Universidad Zaragoza
1542

Trabajo Fin de Grado

Automatización del aprovisionamiento de recursos en proyectos de microservicios: Diseño e implementación de un MVP

Resource provisioning automation in microservices projects: Design and implementation of an MVP

Autor

José Marín Díez

Director

Daniel Domínguez Guillén

Ponente

José Javier Merseguer Hernáiz

Escuela de Ingeniería y Arquitectura
2023

Resumen

En un sector caracterizado por su constante evolución como el de las tecnologías de la información, aparecen continuamente nuevas necesidades y desafíos. Estos factores acarrean inevitablemente un incremento significativo de la complejidad y el tamaño de los sistemas software. En respuesta a esta tendencia, determinados enfoques arquitectónicos, en particular los microservicios, adquieren cada vez más relevancia debido a su capacidad para establecer sistemas altamente escalables, mantenibles y con un rendimiento superior.

Sin embargo, esta complejidad trae consigo requerimientos adicionales que exigen la implantación de procesos de gestión y desarrollo más sofisticados. En este escenario, la combinación de automatización y desarrollo de software emerge como una estrategia imprescindible para alcanzar el éxito.

Este trabajo se centra principalmente en la automatización de uno de los aspectos más críticos en el desarrollo de dichos sistemas. Concretamente, se presenta la creación de una innovadora herramienta destinada a operar como componente auxiliar en el ciclo de desarrollo de software, enfocándose en la tarea particular de aprovisionamiento de recursos.

Los resultados derivados de la construcción de una primera versión, orientada a cubrir las necesidades básicas, presentan un gran potencial. Las pruebas iniciales evidencian una notable simplificación del proceso de gestión de recursos. Asimismo, este avance ha tenido un impacto positivo en el proceso de desarrollo, provocando mejoras sustanciales en la eficiencia y validando la utilidad del sistema desarrollado.

Índice general

1. Introducción **1**
- 1.1. Motivación . . . 1
- 1.2. Objetivo . . . 1
- 1.3. Alcance . . . 2
- 1.4. Estructura del documento . . . 3

2. Análisis **4**
- 2.1. Necesidades del sistema . . . 4
- 2.2. Requisitos funcionales y no funcionales . . . 6
  - 2.2.1. Recomendaciones para la especificación de requisitos . . . 6
  - 2.2.2. Diccionario de datos . . . 8

3. Diseño **9**
- 3.1. Arquitectura . . . 9
  - 3.1.1. Arquitectura de Puertos y Adaptadores . . . 9
  - 3.1.2. Justificación de la elección de la Arquitectura de Puertos y Adaptadores . . . 11
  - 3.1.3. Arquitectura del sistema . . . 12
- 3.2. Modelo de datos . . . 15
  - 3.2.1. Proceso de transformación de datos . . . 15
- 3.3. Interfaz e integración . . . 16
  - 3.3.1. Interfaz del sistema . . . 17
  - 3.3.2. Integración con herramientas de gestión . . . 17
- 3.4. Diseño de algoritmos . . . 19
  - 3.4.1. Procesamiento de lenguaje natural . . . 19

4. Desarrollo **20**
- 4.1. Aplicación . . . 20
  - 4.1.1. Procesamiento de lenguaje natural . . . 20
  - 4.1.2. Integración con la herramienta de gestión . . . 21
- 4.2. Tecnologías utilizadas . . . 22
  - 4.2.1. Aplicación . . . 23
  - 4.2.2. Base de datos . . . 24
  - 4.2.3. Gestión de contenedores . . . 24
  - 4.2.4. Visualización . . . 24
  - 4.2.5. Otras . . . 25

5. Pruebas **26**
- 5.1. Objetivo de las pruebas . . . 26
- 5.2. Estrategia de las pruebas . . . 26
- 5.3. Entorno de pruebas . . . 27

5.4. Resultado de las pruebas 28

6. Conclusión **30**
6.1. Trabajo a futuro 30

Referencias **35**

Apéndices **39**

A. Dedicación **40**

B. Instrucción para la detección de tareas **41**

C. Modelo de datos extendido **42**

D. Documentación de la API **44**

E. Entorno de Pruebas **45**

F. Casos de Prueba **46**

1 Introducción

La automatización desempeña un papel crucial durante el ciclo de vida del software, especialmente en proyectos de microservicios [1, 2], donde debido a la propia naturaleza de la arquitectura, se requiere la creación y gestión de un gran número de servicios. Esta automatización permite liberar a los desarrolladores de tareas manuales y repetitivas, permitiéndoles enfocarse en tareas más técnicas y estratégicas, acelerando así el proceso de desarrollo. Esto no solo ahorra tiempo y recursos, sino que también reduce el riesgo de cometer errores humanos y mejora la calidad general del software. Por ello, la automatización se convierte en un elemento clave para el éxito de los proyectos software.

1.1. Motivación

Durante este último año, como miembro del equipo encargado de la construcción de un nuevo sistema para una de las grandes empresas del comercio electrónico de España, he podido observar la importancia de agilizar y automatizar diferentes tareas. Esta nueva solución permite a la empresa realizar una migración exitosa de su plataforma de comercio, evolucionando desde una arquitectura monolítica hacia una basada en microservicios.

Dentro del proceso de desarrollo de arquitecturas de microservicios, se identifican una serie de procesos mecánicos y repetitivos que no aportan valor al producto desde la perspectiva del negocio. Sin embargo, esas tareas consumen tiempo y esfuerzo. La naturaleza modular y distribuida de los microservicios son ideales para la automatización de diversos aspectos del desarrollo. Algunas de las tareas candidatas para la automatización son:

- Despliegue y gestión de contenedores. Se refiere al proceso de empaquetar y desplegar nuevos cambios en diferentes entornos.
- Generación de código base. Partiendo de plantillas predefinidas, es posible generar automáticamente la estructura básica del código para la creación de nuevos microservicios.
- Pruebas y compilaciones. Mediante la automatización de las pruebas y compilaciones, cada vez que se realiza un cambio, se puede garantizar la calidad del software a medida que evoluciona.
- Aprovisionamiento de recursos. La automatización puede encargarse del aprovisionamiento y configuración de los recursos necesarios, como bases de datos o sistemas de mensajería.

1.2. Objetivo

El objetivo de este Trabajo Fin de Grado es abordar la creación de un sistema de automatización diseñado para agilizar determinadas tareas del proceso de desarrollo de software. En particular, se centra en la automatización de las tareas de aprovisionamiento de recursos de infraestructura y

generación de código. Estas tareas específicas han sido seleccionadas debido a su nivel de adopción actual, que se encuentra menos avanzada en comparación a otras, las cuales están más consolidadas.

Los potenciales beneficios e impactos derivados de esta solución son múltiples. Se espera un incremento notable en la eficiencia del equipo de desarrollo, así como una reducción significativa de los errores humanos. Adicionalmente, se prevé una disminución en los tiempos y demoras causadas por la dependencia de terceros.

En resumen, el propósito principal de este proyecto es reducir el tiempo dedicado a la realización de tareas y procesos inherentes al ciclo de vida del software en el contexto del desarrollo de microservicios. Al minimizar la dependencia de terceros y proporcionar al equipo una mayor fluidez, los desarrolladores podrán enfocarse en tareas de mayor complejidad, lo cual tendrá un impacto positivo en el resultado final del software.

1.3. Alcance

El alcance de este trabajo es el desarrollo de un Producto Mínimo Viable de una herramienta destinada a integrarse en el proceso de construcción de software. La meta es la creación de un software que permita el aprovisionamiento de recursos y la generación de código, con el fin de automatizar y agilizar el proceso de desarrollo de proyectos de gran tamaño.

El producto se crea con las características mínimas para cumplir su propósito principal. Enfocándose inicialmente en ofrecer una funcionalidad básica de aprovisionamiento de recursos. En el desarrollo de microservicios se precisa de una amplia variedad de recursos, que pueden abarcar tanto elementos de software como de infraestructura.

Este proyecto se enfoca en los recursos más comúnmente empleados, incluyendo la creación y configuración de colecciones en bases de datos NoSQL, tanto en MongoDB [3] como en Couchbase [4]. También se contempla la creación y configuración de topics en Kafka [5], además de la creación de repositorios. Por último, también entra dentro del alcance del sistema la generación de plantillas de código base para facilitar nuevos desarrollos.

Hay algunos conceptos y características que, aunque se mencionan, por razones de tiempo, no entran dentro del desarrollo del MVP. Por ejemplo, se menciona la implementación de un módulo de inteligencia artificial. Si bien este es un campo prometedor, se ha decidido no llevar a cabo esta implementación y apostar por la utilización de un modelo ya existente. Tampoco se lleva a cabo la creación de una interfaz gráfica para el sistema. La elaboración de estos componentes implica un esfuerzo significativo que desvía la atención de la funcionalidad central.

El sistema resultante de este trabajo es un software funcional probado en un entorno de desarrollo. De acuerdo con la escala *Technology Readiness Level (TRL)* [6], que es una medida reconocida para evaluar la madurez de una tecnología, este sistema se encuentra entre los niveles 4 y 5. Esto significa, que el software ha demostrado su capacidad para cumplir con los requisitos establecidos en un entorno de laboratorio, aunque aún requiere pruebas adicionales en un entorno relevante para alcanzar el nivel 5. Con este nivel de madurez el sistema ha superado las etapas iniciales de desarrollo y permite extraer las primeras conclusiones.

La idea es poder validar rápidamente la utilidad de la herramienta y, a partir de la información obtenida, mejorar su desarrollo para posibles futuras versiones.

1.4. Estructura del documento

En este documento, se aborda el desarrollo del sistema mencionado, comenzando con un análisis de las necesidades y requisitos del mismo. Se describe en detalle el diseño de la arquitectura propuesta, su interfaz y su modelo de datos. A continuación, se detalla el proceso de desarrollo del sistema, incluyendo las tecnologías utilizadas. Posteriormente, se expone la metodología y el plan de pruebas realizado. Por último, se extraen algunas conclusiones y se proponen recomendaciones para futuros desarrollos.

2 Análisis

Para garantizar el éxito de cualquier aplicación software, es fundamental contar con una clara comprensión de las necesidades del sistema. En este capítulo, se exploran estas necesidades, se discuten los elementos clave de unos buenos requisitos software y se presenta un listado detallado de requisitos funcionales y no funcionales sobre los que se construye la herramienta.

Este análisis proporciona una serie de beneficios. En primer lugar, se establece una contrato que define las funcionalidades y características que el software debe tener. Esto evita malentendidos futuros y asegura que todas las partes están de acuerdo sobre lo que debe hacer la herramienta.

Además, al revisar los requisitos en una etapa temprana, antes de comenzar el diseño, se pueden identificar y corregir omisiones, malentendidos e incoherencias, lo cual resulta mucho más fácil y menos costoso que hacerlo en etapas avanzadas del ciclo de desarrollo, donde incluso el cambio más mínimo puede implicar un rediseño completo.

2.1. Necesidades del sistema

El software debe satisfacer una serie de necesidades fundamentales para lograr los objetivos establecidos con su desarrollo. En primer lugar, se plantea la necesidad de contar con una **interfaz centralizada** que funcione como punto de entrada para el aprovisionamiento de recursos en los proyectos donde se integre. Esta interfaz tiene la responsabilidad de unificar la administración de múltiples infraestructuras tecnológicas, las cuales son comúnmente empleadas en la creación de microservicios. Específicamente, la herramienta gestiona tecnologías de bases de datos, de mensajería y de colaboración, lo que posibilita una gestión más eficiente de estos recursos.

En línea con esto, se contempla la inclusión de bases de datos NoSQL, como MongoDB y Couchbase, ampliamente utilizadas en arquitecturas de microservicios. La creación de colecciones es algo muy común, ya que para reducir el acoplamiento entre servicios, es habitual seguir enfoques como *Database per service* [7].

Además, se reconoce la relevancia de las plataformas de streaming, como Kafka, para mantener la comunicación entre microservicios. Crear topics en estas plataformas es esencial pero laborioso. Por esta razón, la herramienta también asume la responsabilidad crear y configurar estos topics.

Los repositorios, como Github, Bitbucket o GitLab, son otros recursos esenciales para la colaboración y el desarrollo. Estas plataformas permiten llevar un control de versiones y facilitan la integración y entrega continua (CI/CD) [8]. La herramienta en cuestión asume, dado el nivel de complejidad en la creación y configuración de repositorios, la responsabilidad de manejar este proceso.

En otro contexto, es esencial que la herramienta ofrezca una funcionalidad de generación de código fuente. Esto implica que la herramienta de aprovisionamiento debe tener la capacidad de producir una plantilla de código base que sirva como punto de partida para el desarrollo de un nuevo microservicio.

Otra necesidad crucial es que el sistema sea de baja complejidad, simple en su diseño y funcionamiento, que permita una fácil adaptación a cada proyecto. Una vez instalado, se pretende que el software requiera de un **mantenimiento mínimo**. Dado que se trata de una herramienta auxiliar, se pretende minimizar el tiempo dedicado a su gestión. Esto permitirá evitar complicaciones y trabajo adicional.

Por otro lado, se busca que el software sea **independiente de la tecnología y los recursos** que aprovisionará. Esto significa que debe poder integrarse sin problemas en variedad de proyectos que usen diferentes tecnologías y proveedores. Esta independencia tecnológica permitirá que el software sea más flexible y adaptable, facilitando su integración en entornos tecnológicamente heterogéneos. Para ello, es necesario apostar por un sistema modular, que aísle en la medida de lo posible la lógica de negocio de la infraestructura.

Asimismo, es imprescindible que la herramienta sea **intuitiva y accesible** para todos los miembros del proyecto, independientemente de sus conocimientos sobre infraestructura. Al reducir la barrera de entrada, se fomentará la colaboración y se podrá liberar a parte del personal de algunas de las tareas rutinarias.

Finalmente, para proporcionar una representación visual de estas acciones, se presenta el diagrama de casos de uso de la Figura 2.1. Este diagrama ilustra de manera clara y concisa las diversas funciones y operaciones que están a disposición de los usuarios al utilizar la herramienta de aprovisionamiento.

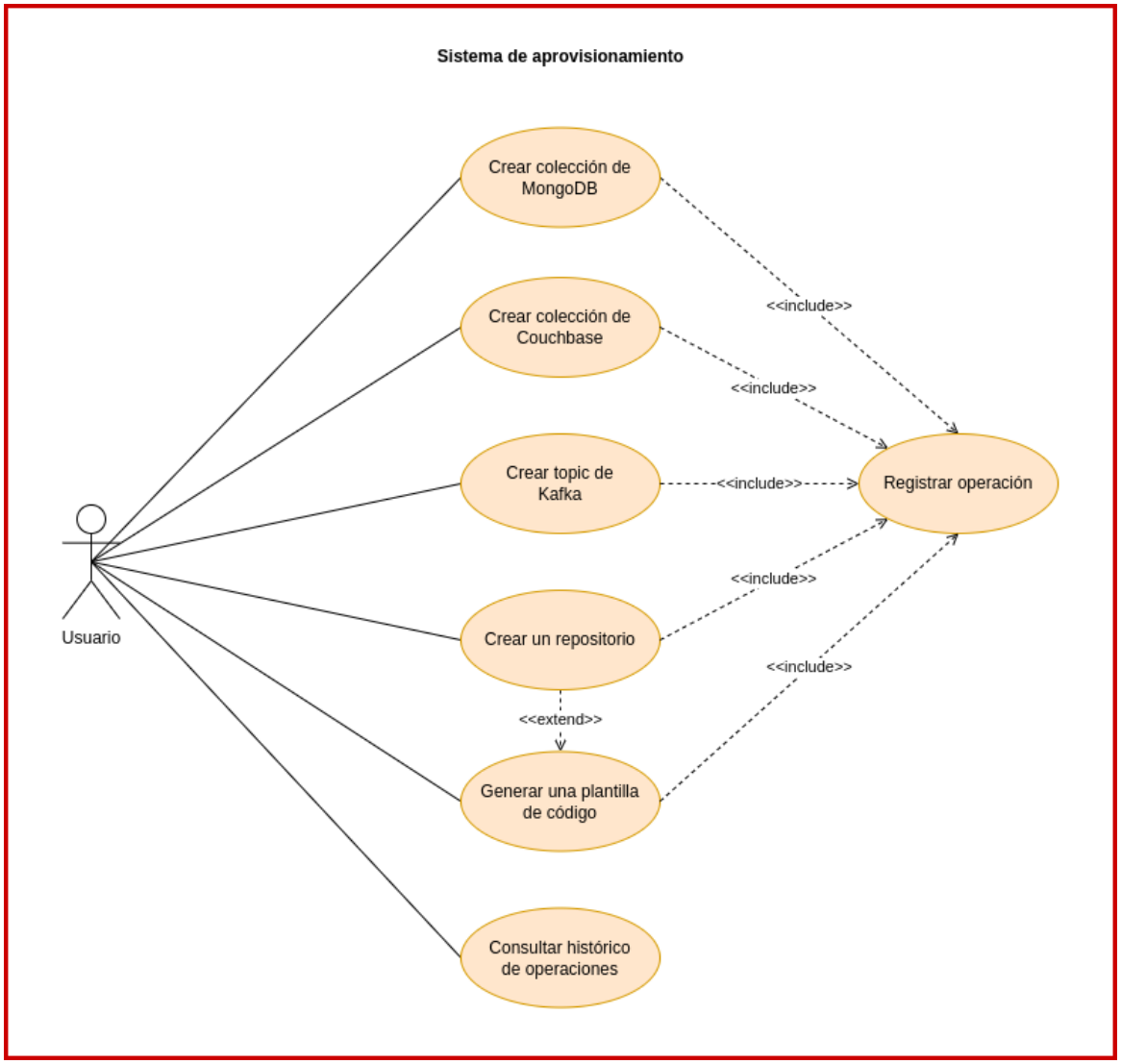


Figura 2.1: Diagrama de Casos de Uso de la aplicación

2.2. Requisitos funcionales y no funcionales

De acuerdo al *IEEE Standard Glossary of Software Engineering Terminology* [9], un requisito se define de la siguiente manera:

1. Una condición o capacidad que necesita un usuario para resolver un problema o alcanzar un objetivo.
2. Condición o capacidad que debe cumplir o poseer un sistema o componente del sistema para satisfacer un contrato, norma, especificación u otros documentos impuestos formalmente.
3. Una representación documentada de una condición o capacidad como en 1 o 2.

