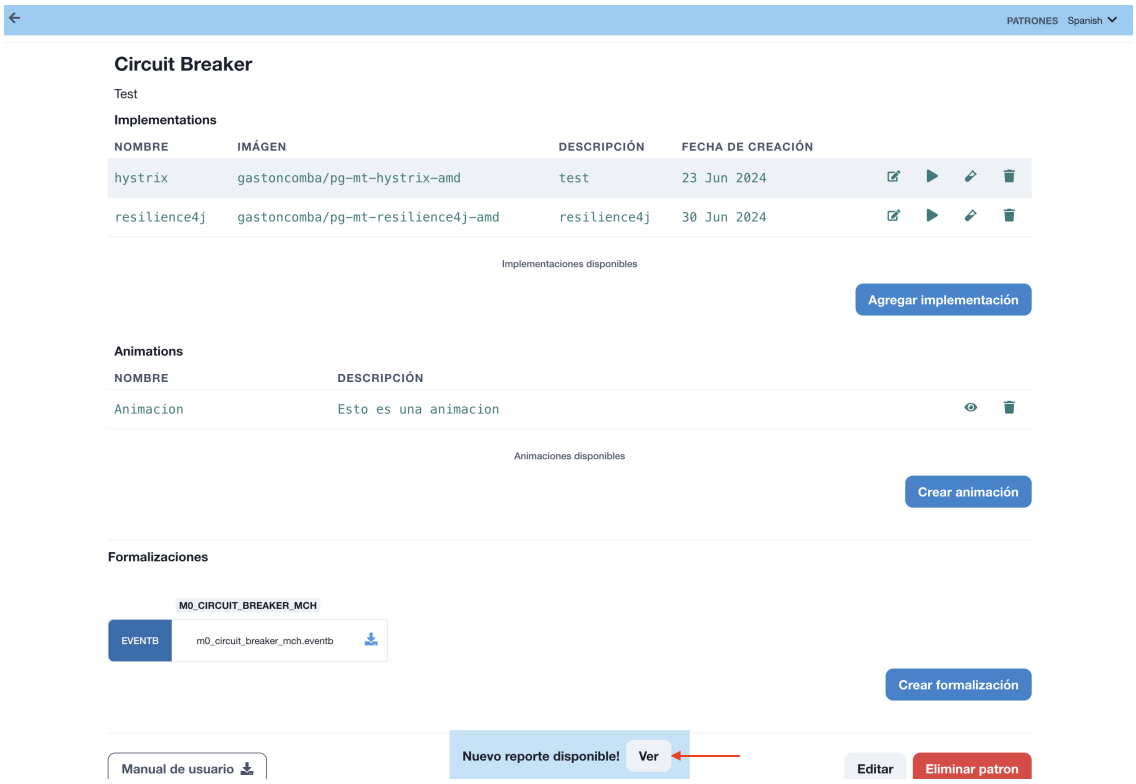

Figura A.11: Ver Reporte.

También se puede acceder al Reporte a través del listado de Reportes de la Implementación.

Anexo B

Archivo de Estrategia

Los archivos de Estrategia son archivos YAML que definen diferentes algoritmos y configuraciones para la generación de casos de prueba abstractos. Esta especificación propuesta por Vergara es utilizada en el Módulo de Procesamiento para brindar al usuario un método para configurar la generación de pruebas, al utilizar ProB. Estos archivos permiten especificar cómo deben generarse y ejecutar las pruebas sobre una formalización de un patrón de arquitectura de microservicios. Existen diferentes tipos de Estrategia, como *cbc*, *sequence* y *random*, cada uno con su propia configuración y propósito.

En estos archivos se definen estrategias para la generación automatizada de casos de prueba abstractos. Entre las distintas estrategias se incluye:

- Validar la cobertura de nodos y transiciones de máquinas de estado.
- Ejecutar secuencias específicas de eventos para verificar comportamientos esperados.
- Generar secuencias aleatorias de eventos para pruebas más exhaustivas y variadas.

Como se mencionó anteriormente, existen varios tipos de estrategias que se pueden utilizar, cada una con sus parámetros y características específicas. Se incluyen:

- ***cbc:*** Generación de casos de prueba basada en restricciones.
- ***sequence:*** Pruebas basadas en secuencias de eventos.
- ***random:*** Pruebas basadas en generación aleatoria.

