Universidad Siglo 21

Trabajo Final de Grado. Prototipado Tecnológico

Carrera: Licenciatura en Informática

Plataforma de Generación de Código y Arquitecturas de Microservicios

Autor: Tomás Darquier

Legajo: VINF10408

Tutor: Pablo Alejandro Virgolini

S.C de Bariloche, noviembre de 2024

Indice

Índice de Figuras

Índice de Tablas

Resumen

Con la creciente adopción del paradigma de microservicios, se han obtenido múltiples beneficios en el desarrollo de software. Aun así, esto no quita que la mencionada metodología también presente ciertos inconvenientes. Se conoció, mediante múltiples técnicas de recolección de datos, que uno de ellos es la repetición innecesaria en el desarrollo de componentes comunes en sistemas genéricos. Los desarrolladores se ven obligados a recrear estos componentes en múltiples ocasiones en sus diferentes sistemas y, además, deben configurar manualmente las comunicaciones entre ellos, lo que consume tiempo y aumenta la complejidad del desarrollo. En respuesta a esta problemática, se desarrolló una plataforma web que, mediante la generación dinámica de código basada en plantillas, facilita la creación y configuración de arquitecturas de microservicios mediante una interfaz gráfica intuitiva. A través de un proceso visual, se les permite a los usuarios seleccionar y conectar microservicios genéricos, estructurando sus arquitecturas de manera personalizada y adecuada a sus requerimientos. La interacción es sencilla: los desarrolladores arrastran y sueltan los elementos sobre un lienzo y establecen visualmente las conexiones entre ellos. Al finalizar, obtienen el código generado, reduciendo a unos cuantos clics el desarrollo de un sistema distribuido completamente funcional.

Palabras clave: microservicios, plataforma web, generación de código, arquitectura distribuida.

Abstract

With the growing adoption of the microservices paradigm, numerous benefits have been achieved in software development. Nonetheless, this methodology also has certain drawbacks. Through various data collection techniques, one identified issue is the unnecessary repetition in the development of common components for generic systems. Developers are often required to recreate these components multiple times across different systems and must manually configure the communications between them, which is time-consuming and increases development complexity. To address this issue, a web platform was developed that, through template-based dynamic code generation, facilitates the creation and configuration of microservices architectures via an intuitive graphical interface. Through a visual process, users can select and connect generic microservices, structuring their architectures in a personalized way that suits their requirements. The interaction is straightforward: developers drag and drop elements onto a canvas and visually establish the connections between them. Upon completion, they obtain the generated code, reducing the development of a fully functional distributed system to just a few clicks.

Keywords: microservices, web platform, code generation, distributed architecture.

Titulo

Plataforma de Generación de Código y Arquitecturas de Microservicios

Introducción

El presente proyecto consistió en una plataforma web orientada a desarrolladores de software que, mediante una interfaz gráfica, permite crear microservicios personalizados con funcionalidades genéricas y, a través de la configuración de su interrelación, obtener arquitecturas funcionales mediante mínima inversión de tiempo.

Antecedentes

La adopción de arquitecturas de microservicios para el desarrollo de software en la nube por parte de grandes compañías como eBay, Amazon y Netflix impulsó una tendencia masiva en la industria.

Este cambio trajo consigo múltiples beneficios en términos de escalabilidad, mantenimiento y encapsulación de código. Sin embargo, también resultó en un notable incremento en la complejidad durante el desarrollo en comparación con una solución monolítica, lo que implicó un mayor requerimiento de capital humano, especialmente en las fases de codificación, pruebas e integración de los módulos (Elgheriani y Ahmed, 2022; Lopez y Garcia 2019).

A su vez, el movimiento Low-code, el cual protagonizó su origen en la última década, permitió a muchas organizaciones beneficiarse de su utilidad, como Gartner, quienes comentan que para la finalización del presente año más de un 65% de los desarrollos provendrán de este medio (Vincent et al., 2019).

Debido a que tanto el origen de la arquitectura de microservicios, como el movimiento Low-code son relativamente cercanos, no abundan combinaciones entre estos tópicos, sin embargo, la aparición reciente de múltiples artículos académicos alrededor de este tema, indica un interés creciente en explorar cómo la integración de

enfoques Low-code con arquitecturas de microservicios puede optimizar el desarrollo de software.

Un claro ejemplo de lo mencionado, y con cierta relación con el presente proyecto, se trata de la idea de un lenguaje específico para microservicios, como un paso hacia la integración de plataformas Low-code, propuesto por Said, Ezzati y Arezki (2023).