Los **requisitos funcionales** se enfocan en las funcionalidades y comportamientos específicos que el sistema debe ofrecer, mientras que los **requisitos no funcionales** definen características y restricciones del sistema relacionadas con la disponibilidad, mantenibilidad, fiabilidad, escalabilidad, etc.

Con base en la identificación de necesidades realizada en la Sección 2.1, se ha elaborado un listado de los principales requisitos funcionales y no funcionales que servirán como guía para el desarrollo del sistema. Estos requisitos se presentan en las Tablas 2.1 y 2.2, respectivamente.

2.2.1. Recomendaciones para la especificación de requisitos

El *IEEE Recommended Practice for Software Requirements Specifications* [10] establece un conjunto de prácticas recomendadas para redactar especificaciones de requisitos de software. Esta práctica recomendada tiene por objeto describir el contenido y cualidades de una buena especificación de requisitos de software (SRS).

Según estas recomendaciones, una buena SRS debe cumplir lo siguiente:

1. Debe definir correctamente todos los requisitos del software. Un requisito de software puede existir debido a la naturaleza de la tarea que debe resolverse o debido a una característica especial del proyecto.
2. No debe describir ningún detalle de diseño o implementación. Estos deben describirse en la fase de diseño del proyecto.
3. No debe imponer restricciones que no estén justificadas por los requisitos del software.

Además, una SRS debe cumplir con las siguientes características:

- **Correcta**: El software cumple todos los requisitos que en ella se establecen.
- **Sin ambigüedades**: Cada requisito debe tener una única interpretación y se deben evitar términos ambiguos.
- **Completa**: La SRS debe incluir todos los requisitos significativos, ya sean relativos a la funcionalidad, el rendimiento o las restricciones de diseño.
- **Consistente**: No debe haber conflictos entre los requisitos dentro de la SRS.
- **Verificable**: Debe ser posible verificar si el software cumple con cada requisito. No se pueden verificar afirmaciones como "funciona bien", ya que es imposible definir los términos "bueno.º "bien".

Tabla 2.1: Requisitos funcionales del sistema

Código	**Descripción**
RF-1	El sistema debe permitir la creación de colecciones de MongoDB.
RF-2	El sistema debe permitir la creación de colecciones de Couchbase.
RF-3	El sistema debe permitir la creación de topics de Kafka.
RF-4	El sistema debe permitir la creación de repositorios.
RF-5	El sistema debe permitir añadir una plantilla de código a un repositorio.
RF-6	El sistema debe crear un nuevo repositorio cuando el usuario solicita añadir una plantilla a un repositorio que no existe.
RF-7	El sistema debe crear una rama en el repositorio con el código de la plantilla al añadirla.
RF-8	El sistema debe permitir al usuario añadir diferentes plantillas de código personalizados al sistema.
RF-9	El sistema debe admitir la configuración por parte del usuario de los recursos a crear. Se debe proporcionar una interfaz para que el usuario ingrese y establezca los valores de configuración pertinentes durante la creación de los recursos.
RF-10	El sistema debe aplicar valores predeterminados para los parámetros no especificados por parte del usuario durante la creación de los recursos.
RF-11	El sistema debe almacenar un registro de todas las operaciones realizadas, incluyendo información como la fecha, hora y detalles específicos de cada acción. Los registros deben ser almacenados en una base de datos designada para tal fin.
RF-12	El sistema debe permitir la consulta de las operaciones realizadas.
RF-13	El sistema debe exponer los servicios de creación de recursos a través de una API REST para facilitar la creación de una interfaz gráfica.
RF-14	El sistema debe suministrar una documentación de la API para facilitar la utilización del sistema.
RF-15	El sistema debe ser capaz de interpretar instrucciones en lenguaje natural para crear recursos, identificando los detalles asociados a dichos recursos.
RF-16	El sistema debe ofrecer una funcionalidad de integración que permita a las aplicaciones de gestión solicitar el aprovisionamiento de un recurso.

Tabla 2.2: Requisitos no funcionales del sistema.

Código	**Descripción**
RNF-1	La API debe seguir la especificación y estándares de OpenAPI superiores a la versión 3.0.0
RNF-2	La documentación de la API debe proporcionar una descripción de todos los endpoints, así como los formatos de solicitud y de respuesta.
RNF-3	Los registros de operaciones deben conservarse durante al menos 3 meses y ser accesibles para fines de auditoría.
RNF-4	El sistema debe poder ser desplegado en cualquier plataforma que admita Docker.
RNF-5	El sistema requiere acceso a internet para su funcionamiento normal y para interactuar con servicios externos.

- **Modificable**: La estructura y el estilo del SRS deben permitir cambios fáciles y consistentes en los requisitos.

2.2.2. Diccionario de datos

Colección Es un grupo de documentos relacionados que comparten una estructura común. Los documentos son registros individuales que almacenan datos en un formato semiestructurado. Las colecciones son análogas a las tablas de las bases de datos relacionales [11] [12].

Topic Es un canal para almacenar y publicar un flujo de datos, donde los productores envían mensajes y los consumidores los reciben, permitiendo la transmisión de información en tiempo real.

Repositorio Es un espacio centralizado donde se almacena, comparte y gestiona el código fuente de un proyecto de software.

Plantilla de código Es un conjunto prediseñado de componentes de código fuente que sigue un modelo común. Proporciona una estructura inicial y predefinida que los desarrolladores pueden seguir y evitar comenzar desde cero el desarrollo al crear nuevos microservicios.

3 Diseño

En este capítulo, se introduce la arquitectura de Puertos y Adaptadores como enfoque central del diseño. Posteriormente, se detalla su aplicación en la creación de un sistema adaptable a diferentes tecnologías y entornos. Asimismo, se proporciona una descripción del modelo de datos, junto con el proceso de conversión entre sus múltiples representaciones. Además, se aborda la interfaz del sistema y su integración con herramientas de gestión. Por último, se introduce la incorporación de un componente de procesamiento de lenguaje natural.

3.1. Arquitectura

3.1.1. Arquitectura de Puertos y Adaptadores

La arquitectura de puertos y adaptadores es un enfoque que destaca por la construcción de sistemas altamente adaptables a diferentes tecnologías y entornos. Para lograr este propósito, la solución está en la separación entre la lógica de negocio y el código encargado de interactuar con servicios externos e infraestructura.

Dicha arquitectura (véase Figura 3.1) identifica explícitamente tres bloques fundamentales en un sistema [13]:

- La **interfaz de usuario**, que permite la interacción con el sistema.
- El **núcleo de la aplicación**, que utiliza la interfaz de usuario para ejecutar la lógica de negocio y hacer que las cosas realmente sucedan.
- El código de **infraestructura**, cuya función es conectar el núcleo de la aplicación con sistemas externos, como bases de datos, motores de búsqueda o servicios de terceros mediante APIs.

El elemento esencial de esta arquitectura es el núcleo de la aplicación. Es el código que permite al sistema cumplir con su propósito. Aunque puedan ser utilizadas diferentes interfaces de usuario (GUI, CLI, API, etc) y diferente infraestructura (SQL, NoSQL, SMS, Email, etc), el código que realmente realiza las tareas es el mismo y se encuentra en el núcleo de la aplicación.

Puertos

En una arquitectura de puertos y adaptadores, los puertos son los puntos de entrada y salida que permiten a la aplicación comunicarse con sistemas externos. Son las interfaces a través de las cuales el núcleo de la aplicación interactúa con el exterior. Estos puertos se diseñan de manera ajena a la tecnología utilizada por los sistemas externos, permitiendo a la aplicación ser utilizada por dichos sistemas, sin depender de sus detalles técnicos.

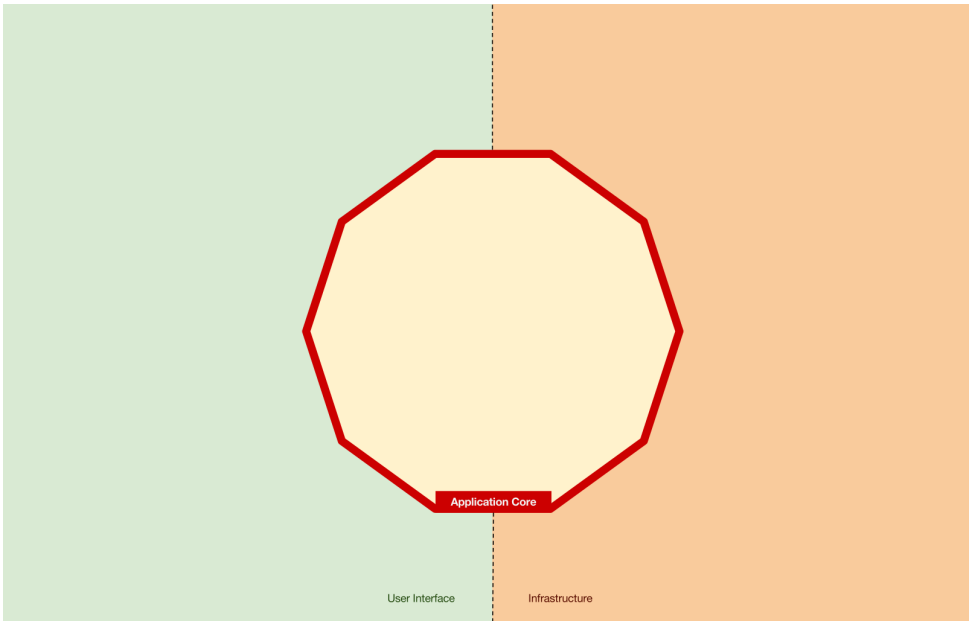


Figura 3.1: Bloques de la arquitectura de puertos y adaptadores

Adaptadores

Los adaptadores son componentes que facilitan la comunicación entre la aplicación y los sistemas externos [14]. Su tarea es adaptar las solicitudes y datos recibidos desde sistemas externos al formato interno del núcleo, y viceversa, convirtiendo las respuestas generadas por el núcleo de la aplicación al formato esperado por los sistemas externos. De esta manera, los adaptadores permiten a la aplicación comunicarse de manera transparente con distintos sistemas.

Existen dos tipos de adaptadores: los Adaptadores Primarios, que le dicen a la aplicación que haga algo, y los Adaptadores Secundarios, que son invocados por la aplicación para hacer algo.

Los **Adaptadores Primarios** (ver Figura 3.2) actúan como envoltorios alrededor de un puerto, utilizando dicho puerto para comunicar al núcleo de la aplicación qué acciones debe realizar. En esencia, transforman solicitudes provenientes de mecanismos de entrada en llamadas a métodos del núcleo de la aplicación.

Un ejemplo es una aplicación que necesita recibir peticiones externas de dos opciones: mediante una API REST y mediante línea de comandos (CLI). Para ambas opciones, se crea un adaptador específico para recibir y procesar los datos entrantes.

En el caso de la comunicación a través de una API REST, el adaptador será el encargado de recibir las solicitudes HTTP provenientes de la API, interpretar los datos y convertirlos en llamadas a los métodos correspondientes del núcleo de la aplicación.

Por otro lado, para la conexión mediante CLI, el adaptador será el responsable de recibir las instrucciones desde la interfaz de línea de comandos, interpretar los datos y pasarlos al núcleo de la aplicación mediante llamadas a los métodos adecuados. De esta forma, la aplicación podrá recibir y procesar tanto solicitudes de la API REST como de la CLI, permitiendo una comunicación versátil con el núcleo de la aplicación.

En cambio, los **Adaptadores Secundarios** (ver Figura 3.3) no envuelven un puerto, sino que implementan una interfaz específica (puerto) que luego es utilizada por el núcleo de la aplicación para interactuar con servicios o tecnologías externas.

Por ejemplo, dada la necesidad de una aplicación de persistir datos, se crea una interfaz que satisfaga

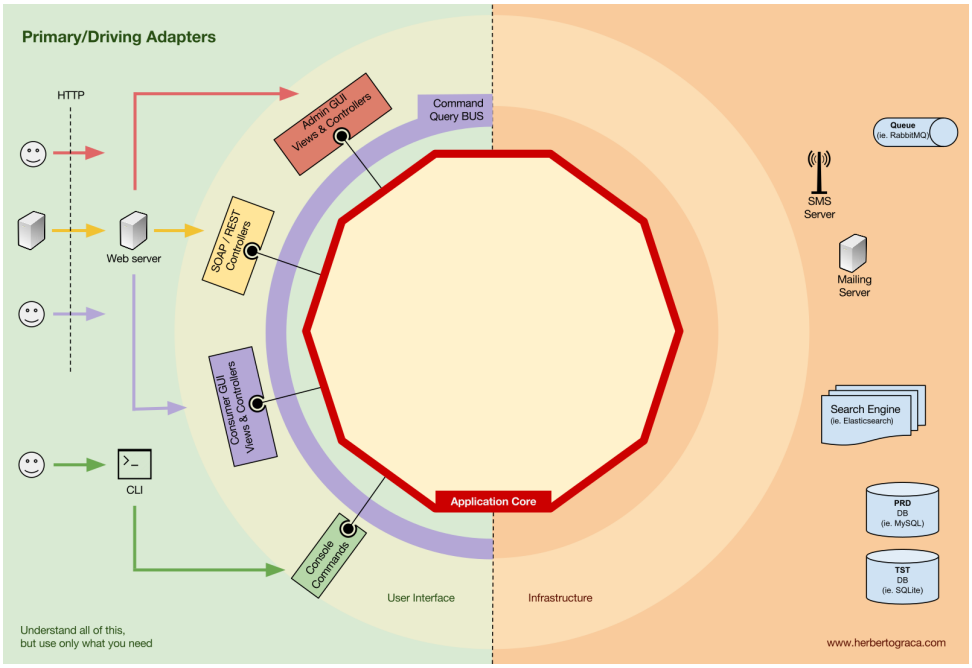


Figura 3.2: Adaptadores Primarios en la arquitectura de puertos y adaptadores

las necesidades. Con métodos, por ejemplo, para guardar y borrar datos. A partir de ahí, siempre que la aplicación requiera guardar o eliminar datos, será necesario un adaptador que implemente la interfaz de persistencia previamente definida.

Si se quiere persistir los datos en una base de datos PostgreSQL, entonces, habrá que crear un adaptador específico para PostgreSQL que implemente la interfaz de persistencia y contenga los métodos necesarios para guardar y eliminar filas en una tabla.

Por otro lado, si se opta por utilizar otro tipo de sistema de base de datos, como MongoDB (un sistema NoSQL), bastaría con crear un nuevo adaptador que implemente la interfaz de persistencia y esté diseñado específicamente para trabajar con MongoDB, incluyendo los métodos necesarios para guardar y eliminar documentos en una colección.