En el Listing B.1 se presenta un archivo de estrategia a modo de ejemplo.

```yaml
---
apiVersion: 1
kind: cbc
metadata:
  name: nodes coverage
expression:
  - circuit_breaker=CLOSED
  - circuit_breaker=HALF_OPEN
  - circuit_breaker=OPEN
steps: 100
timeout: 60 # in seconds

---
apiVersion: 1
kind: sequence
metadata:
  name: edges coverage
events:
  - request(microservice_response=FALSE)
  - request(microservice_response=FALSE)
  - request(microservice_response=FALSE)
  - clock
  - request(microservice_response=FALSE)

---
apiVersion: 1
kind: random
metadata:
  name: random generation
count: 3
steps: 30
timeout: 60
```

Listing B.1: Example YAML file configuration (Vergara, 2021a)

Se detallan las siguientes características:

cbc - generación de casos de prueba basado en restricciones:

- **Propósito:** Utiliza un solucionador de restricciones para construir secuencias factibles de eventos y se detiene cuando cumple la condición de cobertura.
- **Configuración:** Define una lista de expresiones que representan diferentes estados o condiciones a cubrir. Cada expresión se evalúa hasta un máximo número de pasos hasta que se alcance un tiempo determinado *(timeout)*.

sequence - secuencia de eventos:

- **Propósito:** Ejecuta una secuencia específica de eventos para verificar comportamientos particulares.

- **Configuración:** Define una lista de eventos que deben ejecutarse en el orden especificado.

random - generación aleatoria:

- **Propósito:** Generar secuencias aleatorias de eventos para pruebas más variadas.
- **Configuración:** Define la cantidad de secuencias *(count)*, el número de pasos por secuencia *(step)*, y el tiempo máximo de ejecución *(timeout)*.

Anexo C

Archivo de Estrategia de Instanciación de Casos de Prueba

Es un archivo YAML que define cómo se deben instanciar los casos de prueba abstractos generados a partir de una formalización en Event-B. Este archivo especifica la ubicación del proyecto, los tipos y variables a crear, y los mapeos entre los elementos de Event-B y las clases específicas del lenguaje de programación utilizado. Este archivo sirve de configuración para la transformación de casos de prueba abstractos en casos de prueba concretos y ejecutables, lo cual incluye:

- Mapear constantes, variables y parámetros de Event-B a tipos específicos del lenguaje de programación de la implementación.
- Crear un proyecto estructurado con los casos de prueba concretos utilizando JUnit de Java.

En el Listing C presenta un ejemplo de Archivo de Instanciación de Casos de Prueba para el patrón Circuit Breaker.

```
projectLocation: ./services/xml2junit/java/2
groupId: mt.pg.2024
artifactId: junit-circuit-breaker-hystrix-generated
types:
  - name: CircuitBreaker
    supertype: enum
    values: [CLOSED, OPEN, HALF_OPEN]
variables:
  - name: THRESHOLD
    type: Integer
  - name: AMOUNT_TEST_REQUESTS
    type: Integer
```

```
  - name: TIMEOUT_PERIOD
    type: Integer
  - name: test_request_to_go
    type: Integer
  - name: timestamp_cb_trips
    type: Integer
  - name: consecutive_errors
    type: Integer
  - name: circuit_breaker
    type: CircuitBreaker
  - name: microservice_response
    type: Boolean
  - name: new_circuit_breaker
    type: CircuitBreaker
    ignore: true
  - name: new_consecutive_errors
    type: Integer
    ignore: true
  - name: new_test_request_to_go
    type: Integer
    ignore: true
  - name: new_timestamp_cb_trips
    type: Integer
    ignore: true
  - name: consecutive_errors
    type: Integer
  - name: time
    type: Integer
```

Donde se detallan las propiedades de la siguiente manera:

- *projectLocation:*
  - Propósito: Indica la ruta relativa donde se crea el proyecto.
  - Ejemplo: */services/xml2junit/java/2*
- *groupId:*
  - Propósito: Nombre del artefacto generado.
  - Ejemplo: *junit-circuit-breaker-hystrix-generated*
- *types:*
  - Propósito: Define nuevos tipos a ser creados, como por ejemplo enumerados.
  - Ejemplo:
    - *name:* Circuit Breaker
    - *supertype:* enum
    - *values*: [CLOSED, OPEN, HALF_OPEN]

- *variables:*
  - Propósito: Define las variables utilizadas en los casos de prueba y sus tipos.
  - Ejemplo:
    - *name:* TRESHOLD
    - *type:* Integer