En resumen, los puertos son interfaces que definen la forma en que el núcleo del sistema interactúa con el exterior, mientras que los adaptadores implementan dichas interfaces para conectarse con tecnologías concretas. Esta modularidad y desacoplamiento permite al sistema adaptarse a diferentes tecnologías y servicios sin necesidad de modificar el núcleo del mismo.

Esta arquitectura resulta especialmente beneficiosa cuando se busca adaptar el sistema a múltiples entornos o cuando se prevén cambios frecuentes debido a requisitos cambiantes o nuevas tecnologías.

3.1.2. Justificación de la elección de la Arquitectura de Puertos y Adaptadores

La decisión de aplicar esta arquitectura está motivada por las necesidades identificadas en la Sección 2.1. La arquitectura de puertos y adaptadores ofrece una serie de beneficios que la hacen especialmente apropiada para el contexto de la herramienta que se busca crear.

En primer lugar, **proporciona una interfaz centralizada** a través de los puertos, que actúan como puntos de entrada y salida del sistema. Esto permite gestionar toda la infraestructura tecnológica desde un único lugar, lo que facilita el control y el aprovisionamiento de recursos.

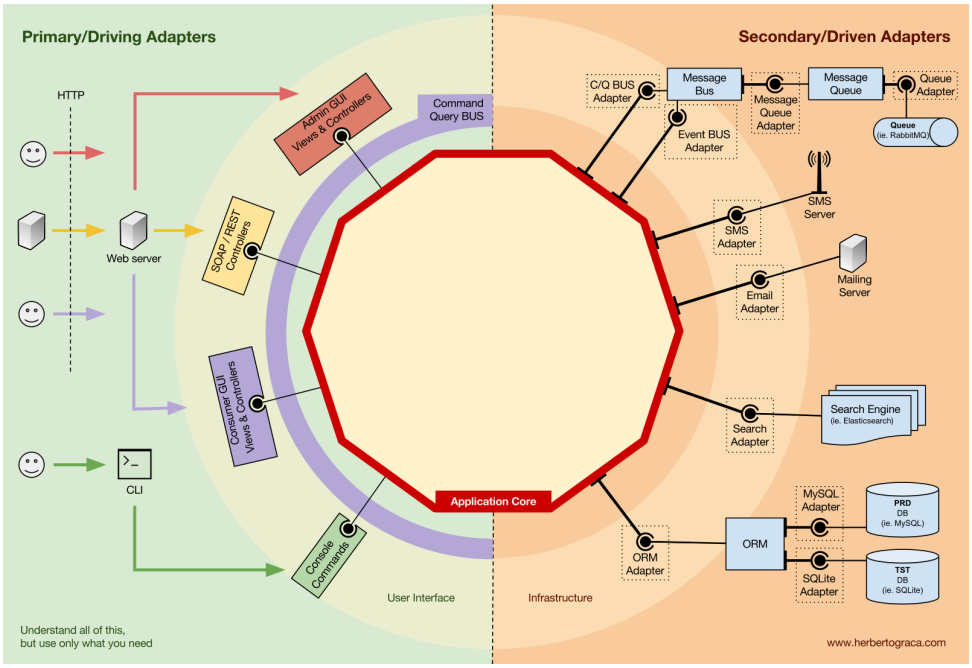


Figura 3.3: Adaptadores Secundarios en la arquitectura de puertos y adaptadores

Gracias a la separación de la lógica de negocio del resto de componentes, se consigue aislar y proteger el núcleo del sistema de las dependencias externas y la infraestructura. De igual manera, los adaptadores **permiten a la aplicación funcionar con diferentes tecnologías**, al tiempo que se mantienen desacoplados los detalles de implementación del aprovisionamiento de recursos. Esta característica permite a la aplicación integrarse fácilmente en diversos proyectos con proveedores y tecnologías heterogéneas sin requerir cambios significativos en su núcleo.

Además, la arquitectura ayuda a alcanzar el objetivo de desarrollar un **sistema mantenible y extensible**. Al dividir el sistema en capas bien definidas y con responsabilidades específicas, y desacoplarlo de los componentes externos, se facilita la modificación o reemplazo de estos últimos sin afectar al núcleo de la aplicación, facilitando el mantenimiento y minimizando el impacto derivado de futuras modificaciones. Esto también facilita las pruebas del sistema.

En resumen, la arquitectura de puertos y adaptadores responde adecuadamente a las necesidades del sistema que se pretende desarrollar. Su capacidad para adaptarse a diferentes tecnologías y entornos, para proporcionar una interfaz centralizada y para dotar a la aplicación de un fácil mantenimiento, la convierten en una solución óptima para la herramienta de automatización de recursos en proyectos de microservicios.

3.1.3. Arquitectura del sistema

Siguiendo los conceptos y pautas descritas en la sección anterior, se ha diseñado la estructura de la aplicación. Este diseño, basado en la arquitectura de puertos y adaptadores, potencia la adaptabilidad y la independencia tecnológica, aspectos fundamentales del sistema.

El diseño y la estructuración de la arquitectura tienen como objetivo principal cumplir con los requisitos y las necesidades identificadas del proyecto. La Figura 3.4 ilustra cómo se han materializado los conceptos en términos de la arquitectura.

A continuación, se presenta cómo cada adaptador se relaciona con un puerto de entrada o salida y facilita la comunicación entre el núcleo y el entorno externo.

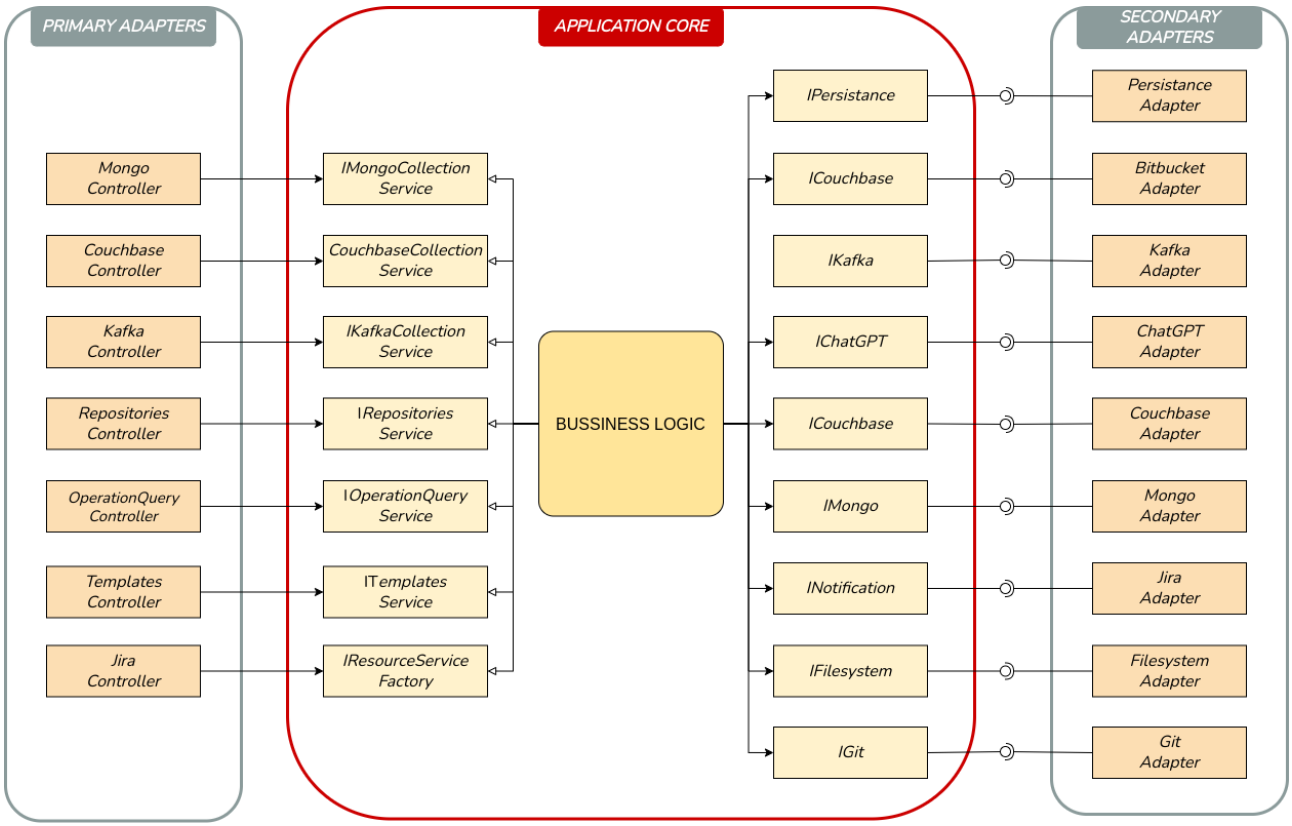


Figura 3.4: Diagrama de la arquitectura basada en puertos y adaptadores

Adaptadores primarios y puertos de entrada

Los adaptadores primarios cumplen con la función de permitir la interacción con la interfaz de usuario y convertir las solicitudes en llamadas al núcleo de la aplicación definidas por los puertos de entrada. A continuación, se presenta la lista de adaptadores primarios junto con sus respectivos puertos de entrada:

- **Adaptadores API REST**: Estos adaptadores reciben las solicitudes provenientes de la API REST y las traducen en objetos y llamadas a métodos específicos del núcleo para ejecutar las operaciones correspondientes.
  - *MongoController*: Sirve como punto de entrada para operaciones relacionadas con MongoDB. Actúa como intermediario para la creación de colecciones, adaptando las operaciones definidas por el puerto *IMongoCollectionService*.
  - *CouchbaseController*: Gestiona solicitudes asociadas a Couchbase. Hace posible la creación de colecciones, funcionando como enlace a través del puerto de entrada *ICouchbaseCollectionService*.
  - *KafkaController*: Facilita operaciones en el broker de Kafka. Encapsula la creación de topics mediante el uso del puerto de entrada *IKafkaTopicService*.
  - *RepositoriesController*: Maneja solicitudes que implican operaciones en repositorios externos, como Bitbucket. Facilita la creación de repositorios a través del puerto de entrada *IRepositoriesService*.
  - *TemplatesController*: Transforma las solicitudes para la generación de código fuente mediante el uso del puerto de entrada *ITemplatesService*.

    - *OperationQueryController*: Se enfoca en consultas a la base de datos de la aplicación. Facilita las acciones de búsqueda de operaciones a través del puerto de entrada *IOperationQueryService*.
- **Adaptador Webhook**: Permite la integración del sistema con aplicaciones de gestión al transformar los datos recibidos por Webhooks en acciones entendibles por el núcleo de la aplicación. Se explica con más detalle en la Sección 3.3.2.
    - *JiraController*: Utilizado para la interacción con Jira. Traduce las peticiones realizadas por Jira en operaciones que la aplicación puede procesar. Facilita la creación de recursos a través de llamadas a los métodos definidos por el puerto de entrada *IResourceServiceFactory*.

Adaptadores secundarios y puertos de salida

Los adaptadores secundarios son responsables de interactuar con la infraestructura y sistemas externos. A continuación, se presentan los adaptadores secundarios y los puertos de salida que implementan:

- **Adaptadores de base de datos**:
    - *PersistanceAdapter*: Administra la persistencia de los datos de la aplicación. Transforma las operaciones definidas por *IPersistance* en acciones compatibles con la API específica del sistema de bases de datos utilizado.
- **Adaptadores de sistemas externos**:
    - *BitbucketAdapter*: Se encarga de la comunicación con el sistema de control de versiones. Traduce las operaciones definidas en la aplicación a llamadas compatibles con la API de Bitbucket, a través del puerto de salida *IRepository*.
    - *KafkaAdapter*: Posibilita la comunicación con Kafka. Transforma las operaciones definidas en el puerto *IKafka* en acciones ejecutables mediante la API de Kafka.
    - *ChatGPTAdapter*: Facilita la comunicación con ChatGPT. Convierte las solicitudes y respuestas definidas en el puerto *IChatGPT* en llamadas que la API de OpenAI puede entender.
    - *CouchbaseAdapter*: Encargado de interactuar con Couchbase. Adapta las operaciones del puerto de salida *ICouchbase* a la API de Couchbase.
    - *MongoAdapter*: Simplifica la comunicación con MongoDB. Adapta las operaciones del puerto *IMongo* a la API de MongoDB.
    - *JiraAdapter*: Permite la interacción del sistema con la API de Jira.
    - *FilesystemAdapter*: Gestiona el sistema de ficheros, permitiendo a la aplicación interactuar con él. Traduce las operaciones definidas en *IFileSystem* a llamadas al sistema operativo.
    - *GitAdapter*: Ajusta las operaciones de *IGit* para trabajar con el sistema de control de versiones Git. Traduce las operaciones en comandos comprensibles por Git.

Flujo de ejecución

El proceso de ejecución de la aplicación sigue el camino que se detalla en la Sección 3.1.1. Cuando un usuario interactúa con la interfaz de usuario, esta acción genera una solicitud que se dirige al sistema. Esta solicitud se canaliza a través de un adaptador primario, el cual a su vez llama al núcleo de la aplicación. En el núcleo se llevan a cabo las operaciones de aprovisionamiento. Para hacer esto, el núcleo se comunica con los adaptadores secundarios que a su vez establecen interacciones con la base de datos de la aplicación y con diversos componentes de infraestructura externos (MongoDB, Couchbase, Kafka, etc.). Una vez se completa la operación, el resultado se devuelve a través de los mismos adaptadores, pasando nuevamente por el núcleo y finalmente presentándose al usuario a través de la interfaz de usuario.

3.2. Modelo de datos

El modelo de datos de una aplicación se refiere a la **representación estructurada de la información** que maneja y procesa. Para esta aplicación, como se detalla en el diagrama del Apéndice C, se opta por continuar con un enfoque modular al dividir el modelo de datos en tres representaciones distintas: el modelo de datos interno, el modelo de datos de la base de datos y el modelo de datos de transferencia. Cada uno de estos modelos desempeña un papel específico en el funcionamiento de la aplicación.

El **modelo de datos interno**, se encuentra en el núcleo de la aplicación. Este modelo define las entidades clave de la lógica de negocio, sus reglas de validación y otros elementos que rigen la aplicación. El modelo de datos interno es independiente de las tecnologías de almacenamiento y presentación, lo que garantiza que la lógica de negocio no se vea comprometida por cambios externos al núcleo de la aplicación.

El **modelo de datos de la base de datos** representa la estructura para el almacenamiento y consulta de datos de la aplicación. Este modelo se basa en el modelo de datos interno, pero está diseñado y optimizado para satisfacer las necesidades de persistencia y recuperación en la base de datos. La separación entre el modelo de datos de la base de datos y el modelo de datos interno aísla la lógica de negocio de la tecnología de la base de datos (Relacional, NoSQL, Grafo).

El **modelo de datos de transferencia**, actúa como puente entre los sistemas externos y el núcleo de la aplicación. Los DTOs (*Data Transfer Objects*) encapsulan y transmiten la información necesaria para realizar las operaciones, evitando así la transferencia innecesaria de datos, además de la exposición directa del modelo interno. La utilización de DTOs también facilita la adaptación a cambios en los requisitos de presentación sin afectar la lógica del negocio en el núcleo de la aplicación.

3.2.1. Proceso de transformación de datos

La comunicación entre distintos bloques se logra mediante el uso de adaptadores y mappers. Cada adaptador de la aplicación, se encarga de convertir los datos del modelo específico de su tecnología al modelo de datos interno del núcleo, y viceversa [15].

El proceso de transformación comienza cuando los datos llegan a la capa de presentación. Estos datos se representan utilizando un modelo de datos de transferencia (DTOs). Luego, los adaptadores de la capa de presentación, mediante el uso de mappers, transforman los DTOs al modelo de datos interno. Una vez que los datos son procesados en el núcleo de la aplicación, los adaptadores realizan la transformación inversa: emplean mappers para convertir los datos desde el modelo de datos interno

en DTOs de respuesta. De manera similar, los adaptadores de la capa de infraestructura gestionan la transformación entre el modelo de datos interno y el modelo requerido para llevar a cabo sus operaciones.

Por ejemplo, para el caso en el que la aplicación recibe una solicitud HTTP a través de la API REST. Los datos de esta solicitud necesitan ser traducidos de JSON a objetos de dominio comprensibles para la lógica de la aplicación. Cuando la aplicación necesita responder a esta solicitud, los objetos de dominio se convierten nuevamente a formato JSON. De manera análoga, cuando la aplicación necesita almacenar datos en base de datos, los objetos de dominio se transforman en los modelos adecuados, como colecciones para bases de datos NoSQL o tablas para bases de datos SQL. Esto también aplica en sentido inverso para las operaciones de consulta en la base de datos.

El siguiente diagrama (Figura 3.5) ilustra cómo interactúan y se relacionan las tres representaciones del modelo de datos en la aplicación.

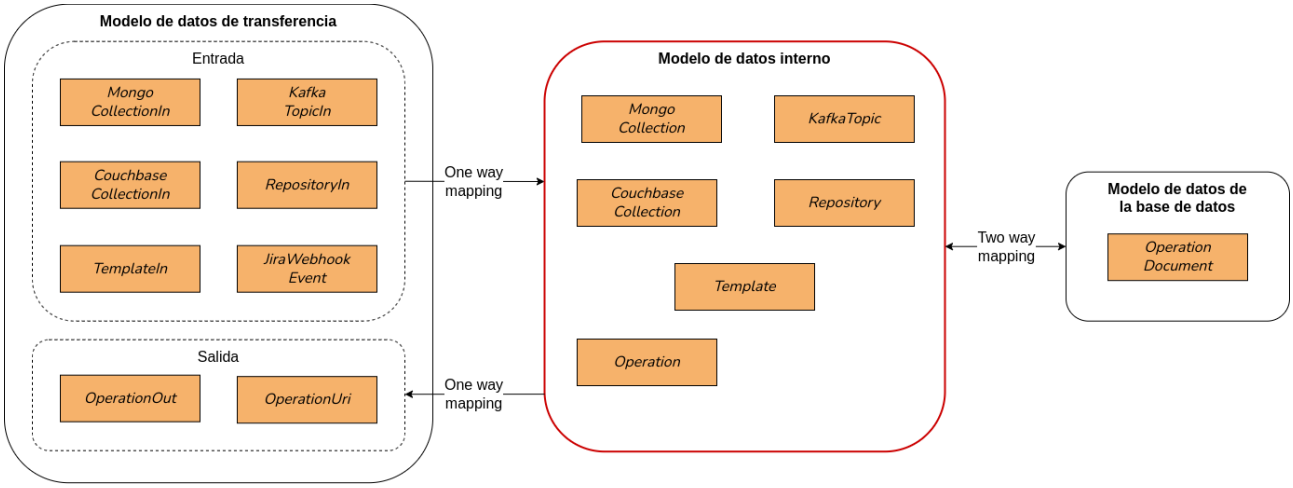


Figura 3.5: Modelo de datos: Interacción entre modelo interno, de la base de datos y de transferencia

3.3. Interfaz e integración

En cualquier aplicación software, la interfaz desempeña un papel fundamental al ser el medio a través del cual los usuarios y otros sistemas se comunican con ella. La función de las interfaces es facilitar la interacción del usuario, mejorar su experiencia y permitir la conexión con otros sistemas. En definitiva, potenciar la utilización del sistema.

Existen diversos tipos de interfaces, cada uno con un propósito específico. Dentro de este proyecto, se distinguen dos tipos de interfaces: la interfaz del sistema y la interfaz de usuario.

La **interfaz del sistema** es el canal a partir del cual la aplicación permite a otros sistemas comunicarse y colaborar con ella. Para este proyecto, la interfaz del sistema se presenta como una API REST.

Por otro lado, la **interfaz de usuario** permite a los usuarios finales interactuar con el software de manera visual e intuitiva. En este caso, en lugar de desarrollar una interfaz gráfica propia, se ha decidido apostar por la integración con herramientas ya existentes.

3.3.1. Interfaz del sistema

La API REST permite la ejecución de operaciones a través de una serie de endpoints. Cada uno de estos endpoint se corresponde con una funcionalidad específica del sistema y está diseñado para recibir y enviar datos en formato JSON.

Cumpliendo con el requisito *RF-14*, para facilitar la comprensión, se provee una documentación técnica junto con la API REST. Esta documentación se encuentra en formato OpenAPI e incluye la descripción de todos los endpoints, así como los formatos de solicitud y de respuesta. Además, mediante el uso de tecnologías como Swagger, es posible generar una interfaz web que permite visualizar e incluso probar la API REST (véase Apéndice D).

A continuación, se presenta un breve resumen de los endpoints de la aplicación para el aprovisionamiento de recursos:

- **POST /resources/mongo/databases/**{**database_name**}: Crea una colección en una base de datos MongoDB.
- **POST /resources/kafka/topics/**: Crea un topic en un broker de Apache Kafka.
- **POST /resources/couchbase/buckets/**{**bucket_name**}: Crea una colección en un bucket de Couchbase.
- **POST /resources/repositories/projects/**{**project_key**}: Crea un repositorio remoto en un proyecto específico.
- **POST /resources/templates/**{**template_type**}: Genera una plantilla específica y/o la carga en un repositorio remoto.
- **GET /operations/**: Obtiene una lista de operaciones realizadas por el sistema
- **GET /operations/**{**operation_id**}: Obtiene los detalles de una operación concreta.

En resumen, la API REST permite a otras aplicaciones y servicios aprovechar las capacidades del sistema desarrollado, permitiendo la creación de soluciones personalizadas que se adapten a las necesidades específicas de los proyectos y organizaciones.

3.3.2. Integración con herramientas de gestión

En el ámbito de los proyectos de software, especialmente en aquellos de envergadura considerable, surge un gran desafío: su gestión. Para asegurar el éxito de estos proyectos, es una práctica común recurrir a herramientas de gestión especializadas. Estas herramientas facilitan la planificación de tareas, el seguimiento de los avances y la gestión de incidencias en todas las etapas del ciclo de desarrollo.

Conforme al requisito funcional *RF-16*, resulta esencial que el sistema de aprovisionamiento de recursos pueda integrarse con estas herramientas de gestión. En la línea con esto, en lugar de crear una interfaz gráfica de cero, se opta por la integración con herramientas ampliamente reconocidas en el ámbito de proyectos de software, como GitHub Projects, GitLab Projects, Trello o Jira. Esta decisión se fundamenta en la idea de que los equipos de desarrollo ya están familiarizados con estas plataformas y las utilizan de manera regular para gestionar sus proyectos.

El propósito de esta integración es simplificar la experiencia de los usuarios finales. De esta manera, se evita que los equipos de desarrollo tengan que aprender a usar una nueva herramienta, ya que el sistema se convierte en una extensión del entorno de trabajo que usan habitualmente. Esta

estrategia no solo reduce la curva de aprendizaje, sino que también elimina la necesidad de alternar entre diferentes interfaces, lo cual mejora la eficiencia y la productividad.

Aunque existen diversas herramientas disponibles en el mercado, todas comparten similitudes en cuanto a su integración con servicios externos. Estas herramientas principalmente ofrecen dos métodos de comunicación y colaboración con otros sistemas:

1. **REST APIs**: Permiten la interacción con la plataforma de manera programática. A través de estas APIs, es posible consultar, crear, actualizar y administrar tareas. Esto habilita a aplicaciones externas a comunicarse con la aplicación de gestión.
2. **Webhooks**: Son notificaciones automáticas enviadas desde la plataforma de gestión a una URL específica en respuesta a eventos predefinidos, como la creación o modificación de una tarea. Los Webhooks permiten a aplicaciones externas recibir información en tiempo real desde la plataforma de gestión.

Estos dos mecanismos posibilitan una comunicación bidireccional entre la plataforma de gestión y el sistema de aprovisionamiento de recursos, garantizando que ambos sistemas puedan colaborar para cumplir con el objetivo.

El diagrama de secuencia de la Figura 3.6 ilustra cómo este enfoque permite que la aplicación de gestión active notificaciones ante eventos específicos, desencadenando las acciones correspondientes en el sistema de aprovisionamiento. Luego, el sistema informa sobre los resultados de la operación, manteniendo una trazabilidad de las operaciones.

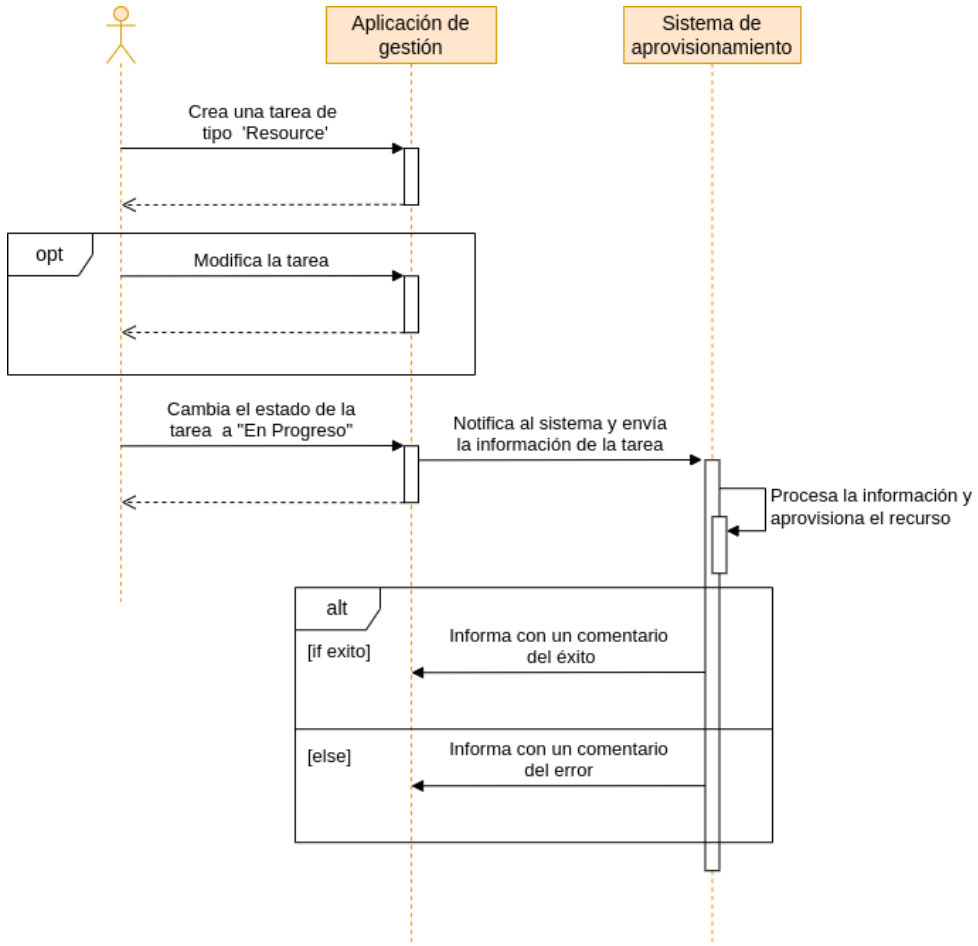


Figura 3.6: Diagrama de Secuencia: Creación y aprovisionamiento de un recurso a través de una aplicación de gestión

3.4. Diseño de algoritmos

3.4.1. Procesamiento de lenguaje natural

En la fase de análisis se identifica una necesidad relacionada con la accesibilidad de la herramienta. Con el propósito de reducir su complejidad y fomentar su uso por todo tipo de usuarios, se refleja el requisito funcional *RF-15* (ver Tabla 2.1), que establece la capacidad del sistema para **interpretar instrucciones en lenguaje natural**.

Para abordar el desafío de hacer la comunicación con la aplicación más sencilla, y accesible, se opta por incorporar inteligencia artificial a la solución. La función de esta IA es agregar una capa adicional al sistema, transformando instrucciones en lenguaje natural (sin estructura) en un formato estructurado (JSON), que la aplicación pudiera comprender y procesar de manera efectiva.

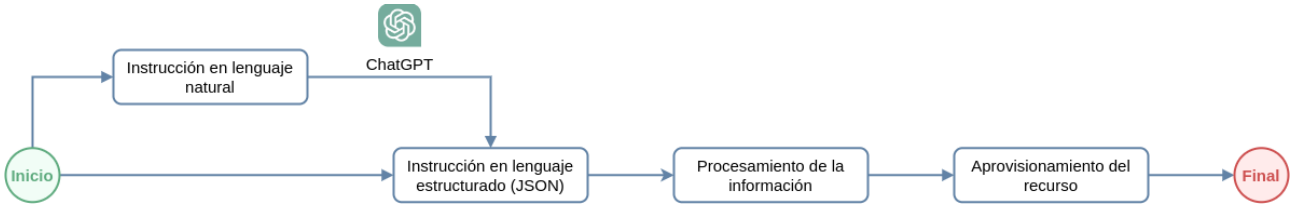


Figura 3.7: Proceso de transformación de instrucciones

Dado que el desarrollo completo de un sistema de inteligencia artificial está fuera del alcance de este proyecto, se decide utilizar un modelo existente: GPT-3.5-Turbo de OpenAI. Esta tecnología ha demostrado ser una opción sólida para tareas de inferencia, extracción de información y transformación de texto. Su propósito es lograr la conversión de instrucciones en lenguaje natural a JSON, permitiendo así que la aplicación comprenda y ejecute las solicitudes de los usuarios.

En resumen, la utilización de inteligencia artificial permite la transformación del lenguaje natural a un formato estructurado comprensible por la aplicación. Esto permite satisfacer el requisito funcional de interpretar instrucciones en lenguaje natural y crear recursos, facilitando la accesibilidad de la herramienta a todos los miembros del proyecto.

4 Desarrollo

En este capítulo, se detalla el proceso de desarrollo del sistema, abarcando alguno de sus aspectos más característicos, y su integración con una herramienta de gestión. En primer lugar, se explora cómo se implementa la transformación de instrucciones en lenguaje natural a JSON, mediante tecnologías de inteligencia artificial. Además, se presenta la integración con Jira, una de las herramientas de gestión más utilizadas, mostrando cómo se configura un Webhook. Después, se analizan las tecnologías empleadas para llevar a cabo con éxito la creación de la aplicación. Desde la elección del lenguaje de programación hasta la elección de la base de datos.

4.1. Aplicación

4.1.1. Procesamiento de lenguaje natural

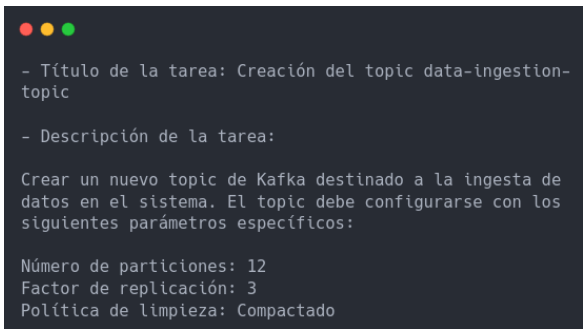
Como se indicó en la Sección 3.4.1, el sistema debe ser capaz de convertir instrucciones redactadas en lenguaje natural en datos con formato JSON. Para lograr esta transformación, se ha utilizado la tecnología GPT, reconocida por su habilidad para comprender y procesar textos.

El proceso de desarrollo ha consistido en la integración del sistema con la API de OpenAI y, en mayor medida, en la elaboración de una instrucción optimizada para la tarea en cuestión, cuyo resultado final se presenta en el Apéndice B. La finalidad de la instrucción es orientar al modelo de inteligencia artificial para capturar con precisión el propósito de la tarea, extraer la información necesaria y convertirla a un formato adecuado para su procesado por la aplicación. Esta interacción se realiza de manera sencilla: el usuario proporciona instrucciones en lenguaje natural, la aplicación las envía a procesar y el modelo, entrenado con una amplia variedad de datos, produce respuestas en el formato adecuado. Finalmente, la aplicación recibe estas respuestas y las utiliza para su propósito.

Para visualizar el proceso, se presenta un ejemplo en la Figura 4.1, que demuestra cómo una instrucción en lenguaje natural se transforma en un objeto JSON empleando inteligencia artificial.

En la figura, se observa como se introduce una instrucción en lenguaje natural para la creación de un topic de Kafka, especificando ciertos aspectos técnicos de configuración. A través de la interacción con el servicio de inteligencia artificial, el sistema interpreta la instrucción y genera el objeto JSON correspondiente.

Este ejemplo demuestra la capacidad del sistema para reconocer las intenciones del usuario, identificando la operación que desea llevar a cabo e inferir la información relacionada con la configuración del recurso que el usuario quiere crear. Además, ilustra su precisión en la conversión del texto en una estructura de datos específica.



(a) Especificación de la tarea

(b) Resultado del procesamiento

Figura 4.1: Procesamiento de lenguaje natural para la detección de tareas de aprovisionamiento

4.1.2. Integración con la herramienta de gestión

El proceso de desarrollo ha consistido en la configuración y creación de una serie de elementos en Jira para su integración con la aplicación desarrollada. La integración se lleva a cabo mediante la configuración de un Webhook, que, cuando una tarea de tipo específico se inicia, realiza una llamada a un endpoint expuesto por la API de la aplicación. A continuación, se presenta una descripción detallada del proceso, acompañada de una serie de imágenes ilustrativas.

Creación de un tipo de incidencia

En primer lugar, se diseñó un nuevo tipo de tarea en Jira, denominado 'Resource'. Este tipo de tarea se creó para distinguir las tareas relacionadas con el aprovisionamiento de recursos de otras tareas que se llevan a cabo durante el desarrollo.

Figura 4.2: Configuración del tipo de incidencia 'Resource' en Jira

Definición del flujo de trabajo

Se definió un flujo de trabajo específico para el tipo de tarea 'Resource' en Jira. El flujo de trabajo comprende una serie de estados que representan el progreso del proceso de aprovisionamiento de recursos. Estos estados incluyen 'Por hacer', 'En progreso', 'Error' y 'Hecho'

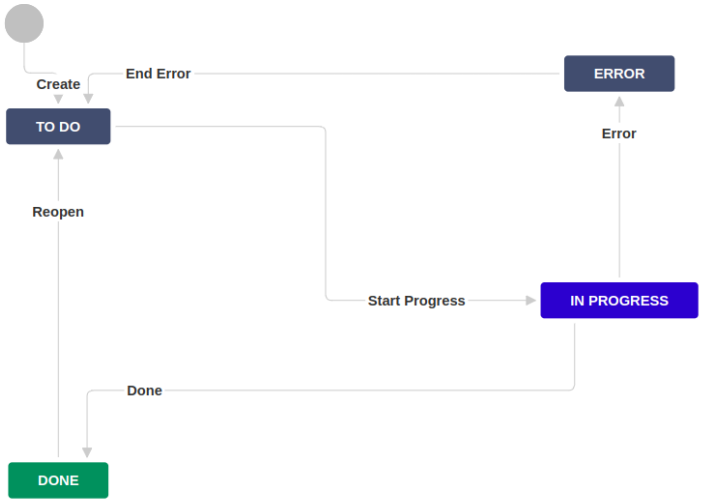


Figura 4.3: Flujo de trabajo para el tipo de incidencia 'Resource'

Configuración del Webhook

Para lograr la integración, se configuró un Webhook en Jira. Este Webhook está vinculado al tipo de tarea 'Resource' y se activa cuando la tarea cambia a un estado específico, en concreto, cuando se encuentra en el estado 'En progreso'. Para garantizar que el Webhook solo se activa en las situaciones deseadas, se añaden una serie de filtros. Además, se proporciona la URL del endpoint de la API de la aplicación, que debe ser invocado cuando se cumplen las condiciones del Webhook.

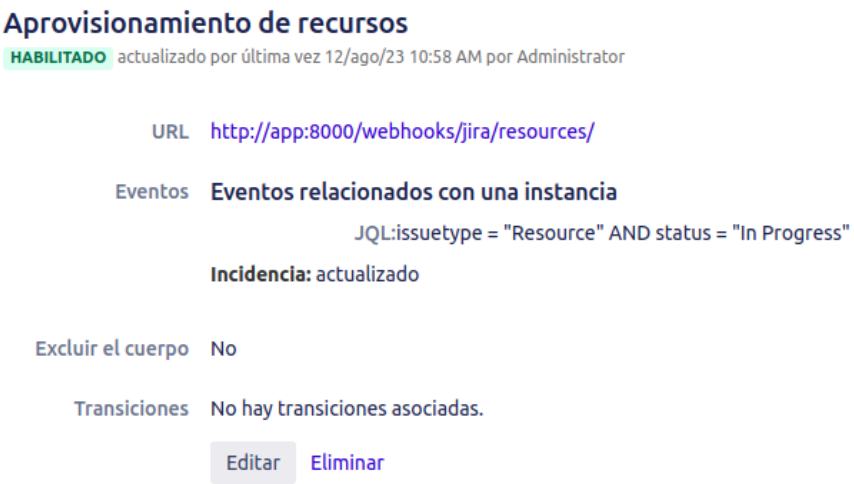


Figura 4.4: Configuración del Webhook de Jira para notificaciones en tiempo real

4.2. Tecnologías utilizadas

La industria del software es un sector en constante evolución, impulsado por una continua aparición de tecnologías que fomentan la creación de nuevos soluciones informáticas. Este rápido progreso genera un amplio abanico de tecnologías disponibles para utilizar en el desarrollo de software. El propósito de este apartado es detallar, entre la extensa gama de posibilidades, las tecnologías empleadas en el

desarrollo de la herramienta. Estas abarcan desde el lenguaje de programación y el framework web empleados para construir la aplicación, hasta aspectos relevantes como la persistencia de datos, la gestión de contenedores, el control de versiones o herramientas de visualización

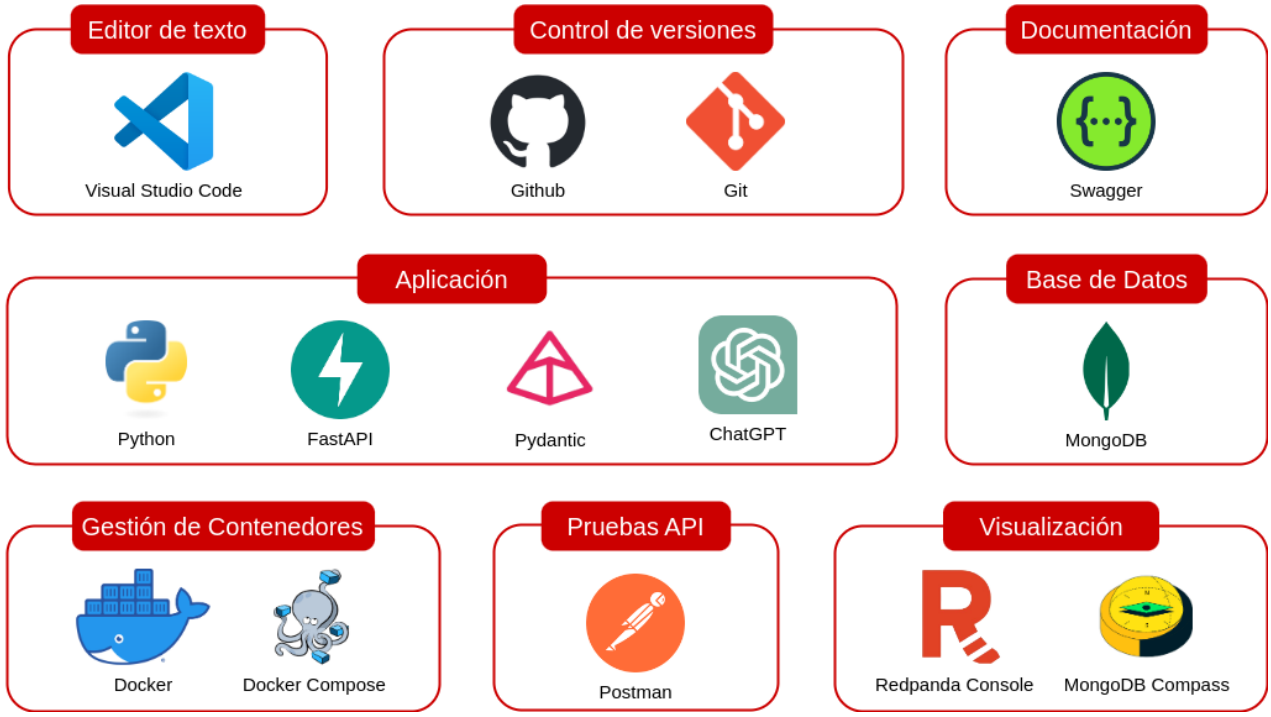


Figura 4.5: Diagrama de las tecnologías utilizadas en el desarrollo de la aplicación

4.2.1. Aplicación

Una decisión clave en el desarrollo de cualquier aplicación es seleccionar el lenguaje de programación adecuado. Para esta herramienta, se ha optado por Python debido a sus numerosas ventajas. Esta elección se basa principalmente en la facilidad de desarrollo y flexibilidad que proporciona el lenguaje, aspectos necesarios para la creación de un MVP.

Python es conocido por su sintaxis simple, que facilita y acelera el proceso de desarrollo. Además, la amplia comunidad con la que cuenta detrás ofrece y mantiene una gran cantidad de bibliotecas y módulos, que pueden aprovecharse para implementar soluciones de todo tipo.

En cuanto al framework para construir la API, se ha escogido FastAPI debido a sus características y facilidad de uso. FastAPI es un framework moderno que permite construir una API potente al mismo tiempo que se mantienen todas las ventajas del lenguaje.

Aunque existen otras opciones más completas en el desarrollo web con Python, como Django, ideales para el desarrollo de aplicaciones más complejas, en el caso de un MVP, resultan innecesarias, ya que se valoran otras características, como la agilidad para adaptar rápidamente el producto a nuevas necesidades o requisitos cambiantes.

Otro factor clave que ha ayudado a la elección de FastAPI es su uso de Pydantic para definir el modelo de datos, y asegurar que los datos enviados a la API y utilizados por la aplicación son válidos, reduciendo así el número de errores y mejorando la calidad general del código.

Además, FastAPI incorpora la generación automática de documentación basada en OpenAPI, lo que facilita el uso de la API tanto a usuarios que quieran hacer uso de ella, como para terceros que deseen integrarse.

En resumen, la elección de Python y FastAPI proporciona una combinación de agilidad y simplicidad en el desarrollo, ideal para sentar las bases de la herramienta y comenzar el desarrollo.

4.2.2. Base de datos

Para gestionar y almacenar los datos de manera eficiente, la aplicación se apoya en MongoDB como sistema de almacenamiento. Esta elección se fundamenta principalmente en su flexibilidad y capacidad de adaptación a datos heterogéneos. MongoDB, como base de datos no relacional, permite almacenar documentos JSON sin un esquema rígido y predefinido, lo que resulta especialmente relevante en el contexto de la herramienta, donde los datos no siempre tienen la misma forma o estructura.

Además, al permitir realizar cambios en el esquema de datos sin afectar a operaciones existentes, MongoDB facilita la implementación de nuevas funcionalidades y modificaciones futuras del MVP, sin requerir migraciones de datos. Esta ventaja resulta crucial para garantizar la adaptabilidad y mantenimiento eficiente de la aplicación a medida que evoluciona y se ajusta a las diferentes necesidades de los distintos proyectos.

4.2.3. Gestión de contenedores

La gestión de contenedores desempeña un papel fundamental en el despliegue de la aplicación y sus servicios. Docker se ha elegido como la herramienta principal para empaquetar la aplicación debido a las numerosas ventajas y facilidades que proporciona en el proceso de despliegue y la gestión de infraestructura.

Al encapsular la aplicación junto a sus dependencias en contenedores, Docker proporciona un entorno de ejecución aislado, asegurando que la aplicación se ejecuta de la misma manera, sin requerir configuraciones específicas, en cualquier entorno, ya sea local, de pruebas o en producción.

Además, se utiliza Docker Compose para desplegar los múltiples contenedores que forman el sistema, incluyendo la aplicación, la base de datos y los demás servicios. Docker Compose permite definir y gestionar toda infraestructura en un único archivo de configuración. Al ejecutar un solo comando, es capaz de iniciar y coordinar la creación de todos los contenedores necesarios.

En resumen, la elección de Docker para empaquetar la aplicación y Docker Compose para gestionar posteriormente el despliegue de los contenedores permite una gestión y un mantenimiento eficiente de la infraestructura, facilitando así el proceso de desarrollo y despliegue del sistema completo.

4.2.4. Visualización

Durante el desarrollo de la aplicación, se han empleado diversas herramientas para visualizar la infraestructura, como Redpanda Console y MongoDB Compass, para los topics y colecciones de MongoDB respectivamente. Además, se han aprovechado las herramientas de visualización nativas proporcionadas por Couchbase y Bitbucket para obtener información detallada sobre las colecciones y los repositorios de código, facilitando así el desarrollo y monitorización de la aplicación.

4.2.5. Otras

Para el desarrollo del código, se ha utilizado el editor de texto Visual Studio. Asimismo, para el versionado del código se ha utilizado Git, y Github para el almacenamiento del mismo. Además, se ha empleado Postman, una herramienta que ha permitido probar y validar manualmente las distintas funcionalidades de la aplicación, de forma rápida y precisa.

5 Pruebas

En el desarrollo de software, las pruebas automáticas desempeñan un papel fundamental al detectar errores y garantizar la calidad y fiabilidad del producto final. Estas pruebas permiten verificar de manera rápida el funcionamiento del software en diferentes escenarios, asegurando el cumplimiento de los requisitos establecidos.

5.1. Objetivo de las pruebas

El desarrollo de pruebas automáticas se ha convertido en una práctica ampliamente adoptada debido a sus numerosas ventajas. En primer lugar, las pruebas permiten identificar y corregir errores en el software antes de que afecten a los usuarios finales. Esto permite garantizar la calidad y fiabilidad del producto, evitando problemas que puedan afectar al resultado final. Las pruebas contribuyen a la detección de errores en el código y problemas de integración entre componentes, lo que contribuye a la estabilidad y la confianza del software.

En segundo lugar, las pruebas en el desarrollo de software son un elemento clave para asegurar el cumplimiento de los requisitos. A través de las pruebas, se verifica si el software funciona según lo acordado y si cumple con los criterios de aceptación establecidos.

Además, a medida que se realizan cambios en el código o se añaden nuevas funcionalidades, las pruebas automáticas aseguran que las modificaciones no introduzcan nuevos errores sobre las funcionalidades previamente desarrolladas. Esto brinda confianza en el producto y proporciona a los desarrolladores la seguridad de que todo continúa funcionando.

Por último, también mejoran la eficiencia y la productividad del equipo de desarrollo. Al detectar errores en etapas tempranas, se evita tener que rehacer trabajo y los costes asociados. Y adicionalmente, por el hecho de estar automatizadas, los desarrolladores pueden ejecutar conjuntos de pruebas completos, de manera más repetitiva, rápida y segura, lo que ahorra tiempo y recursos a largo plazo. En resumen, las pruebas en el desarrollo de software son esenciales para garantizar la calidad, cumplir con los requisitos del cliente y mejorar la eficiencia del proceso de desarrollo.

5.2. Estrategia de las pruebas

En el ámbito del software, existen numerosas técnicas de pruebas que permiten asegurarse del correcto funcionamiento del producto. Se distinguen fundamentalmente dos tipos de pruebas: las pruebas manuales y las pruebas automáticas. Las pruebas manuales se realizan en persona, interactuando con el software, haciendo clic y navegando por la aplicación. Por otro lado, las pruebas automáticas son realizadas por una máquina que ejecuta un conjunto de escenarios o casos de prueba previamente definidos.

En el contexto de este proyecto, para abordar la necesidad de crear un producto con bajo coste de mantenimiento, se ha optado por la implementación de pruebas automáticas. Las pruebas automáticas presentan una mayor consistencia y reducen las posibilidades de errores humanos, lo que las convierte en la opción preferida para asegurar la calidad del software en un entorno de desarrollo en constante evolución.

Dentro del espectro de pruebas automáticas, se han considerado diversos tipos, tales como pruebas unitarias, pruebas de integración y pruebas funcionales. Sin embargo, debido a las limitaciones de tiempo y a la naturaleza específica de la aplicación, se ha decidido enfocar los esfuerzos en las pruebas funcionales.

Las **pruebas funcionales se centran en validar el correcto funcionamiento de los requisitos del sistema**, incluyendo la interacción con todos los componentes involucrados en cada caso de uso. Estas pruebas aseguran que la aplicación cumpla con los requisitos establecidos y se comporte de acuerdo con las expectativas del usuario final.

Esta elección está fundamentada en la naturaleza de la aplicación, que se basa principalmente en la integración de diversos servicios externos, y no tanto en una lógica de negocio compleja. Las pruebas funcionales permiten abordar de manera exhaustiva todas las funcionalidades de la aplicación, garantizando la correcta integración de los servicios externos y la respuesta adecuada del software ante diversos escenarios.

Además, la automatización de las pruebas funcionales agiliza el proceso de verificación, permitiendo obtener resultados precisos en un tiempo reducido. De esta forma, se logra una mayor eficiencia en el proceso de pruebas y se asegura que el software cumpla con los estándares de calidad requeridos, sentando las bases para un desarrollo de bajo coste de mantenimiento.

En el Apéndice F se detallan los casos de prueba realizados.

5.3. Entorno de pruebas

Durante la etapa de pruebas de un software, es crucial disponer de un entorno de pruebas que permita asegurar la calidad del producto antes de su despliegue en producción. Este entorno debe tratarse de un ambiente aislado y controlado, que permita realizar experimentos y ejecutar pruebas sin correr riesgos ni afectar a sistemas críticos.

En el marco de este trabajo, se ha optado por el uso de Docker para la construcción del entorno de laboratorio (véase Apéndice E). Docker es una solución ligera y portable, especialmente adecuada para la creación de entornos de pruebas controlados, que encapsula la aplicación y sus dependencias en un contenedor. Además, permite implementar fácilmente redes internas para la comunicación entre diferentes componentes, simulando, a menor escala, la infraestructura de un entorno de producción.

Este tipo de entornos asegura una ejecución más fiable y precisa de las pruebas, facilitando la limpieza inmediata tras cada ejecución. Esta solución permite realizar cada prueba de manera independiente y aislada, evitando cualquier posible interferencia con residuos de ejecuciones previas. Además, simplifica el proceso de pruebas, ya que los desarrolladores no necesitan preocuparse del estado del entorno.

Otra ventaja significativa radica en la rapidez de despliegue que Docker proporciona. Los contenedores pueden ser creados y destruidos rápidamente, lo que acelera el ciclo de pruebas y facilita la adaptación del entorno de laboratorio a diferentes configuraciones.

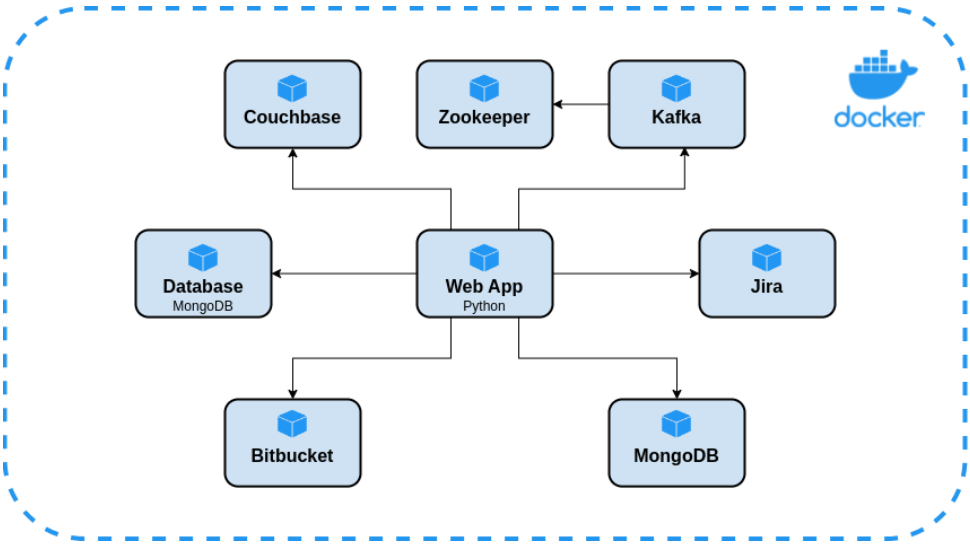


Figura 5.1: Diagrama del entorno de pruebas implementado para garantizar la funcionalidad de la aplicación

5.4. Resultado de las pruebas

Los resultados obtenidos en las pruebas han sido positivos y satisfactorios (véase Figura 5.2), demuestran la calidad del software realizado y contribuyen a la validación temprana de la herramienta. Este logro refleja el enfoque acertado en las pruebas automáticas, permitiendo obtener resultados rápidos y precisos, proporcionando una base sólida para su mejora continua en el futuro y garantizando un producto robusto y de bajo mantenimiento.

Pese a que se demuestra la capacidad de la herramienta para cumplir con los requisitos en un entorno de laboratorio con datos de muestra, se reconoce la necesidad de realizar pruebas adicionales en un entorno relevante y con datos reales para alcanzar el nivel de madurez superior.

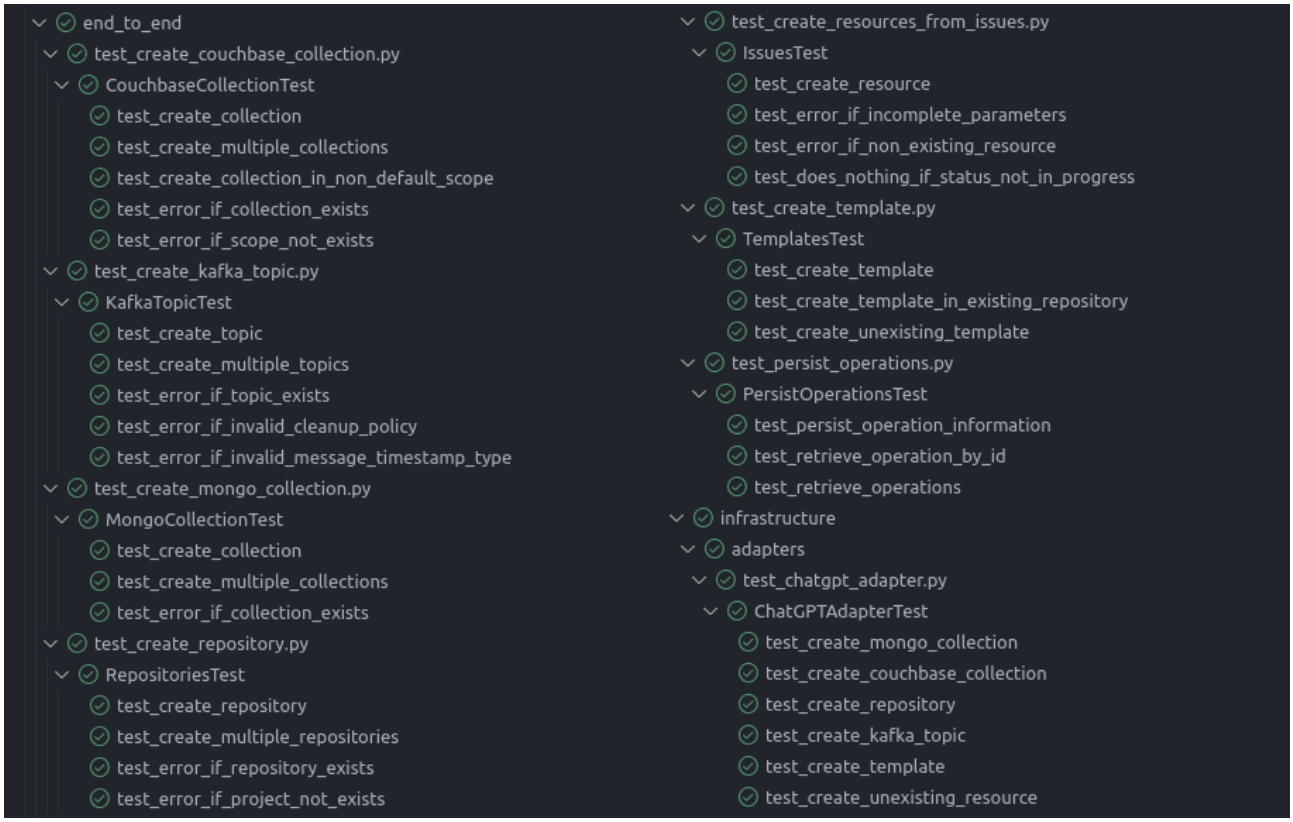


Figura 5.2: Resultado de ejecución de las pruebas. Todos los tests han sido superados, validando el correcto funcionamiento del software.

6 Conclusión

A lo largo de este documento, se han detallado las fases de creación de un Producto Mínimo Viable (MVP), explorando todas sus etapas de desarrollo, desde la idea inicial hasta la obtención de un software totalmente funcional.

El resultado, representado en este Trabajo de Fin de Grado, se ha materializado en un sistema destinado a integrarse como herramienta auxiliar en el ciclo de desarrollo de software, ofreciendo una funcionalidad básica en un área clave: el aprovisionamiento de recursos.

Esta herramienta demuestra su valía especialmente en la construcción de sistemas a gran escala, con un enfoque particular en aquellos basados en arquitecturas de microservicios. Dentro de este contexto es donde se espera que la herramienta alcance su máximo potencial, ya que estas arquitecturas requieren de la gestión una gran cantidad de recursos.

Los objetivos planteados al comienzo de este trabajo se han cumplido con éxito en su totalidad, desarrollando el producto acordado y probándolo en un entorno de laboratorio. A medida que se iba avanzando en el desarrollo, la aparición de algunos problemas ha impulsado la búsqueda de soluciones innovadoras. En particular, la incorporación de inteligencia artificial ha enriquecido la herramienta, dotándola de un valor adicional.

Este proyecto no solamente ha supuesto la construcción de un sistema software, sino que también ha sentado las bases para futuros desarrollos en el campo de la automatización. Los logros alcanzados ofrecen una contribución significativa en el ámbito del aprovisionamiento de recursos.

6.1. Trabajo a futuro

A pesar de haber logrado un Producto Mínimo Viable (MVP) funcional que cumple con el propósito principal de automatizar y agilizar el proceso de desarrollo mediante el aprovisionamiento de recursos y generación de código, existen diversos aspectos que pueden hacer evolucionar y mejorar la herramienta. A continuación, se presentan algunos de los elementos que tienen potencial para ser considerados en trabajos futuros:

1. **Ampliación de recursos**: El alcance del MVP se centra en los recursos más utilizados, como las bases de datos NoSQL o topics de Kafka. Para futuros desarrollos, sería conveniente expandir al abanico de recursos soportados. Esta expansión podría incluir otras bases de datos, sistemas de mensajería e incluso el aprovisionamiento de recursos en proveedores en la nube, como AWS, Google Cloud o Microsoft Azure.
2. **Interfaz Gráfica de Usuario (GUI)**: Conforme el producto evoluciona, sería beneficioso incorporar una interfaz gráfica de administración. Dicha interfaz podría mejorar significativamente el sistema en términos de monitorización y visualización, facilitando el control de todas las operaciones que se llevan a cabo.

3. **Inteligencia Artificial Avanzada**: Aunque se ha decidido no implementar un módulo de inteligencia artificial propio, sería interesante explorar si incorporar una inteligencia artificial específicamente entrenada para esta tarea podría optimizar aún más el proceso de aprovisionamiento. Se podría ir más allá, permitiendo que la IA no solamente transforme las descripciones de los recursos a crear en un formato adecuado para la aplicación. Sino que el usuario ni siquiera tuviera que indicar qué recursos deben crearse. En su lugar, la propia IA, basándose en patrones y conocimiento previo, sería capaz de identificar cuáles son los recursos que deben generarse para cada labor.

Acrónimos

API Application Programming Interface.

CLI Command Line Interface.

DTO Data Transfer Object.

GPT Generative Pretrained Transformer.

GUI Graphical User Interface.

IA Inteligencia Artificial.

JSON JavaScript Object Notation.

MVP Minimum Viable Product.

REST REpresentational State Transfer.

SRS Software Requirements Specifications.

Glosario

Caso de Prueba (1) Conjunto de entradas de prueba, condiciones de ejecución y resultados esperados desarrollados para un objetivo concreto, como ejercitar una determinada ruta de programa o verificar el cumplimiento de un requisito específico.
(2) Documentación que especifica las entradas, los resultados previstos y un conjunto de condiciones de ejecución para un elemento de prueba.

Entorno software Conjunto específico de configuraciones, recursos y condiciones bajo las cuales una aplicación o sistema opera.

Herramienta software Software utilizado en el desarrollo, comprobación, análisis o mantenimiento de un programa o su documentación.

Microservicios Los microservicios son un enfoque arquitectónico y organizativo para el desarrollo de software donde el software está compuesto por pequeños servicios independientes que se comunican a través de API bien definidas. Los propietarios de estos servicios son equipos pequeños independientes [2].

Producto Mínimo Viable Estrategia de desarrollo de software que consiste en crear un producto con un conjunto mínimo de características necesarias para satisfacer a los primeros usuarios y recopilar información valiosa para el desarrollo posterior.

Prompt Conjunto de palabras que desencadenan la generación de contenidos a través de un software de inteligencia artificial.

Pruebas funcionales (1) Pruebas que ignoran el mecanismo interno de un sistema o componente y se centran únicamente en los resultados generados en respuesta a entradas y condiciones de ejecución seleccionadas.
(2) Pruebas realizadas para evaluar la conformidad de un sistema o componente con los requisitos funcionales especificados.

Requisito funcional Requisito que especifica una función que un sistema o componente de un sistema debe ser capaz de realizar.

Webhook Función que permite a una aplicación o servicio recibir notificaciones automáticas y en tiempo real de eventos o cambios ocurridos en otro sistema externo. Es una forma de comunicación unidireccional en la que el sistema externo envía datos estructurados a una URL específica (endpoint) previamente configurada en la aplicación receptora.

Referencias

[1] S. Newman, Building Microservices. O'Reilly Media, 2021, ISBN: 9781492033998. dirección: https://books.google.es/books?id=aPM5EAAAQBAJ.

[2] AWS. «What are Microservices?» (), dirección: https://aws.amazon.com/microservices/ (visitado 28-08-2023).

[3] MongoDB. «¿Qué Es MongoDB?» (), dirección: https://www.mongodb.com/es/what-is-mongodb (visitado 28-08-2023).

[4] Couchbase. «Couchbase: The Modern Database for Enterprise Applications.» (), dirección: https://www.couchbase.com/ (visitado 28-08-2023).

[5] ConfluentInc. «What is Apache Kafka?» (), dirección: https://www.confluent.io/what-is-apache-kafka/ (visitado 28-08-2023).

[6] NASA. «Technology Readiness Levels - NASA Earth Science and Technology Office.» (19 de mar. de 2020), dirección: https://esto.nasa.gov/trl/ (visitado 11-07-2023).

[7] C. Richardso. «Microservices Pattern: Database per service.» (), dirección: https://microservices.io/patterns/data/database-per-service.html (visitado 26-08-2023).

[8] GitLab. «What is CI/CD?» (), dirección: https://about.gitlab.com/topics/ci-cd/ (visitado 27-08-2023).

[9] «IEEE Standard Glossary of Software Engineering Terminology,» IEEE Std 610.12-1990, págs. 1-84, dic. de 1990. DOI: 10.1109/IEEESTD.1990.101064.

[10] «IEEE Recommended Practice for Software Requirements Specifications,» IEEE Std 830-1998, págs. 1-40, 1998. DOI: 10.1109/IEEESTD.1998.88286.

[11] Macrometa. «What Are Collections in Databases?» (), dirección: https://www.macrometa.com/articles/what-are-collections-in-databases.

[12] MongoDB. «Databases and Collections; MongoDB Manual.» (), dirección: https://www.mongodb.com/docs/manual/core/databases-and-collections/ (visitado 24-08-2023).

[13] H. Graça. «DDD, Hexagonal, Onion, Clean, CQRS, ... How I put it all together.» (16 de nov. de 2017), dirección: https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/ (visitado 05-08-2023).

[14] S. Woltmann. «Hexagonal Architecture - What Is It? Why Should You Use It?» (18 de ene. de 2013), dirección: https://www.happycoders.eu/software-craftsmanship/hexagonal-architecture/ (visitado 26-08-2023).

[15] THECODEST. «The Power of Hexagonal Architecture.» (13 de jun. de 2023), dirección: https://thecodest.co/blog/t-strong-he-power-of-hexagonal-architecture/ (visitado 17-08-2023).

[16] I. Fulford y A. Ng. «ChatGPT Prompt Engineering for Developers.» (30 de abr. de 2023), dirección: https://www.deeplearning.ai/short-courses/chatgpt-prompt-engineering-for-developers/ (visitado 23-06-2023).

[17] OMG. «Unified Modeling Language, v2.5.1.» (12 de abr. de 2019), dirección: https://www.omg.org/spec/UML/2.5.1/PDF (visitado 16-08-2023).

Lista de Tablas

2.1. Requisitos funcionales del sistema 7
2.2. Requisitos no funcionales del sistema. 7

A.1. Horas dedicadas a la elaboración del trabajo 40

E.1. Imágenes de Docker empleadas para cada componente del sistema 45

F.1. Caso de prueba para la creación de una colección de MongoDB 46
F.2. Caso de prueba para la creación de múltiples colecciones de MongoDB 46
F.3. Caso de prueba para el manejo del error al intentar crear una colección de mongo que ya existe 47
F.4. Caso de prueba para la creación de una colección de Couchbase 47
F.5. Caso de prueba para la creación de múltiples colecciones de Couchbase 47
F.6. Caso de prueba para la creación de una colección de Couchbase en un scope personalizado 47
F.7. Caso de prueba para el manejo del error al intentar crear una colección de Couchbase que ya existe 48
F.8. Caso de prueba para el manejo del error al intentar crear una colección de Couchbase en un scope que no existe 48
F.9. Caso de prueba para la creación de un topic de Kafka 48
F.10. Caso de prueba para la creación de múltiples topics de Kafka 48
F.11. Caso de prueba para el manejo del error al intentar crear un topic de Kafka que ya existe 49
F.12. Caso de prueba para el manejo del error al intentar crear un topic de Kafka con una cleanup_policy inválida 49
F.13. Caso de prueba para el manejo del error al intentar crear un topic de Kafka con un message_timestamp_type inválido 49
F.14. Caso de prueba para la creación de un repositorio 49
F.15. Caso de prueba para la creación de múltiples repositorios 50
F.16. Caso de prueba para el manejo del error al intentar crear un repositorio que ya existe 50
F.17. Caso de prueba para el manejo del error al intentar crear un repositorio en un proyecto que no existe 50
F.18. Caso de prueba para la creación de una plantilla en un nuevo repositorio 51
F.19. Caso de prueba para la creación de una plantilla en un repositorio existente 51
F.20. Caso de prueba para el manejo del error al intentar crear una plantilla no disponible 51
F.21. Caso de prueba para la persistencia de los detalles cuando se realiza una operación . 51
F.22. Caso de prueba para la consulta de operaciones por identificador 52
F.23. Caso de prueba para consultar las últimas operaciones realizadas 52
F.24. Caso de prueba para la creación de un recurso a partir de una incidencia 52

F.25. Caso de prueba para el manejo del error al intentar crear un recurso a partir de una incidencia sin la información necesaria 52
F.26. Caso de prueba para el manejo del error al intentar crear un recurso desconocido a partir de una incidencia 53
F.27. Caso de prueba para comprobar que el aprovisionamiento no se inicia si la incidencia no está en progreso 53
F.28. Caso de prueba para la detección de un creado de colección de MongoDB 53
F.29. Caso de prueba para la detección de un creado de colección de Couchbase 54
F.30. Caso de prueba para la detección de un creado de repositorio 54
F.31. Caso de prueba para la detección de un creado de topic de Kafka 54
F.32. Caso de prueba para la generación de una plantilla 54
F.33. Caso de prueba para el manejo del error al detectar la creación de un recurso que no existe 55

Lista de Figuras

2.1. Diagrama de Casos de Uso de la aplicación 5

3.1. Bloques de la arquitectura de puertos y adaptadores 10
3.2. Adaptadores Primarios en la arquitectura de puertos y adaptadores 11
3.3. Adaptadores Secundarios en la arquitectura de puertos y adaptadores 12
3.4. Diagrama de la arquitectura basada en puertos y adaptadores 13
3.5. Modelo de datos: Interacción entre modelo interno, de la base de datos y de transferencia 16
3.6. Diagrama de Secuencia: Creación y aprovisionamiento de un recurso a través de una aplicación de gestión 18
3.7. Proceso de transformación de instrucciones 19

4.1. Procesamiento de lenguaje natural para la detección de tareas de aprovisionamiento 21
4.2. Configuración del tipo de incidencia 'Resource' en Jira 21
4.3. Flujo de trabajo para el tipo de incidencia 'Resource' 22
4.4. Configuración del Webhook de Jira para notificaciones en tiempo real 22
4.5. Diagrama de las tecnologías utilizadas en el desarrollo de la aplicación 23

5.1. Diagrama del entorno de pruebas implementado para garantizar la funcionalidad de la aplicación 28
5.2. Resultado de ejecución de las pruebas. Todos los tests han sido superados, validando el correcto funcionamiento del software. 29

A.1. Desglose de horas dedicadas a cada tarea: Distribución visual del esfuerzo 40

B.1. Instrucción utilizada para solicitar la clasificación de tareas al modelo de inteligencia artificial 41

C.1. Modelo de datos extendido: Modelo interno, de la base de datos y de transferencia . 43

D.1. Interfaz generada con Swagger UI para documentar los endpoints de la API 44

Apéndices

A Dedicación

A continuación se proporciona un desglose del tiempo dedicado a cada una de las diferentes etapas del proyecto. Estos datos ilustran una la dedicación cercana a la estimación planteada inicialmente para la realización del Trabajo Fin de Grado ($\approx$ 300 horas).

Tabla A.1: Horas dedicadas a la elaboración del trabajo

Tarea	**Tiempo dedicado**
Análisis	10.5
Diseño	22
Desarrollo	155
Pruebas	42
Memoria	111.5
Reuniones	5.5
Total	**346.5** horas

Figura A.1: Desglose de horas dedicadas a cada tarea: Distribución visual del esfuerzo

B Instrucción para la detección de tareas

En la siguiente imagen, se presenta la instrucción (*prompt*) diseñada para la detección de tareas de aprovisionamiento de recursos. Esta instrucción ha sido diseñada con la finalidad de guiar a un modelo de inteligencia artificial en la identificación y extracción de información vinculada al proceso de aprovisionamiento de recursos.

En este *prompt*, se puede observar cómo, en primer lugar, se provee un listado de todas las opciones disponibles para su clasificación. Posteriormente, se solicita inferir los parámetros asociados a cada una de estas alternativas. Finalmente, se indica el formato en el cual deben ser presentados los resultados obtenidos.

```
"""
Identifica que operación se solicita realizar a partir del título y descripción de una tarea. Responde con
una de las siguientes opciones:
    - Create MongoDB collection: (database_name, name)
    - Create Couchbase collection: (bucket_name, scope_name, name)
    - Create Kafka topic: (num_partitions, replication_factor, name, cleanup_policy, max_message_bytes,
min_cleanable_dirty_ratio, message_timestamp_type)
    - Create repository: (project_key, name)
    - Generate source code template: (project_key, repository_name, type[agr, mixbi, pcs])

El título y la descripción están delimitados por triple comillas. Formatea la respuesta como un objeto JSON
con "operation_type" como clave. Si la descripción no hace referencia a ninguna de las opciones
proporcionadas, usa "null" como valor para "operation_type".

Completa el objeto JSON con la clave "details" especificando los parámetros de la solicitud. Los parámetros
necesarios para cada operación se encuentran delimitados por paréntesis. Para alguno de estos parámetros,
se especifica sus posibles valores en una lista delimitada por corchetes. Para todos los parámetros no
especificados, utiliza "null" como valor.

Título: '''{}'''
Descripción: '''{}''
"""
```

Figura B.1: Instrucción utilizada para solicitar la clasificación de tareas al modelo de inteligencia artificial

C Modelo de datos extendido

Este apéndice presenta una versión más detallada del modelo de datos previamente introducido en la Sección 3.2. En este apartado, además de exponer los diversos modelos de datos que constituyen la aplicación, se proporciona una visión de los atributos que conforman cada objeto. La Figura C.1 muestra las tres representaciones distintas de estos modelos.

Los modelos de datos de transferencia desempeñan la función de recopilar la información procedente de la interfaz, ya sea esta la interfaz del sistema o la interfaz de usuario. Este proceso implica su posterior transformación al modelo interno de la aplicación, previa validación exhaustiva de cada uno de los parámetros. Esta validación se logra a través de comprobaciones numéricas y mediante el uso de enumeraciones para limitar ciertos valores.

Finalmente, se presenta el modelo de datos correspondiente a la base de datos. Dicho modelo consiste en un objeto que representa un documento dentro de una base de datos NoSQL, específicamente MongoDB, la cual es utilizada en este sistema.

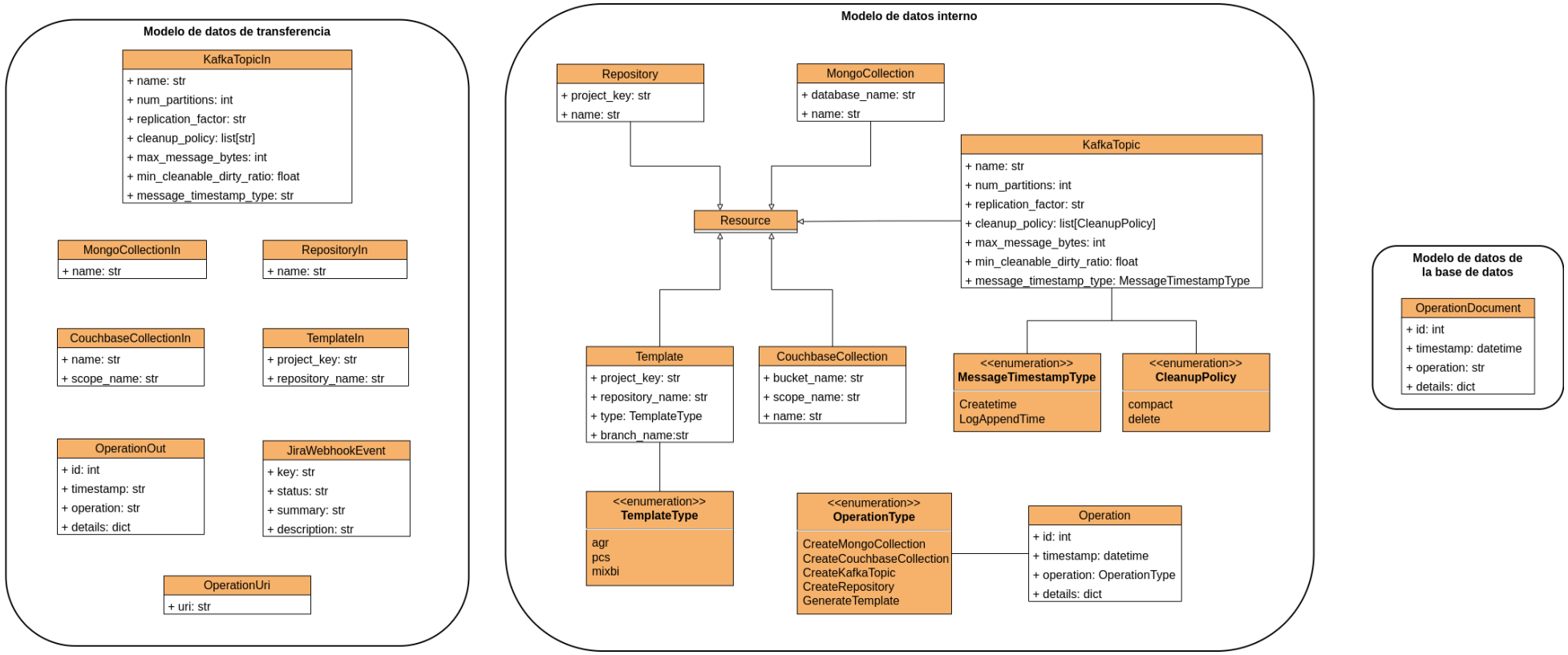


Figura C.1: Modelo de datos extendido: Modelo interno, de la base de datos y de transferencia

D Documentación de la API

En este apéndice, se expone la documentación generada en formato de página web correspondiente a la API de este proyecto. Para su desarrollo, se ha empleado la especificación OpenAPI generada de manera automática por el framework FastAPI, a partir del código fuente de la aplicación. Esta documentación permite visualizar de manera cómoda e intuitiva los diferentes endpoints y modelos de datos asociados a las tareas de aprovisionamiento de recursos. Además, ofrece la funcionalidad de efectuar pruebas mediante el envío interactivo de peticiones directamente a la aplicación.

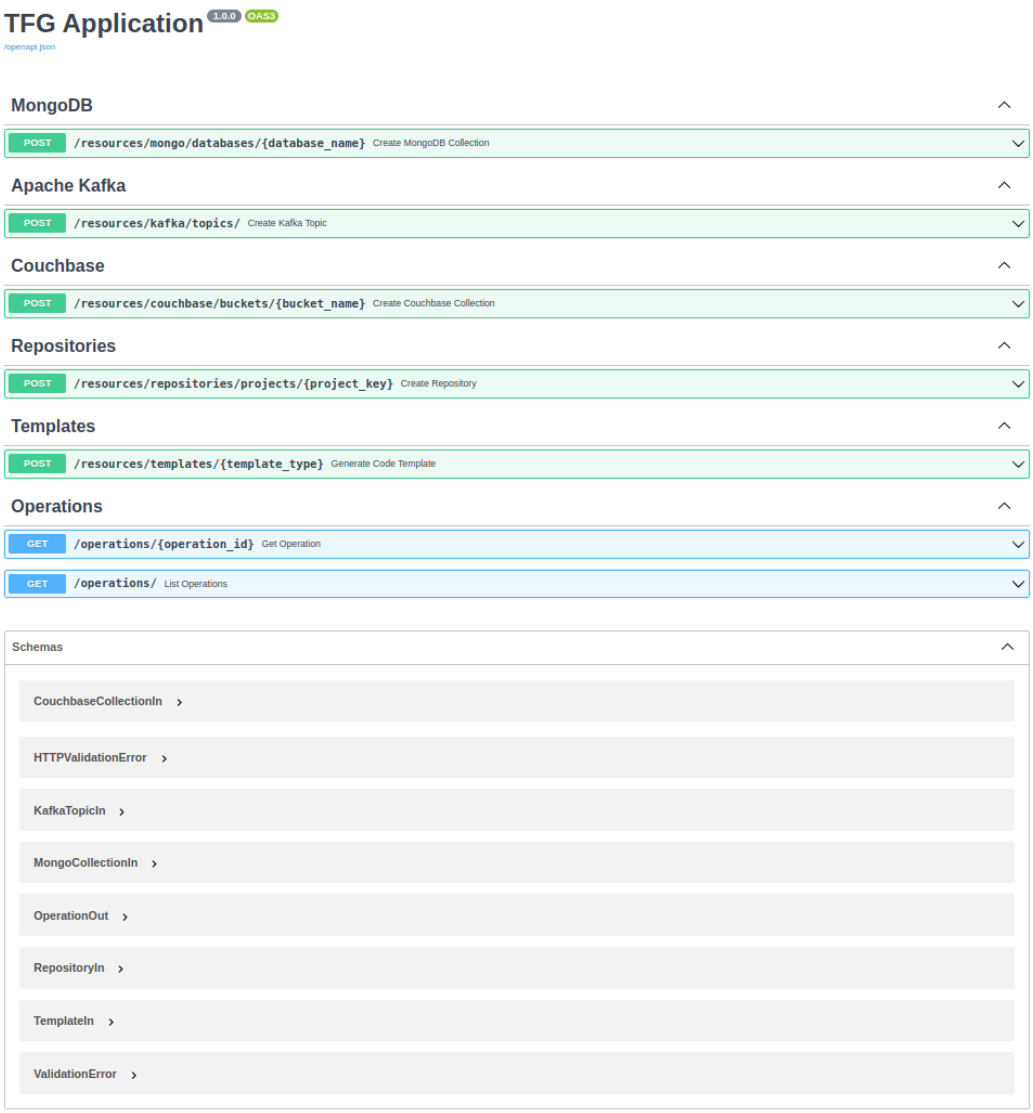


Figura D.1: Interfaz generada con Swagger UI para documentar los endpoints de la API

E Entorno de Pruebas

El entorno está formado por varios componentes, cada uno de los cuales se despliega en un contenedor Docker. Para facilitar su gestión se ha creado un archivo Docker Compose. A continuación, se muestran las imágenes de Docker utilizadas para cada uno de los componentes:

Tabla E.1: Imágenes de Docker empleadas para cada componente del sistema

Componente	**Descripción de la imagen**	**Imagen de Docker**
Web App	Construida a partir de una imagen de Python con el código de la aplicación	python:3.10.11[1]
Database	Imagen oficial de MongoDB	mongo:latest[2]
MongoDB	Imagen oficial de MongoDB	mongo:latest[3]
Couchbase	Imagen oficial de Couchbase	couchbase:latest[4]
Zookeeper	Imagen de Zookeeper proporcionada por Confluent Inc	confluentinc/cp-zookeeper:latest[5]
Kafka	Imagen de Kafka proporcionada por Confluent Inc	confluentinc/cp-kafka:latest[6]
Jira	Imagen de Jira Core de Atlassian	atlassian/jira-core:latest[7]
Bitbucket	Imagen de Bitbucket de Atlassian.	atlassian/bitbucket:latest[8]

[1] https://hub.docker.com/_/python
[2] https://hub.docker.com/_/mongo
[3] https://hub.docker.com/_/mongo
[4] https://hub.docker.com/_/couchbase
[5] https://hub.docker.com/r/confluentinc/cp-zookeeper
[6] https://hub.docker.com/r/confluentinc/cp-kafka
[7] https://hub.docker.com/r/atlassian/jira-core/
[8] https://hub.docker.com/r/atlassian/bitbucket

F Casos de Prueba

Con el fin de asegurar el correcto funcionamiento de la aplicación, se ha diseñado un conjunto de casos de prueba. Estos abarcan una amplia gama de situaciones, destinadas a evaluar cómo la aplicación responde al aprovisionar recursos en distintos estados de la infraestructura. Con este objetivo, se han considerado especialmente los siguientes escenarios:

- No hay recursos previos en la infraestructura. Se procede a validar la correcta creación de un recurso.
- Ya se han creado recursos previamente. Se procede a validar la creación de otro recurso adicional.
- Existe un recurso previamente creado. Se verifica el manejo de errores al intentar crear un recurso idéntico al ya existente.
- Se verifican los posibles errores que pueden surgir al intentar crear recursos con configuraciones inválidas.

A continuación, se detallan todos los escenarios creados para poner a prueba el sistema:

Tabla F.1: Caso de prueba para la creación de una colección de MongoDB

Caso de Prueba	1
Nombre	Crear una colección de MongoDB
Entrada	database_name: mydatabase
	data: {'name': 'mycollection'}
Pre-condición	La base de datos no contiene ninguna colección
Resultado Esperado	Respuesta con código de estado HTTP 201
	El cuerpo de la respuesta contiene la información de la colección creada
	La colección se ha creado en la base de datos

Tabla F.2: Caso de prueba para la creación de múltiples colecciones de MongoDB

Caso de Prueba	2
Nombre	Crear múltiples colecciones de MongoDB
Entrada	database_name: mydatabase
	data: {'name': 'new_collection'}
Pre-condición	Existe una colección previamente creada en la base de datos
Resultado Esperado	Respuesta con código de estado HTTP 201
	El cuerpo de la respuesta contiene la información de la colección creada
	Ambas colecciones están creadas en la base de datos

Tabla F.3: Caso de prueba para el manejo del error al intentar crear una colección de mongo que ya existe

Caso de Prueba	3
Nombre	Crear una colección de MongoDB repetida
Entrada	database_name: mydatabase
	data: {'name': 'mycollection'}
Pre-condición	Existe una colección previamente creada en la base de datos con el mismo nombre
Resultado Esperado	Respuesta con código de estado HTTP 409
	El cuerpo de la respuesta contiene un mensaje de error indicando que la colección ya existe
	No se ha creado una nueva colección

Tabla F.4: Caso de prueba para la creación de una colección de Couchbase

Caso de Prueba	4
Nombre	Crear una colección de Couchbase
Entrada	bucket_name: testbucket
	data: {'name': 'mycollection'}
Pre-condición	El bucket no contiene ninguna colección
Resultado Esperado	Respuesta con código de estado HTTP 201
	El cuerpo de la respuesta contiene la información de la colección creada
	La colección se ha creado en el bucket

Tabla F.5: Caso de prueba para la creación de múltiples colecciones de Couchbase

Caso de Prueba	5
Nombre	Crear múltiples colecciones de Couchbase
Entrada	bucket_name: testbucket
	data: {'name': 'previous_collection'}
Pre-condición	Existe una colección previamente creada en el bucket
Resultado Esperado	Respuesta con código de estado HTTP 201
	El cuerpo de la respuesta contiene la información de la colección creada
	Ambas colecciones están creadas en el bucket

Tabla F.6: Caso de prueba para la creación de una colección de Couchbase en un scope personalizado

Caso de Prueba	6
Nombre	Crear una colección de Couchbase en un scope personalizado
Entrada	bucket_name: testbucket
	data: {'name': 'previous_collection', 'scope_name': 'non_default'}
Pre-condición	Existe un scope distinto a '_default'
Resultado Esperado	Respuesta con código de estado HTTP 201
	El cuerpo de la respuesta contiene la información de la colección creada
	La colección se ha creado en el scope personalizado

Tabla F.7: Caso de prueba para el manejo del error al intentar crear una colección de Couchbase que ya existe

Caso de Prueba	7
Nombre	Crear una colección de Couchbase repetida
Entrada	bucket_name: testbucket
	data: {'name': 'mycollection'}
Pre-condición	Existe una colección previamente creada en el bucket con el mismo nombre
Resultado Esperado	Respuesta con código de estado HTTP 409
	El cuerpo de la respuesta contiene un mensaje de error indicando que la colección ya existe
	No se ha creado una nueva colección

Tabla F.8: Caso de prueba para el manejo del error al intentar crear una colección de Couchbase en un scope que no existe

Caso de Prueba	8
Nombre	Crear una colección de Couchbase en un scope que no existe
Entrada	bucket_name: testbucket
	data: {'name': 'previous_collection', 'scope_name': 'non_existing'}
Pre-condición	No existe el scope en el bucket
Resultado Esperado	Respuesta con código de estado HTTP 404
	El cuerpo de la respuesta contiene un mensaje de error indicando que el scope no existe
	No se ha creado el scope ni la colección

Tabla F.9: Caso de prueba para la creación de un topic de Kafka

Caso de Prueba	9
Nombre	Crear un topic de Kafka
Entrada	data: {topic_name: mytopic}
Pre-condición	El broker no tiene ningún topic
Resultado Esperado	Respuesta con código de estado HTTP 201
	El cuerpo de la respuesta contiene la información del topic creado
	El topic se ha creado en el broker

Tabla F.10: Caso de prueba para la creación de múltiples topics de Kafka

Caso de Prueba	10
Nombre	Crear múltiples topics de Kafka
Entrada	data: {topic_name: new_topic}
Pre-condición	Existe un topic previamente creado en el broker
Resultado Esperado	Respuesta con código de estado HTTP 201
	El cuerpo de la respuesta contiene la información del topic creado
	Ambos topics están creadas en el broker

Tabla F.11: Caso de prueba para el manejo del error al intentar crear un topic de Kafka que ya existe

Caso de Prueba	11
Nombre	Crear un topic de Kafka repetido
Entrada	data: {topic_name: existing_topic}
Pre-condición	Existe un topic previamente creado en el broker con el mismo nombre
Resultado Esperado	Respuesta con código de estado HTTP 409
	El cuerpo de la respuesta contiene un mensaje de error indicando que el topic ya existe
	No se ha creado un nuevo topic

Tabla F.12: Caso de prueba para el manejo del error al intentar crear un topic de Kafka con una cleanup_policy inválida

Caso de Prueba	12
Nombre	Crear un topic de Kafka con una cleanup_policy inválida
Entrada	data: {topic_name: invalid_topic, cleanup_policy: ['invalid']}
Pre-condición	El broker no tiene ningún topic
Resultado Esperado	Respuesta con código de estado HTTP 400
	El cuerpo de la respuesta contiene un mensaje de error indicando que la cleanup_policy es inválida
	No se ha creado el topic

Tabla F.13: Caso de prueba para el manejo del error al intentar crear un topic de Kafka con un message_timestamp_type inválido

Caso de Prueba	13
Nombre	Crear un topic de Kafka con un message_timestamp_type inválido
Entrada	data: {topic_name: invalid_topic, message_timestamp_type: 'invalid'}
Pre-condición	El broker no tiene ningún topic
Resultado Esperado	Respuesta con código de estado HTTP 400
	El cuerpo de la respuesta contiene un mensaje de error indicando que el message_timestamp_type el inválido
	No se ha creado el topic

Tabla F.14: Caso de prueba para la creación de un repositorio

Caso de Prueba	14
Nombre	Crear un repositorio
Entrada	project_key: testproj
	data: {'name': 'myrepository'}
Pre-condición	El proyecto no tiene ningún repositorio
Resultado Esperado	Respuesta con código de estado HTTP 201
	El cuerpo de la respuesta contiene la información del repositorio creado
	El repositorio se ha creado en el proyecto

Tabla F.15: Caso de prueba para la creación de múltiples repositorios

Caso de Prueba	15
Nombre	Crear múltiples repositorios
Entrada	project_key: testproj
	data: {'name': 'new_repository'}
Pre-condición	Existe un repositorio previamente creado en el proyecto
Resultado Esperado	Respuesta con código de estado HTTP 201
	El cuerpo de la respuesta contiene la información del nuevo repositorio creado
	Ambos repositorios están creados en el proyecto

Tabla F.16: Caso de prueba para el manejo del error al intentar crear un repositorio que ya existe

Caso de Prueba	16
Nombre	Crear un repositorio repetido
Entrada	project_key: testproj
	data: {'name': 'previous_repository'}
Pre-condición	Existe un repositorio previamente creado en el proyecto con el mismo nombre
Resultado Esperado	Respuesta con código de estado HTTP 409
	El cuerpo de la respuesta contiene un mensaje de error indicando que el repositorio ya existe
	No se ha creado un nuevo repositorio

Tabla F.17: Caso de prueba para el manejo del error al intentar crear un repositorio en un proyecto que no existe

Caso de Prueba	17
Nombre	Crear un repositorio en un proyecto que no existe
Entrada	project_key: non_existing
	data: {'name': 'my_repository'}
Pre-condición	No existe el proyecto
Resultado Esperado	Respuesta con código de estado HTTP 404
	El cuerpo de la respuesta contiene un mensaje de error indicando que el proyecto no existe
	No se ha creado el proyecto ni el repositorio

Tabla F.18: Caso de prueba para la creación de una plantilla en un nuevo repositorio

Caso de Prueba	18
Nombre	Crear una plantilla en un nuevo repositorio
Entrada	template_type: agr
	data: {'project_key': 'testproj', 'name': 'myrepository'}
Pre-condición	No existe ningún repositorio
Resultado Esperado	Respuesta con código de estado HTTP 201
	El cuerpo de la respuesta contiene la información de la plantilla creada
	El repositorio se ha creado en el proyecto
	La plantilla se ha subido a una rama del repositorio

Tabla F.19: Caso de prueba para la creación de una plantilla en un repositorio existente

Caso de Prueba	19
Nombre	Crear una plantilla en un repositorio existente
Entrada	template_type: agr
	data: {'project_key': 'testproj', 'name': 'previous_repository'}
Pre-condición	No existe ningún repositorio
Resultado Esperado	Respuesta con código de estado HTTP 201
	El cuerpo de la respuesta contiene la información de la plantilla creada
	La plantilla se ha subido a una rama del repositorio

Tabla F.20: Caso de prueba para el manejo del error al intentar crear una plantilla no disponible

Caso de Prueba	20
Nombre	Crear una platilla no disponible
Entrada	template_type: 'no_existing'
	data: {'project_key': 'testproj', 'name': 'myrepository'}
Pre-condición	No existe el proyecto
Resultado Esperado	Respuesta con código de estado HTTP 400
	El cuerpo de la respuesta contiene un mensaje de error indicando que la plantilla no existe
	La plantilla no se ha subido a una rama del repositorio

Tabla F.21: Caso de prueba para la persistencia de los detalles cuando se realiza una operación

Caso de Prueba	21
Nombre	Persistir la información cuando se realiza una operación
Entrada	topic_name: 'random'
Pre-condición	Una base de datos sin ninguna operación
Resultado Esperado	La operación queda persistida en base de datos
	El campo 'details' del documento contiene la información del recurso creado

Tabla F.22: Caso de prueba para la consulta de operaciones por identificador

Caso de Prueba	22
Nombre	Consultar la información de una operación a partir de su identificador
Entrada	id: 1
Pre-condición	Una base de datos con una operación de identificador 1
Resultado Esperado	La operación es devuelta
	El campo 'details' del documento contiene la información del recurso creado en la operación 1

Tabla F.23: Caso de prueba para consultar las últimas operaciones realizadas

Caso de Prueba	23
Nombre	Consultar la información de las últimas operaciones realizadas
Entrada	id: 1
Pre-condición	Una base de datos con dos operaciones
Resultado Esperado	Devuelve dos operaciones
	El campo 'details' de ambas operaciones contiene la información de su recurso correspondiente

Tabla F.24: Caso de prueba para la creación de un recurso a partir de una incidencia

Caso de Prueba	24
Nombre	Crea un recurso mediante lenguaje natural
Entrada	Título: 'Crear colección de mongo'
	Descripción: 'Se solicita la creación de la colección testcollection en mydatabase'
	Estado: 'En progreso'
Pre-condición	Un sistema sin recursos previamente creados
Resultado Esperado	Respuesta con código de estado HTTP 204
	Se ha escrito un comentario en la incidencia informando de que la operación ha sido realizada con éxito
	La colección se ha creado correctamente

Tabla F.25: Caso de prueba para el manejo del error al intentar crear un recurso a partir de una incidencia sin la información necesaria

Caso de Prueba	25
Nombre	Crear un recurso mediante lenguaje natural sin la información necesaria
Entrada	Título: 'Crear colección de mongo'
	Descripción: 'Se solicita la creación de la colección testcollection'
	Estado: 'En progreso'
Pre-condición	Un sistema sin recursos previamente creados
Resultado Esperado	Respuesta con código de estado HTTP 400
	El cuerpo de la respuesta contiene un mensaje de error indicando que la información que falta
	Se ha escrito un comentario en la incidencia informando de que la operación no ha sido realizada

Tabla F.26: Caso de prueba para el manejo del error al intentar crear un recurso desconocido a partir de una incidencia

Caso de Prueba	26
Nombre	Crear un recurso desconocido mediante lenguaje natural
Entrada	Título: 'Crear una tabla en postgresql'
	Descripción: 'Se solicita la creación de la tabla testtable en la base de datos mydatabase'
	Estado: 'En progreso'
Pre-condición	Un sistema sin recursos previamente creados
Resultado Esperado	Respuesta con código de estado HTTP 404
	El cuerpo de la respuesta contiene un mensaje de error indicando que el recurso solicitado no existe
	Se ha escrito un comentario en la incidencia informando que la operación no ha sido realizada

Tabla F.27: Caso de prueba para comprobar que el aprovisionamiento no se inicia si la incidencia no está en progreso

Caso de Prueba	27
Nombre	Crear un recurso con una incidencia que no está en progreso
Entrada	Título: 'Crear colección de mongo'
	Descripción: 'Se solicita la creación de la colección testcollection en mydatabase'
	Estado: 'To Do'
Pre-condición	Un sistema sin recursos previamente creados
Resultado Esperado	Respuesta con código de estado HTTP 204
	No se ha realizado ninguna operación de aprovisionamiento

Tabla F.28: Caso de prueba para la detección de un creado de colección de MongoDB

Caso de Prueba	29
Nombre	Infiere de una incidencia en lenguaje natural la creación de una colección de MongoDB
Entrada	Título: 'Crear una colección de mongo'
	Descripción: 'Crea la colección testcollection en mytestdatabase'
Pre-condición	
Resultado Esperado	Infiere la creación de una colección de MongoDB
	Detecta todos los parámetros proporcionados

Tabla F.29: Caso de prueba para la detección de un creado de colección de Couchbase

Caso de Prueba	29
Nombre	Infiere de una incidencia en lenguaje natural la creación de una colección de Couchbase
Entrada	Título: 'Crear una colección de couchbase'
	Descripción: 'Crea la colección testcollection en mytestbucket'
Pre-condición	
Resultado Esperado	Infiere la creación de una colección de Couchbase
	Detecta todos los parámetros proporcionados

Tabla F.30: Caso de prueba para la detección de un creado de repositorio

Caso de Prueba	30
Nombre	Infiere de una incidencia en lenguaje natural la creación de un repositorio
Entrada	Título: 'Crear repositorio'
	Descripción: 'Crea el repositorio testrepository en testproj'
Pre-condición	
Resultado Esperado	Infiere la creación de un repositorio
	Detecta todos los parámetros proporcionados

Tabla F.31: Caso de prueba para la detección de un creado de topic de Kafka

Caso de Prueba	31
Nombre	Infiere de una incidencia en lenguaje natural la creación de un topic de Kafka
Entrada	Título: 'Crear topic'
	Descripción: 'Crea el topic testtopic de 12 particiones y politica delete'
Pre-condición	
Resultado Esperado	Infiere la creación de un topic de Kafka
	Detecta todos los parámetros proporcionados

Tabla F.32: Caso de prueba para la generación de una plantilla

Caso de Prueba	32
Nombre	Infiere de una incidencia en lenguaje natural la generación de una plantilla
Entrada	Título: 'Crear microservicio testservice-agr'
	Descripción: 'Genera la plantilla en testproj'
Resultado Esperado	Infiere la generación de una plantilla
	Detecta todos los parámetros proporcionados

Tabla F.33: Caso de prueba para el manejo del error al detectar la creación de un recurso que no existe

Caso de Prueba	33
Nombre	Infiere de una incidencia en lenguaje natural que el recurso solicitado no existe
Entrada	Título: 'Crear tabla de postgreSQL'
	Descripción: 'Crea la tabla testtable en mydatabase'
Pre-condición	
Resultado Esperado	Detecta que el recurso solicitado no existe