Descripción del Área Problemática

Los desarrolladores de software requieren, de manera cotidiana, lidiar con inconvenientes relacionados a la compleja integración de microservicios, que no solo es propensa a errores, sino requiere una meticulosa gestión de sus interfaces y contratos (Misic et al., 2017). Además, dentro de un entorno de microservicios, la administración de configuraciones agrega una capa de complejidad, debido a que cada servicio puede tener especificas necesidades de configuración, que deben manejarse de manera consistente en varios entornos (Fowler y Lewis, 2014).

A este problema se suma, para los programadores de pequeños productos SaaS, quienes como las siglas indican persiguen el modelo de software como servicio, la necesidad frecuente de diseñar y desarrollar componentes similares debido a los puntos en común entre las necesidades de los clientes, a pesar de que el producto final no guarde relación directa con esos mismos requerimientos. De hecho, ciertos microservicios, como son los referidos a la autenticación, registro de actividades y configuración, suelen ser omnipresentes en distintas arquitecturas debido a su rol esencial en la seguridad, observabilidad y gestión de sistemas distribuidos (Richardson, 2019).

Justificación

La implementación del presente proyecto permitió, a quienes poseen al menos un conocimiento básico en el área de arquitecturas distribuidas, convertir procesos referidos a la creación de microservicios genéricos, que previamente podrían llevar horas o incluso días, a un breve proceso de configuración, mediante una interfaz clara y flexible, tomando no más de unos minutos. De esta forma, el sistema brindó un ahorro temporal

considerable a sus usuarios, lo que, a su vez, se tradujo en una mejora en la productividad. Este aumento en la eficiencia generó una serie de beneficios en el ámbito del desarrollo de software, como la reducción de costos operativos, la aceleración de los tiempos de entrega y una mayor capacidad para gestionar proyectos de manera más efectiva, resultando en una optimización global del proceso de desarrollo. Brown y Smith (2023) destacan los beneficios de este tipo de procesos:

> Las plataformas Low-code y las interfaces gráficas intuitivas no solo facilitan la creación rápida de aplicaciones, sino que también reducen significativamente la complejidad y los costos asociados al desarrollo, permitiendo a los desarrolladores con menos experiencia crear arquitecturas sofisticadas con mayor eficiencia. (p. 78).

La capacidad de configurar microservicios a través de una interfaz gráfica intuitiva representó una mejora significativa en la forma de construir y gestionar las arquitecturas de software, no solo agilizando procesos de desarrollo y despliegue, sino también impulsando la adopción de tecnologías de información y comunicación al reducir la barrera de entrada para la implementación de soluciones distribuidas. Por lo tanto, esta plataforma democratizó el acceso a tecnologías avanzadas al permitir que desarrolladores con conocimientos básicos puedan crear arquitecturas complejas, promoviendo así la inclusión y la capacitación en tecnologías emergentes.

Objetivo General del Proyecto

Desarrollar e implementar un sistema Low-code que facilite a los desarrolladores diseñar, configurar e integrar arquitecturas de microservicios de forma eficiente mediante interfaces visuales y componentes reutilizables, optimizando el tiempo y reduciendo la complejidad en comparación con los enfoques tradicionales, asegurando la calidad mediante validaciones previas.

Objetivos Específicos del Proyecto

Diseñar una interfaz que permita a los desarrolladores crear y configurar arquitecturas de microservicios de manera visual y sin necesidad de codificación extensa.

Desarrollar y ofrecer un conjunto de componentes reutilizables y plantillas predefinidas para facilitar el diseño y la integración de microservicios, reduciendo el tiempo de desarrollo y estandarizando los procesos.

Implementar herramientas de validación y pruebas automatizadas dentro del sistema Low-code para garantizar la calidad y funcionalidad de las arquitecturas de microservicios, minimizando errores.

Establecer un sistema de documentación que proporcione guías y asistencia a los usuarios, con el fin de facilitar la adopción del sistema, optimizando el uso de la plataforma.

Marco Teórico Referencial

Dominio del Problema

Inicialmente, se debe definir a que nos referimos al hablar de una arquitectura distribuida de microservicios. Como señala Newman (2015), "las arquitecturas distribuidas de microservicios dividen una aplicación en servicios pequeños, autónomos y desplegables de forma independiente, lo que permite a las organizaciones escalar componentes específicos según sea necesario y mejorar la resiliencia general del sistema" (p.5).

Este tipo de arquitecturas permite la interoperabilidad entre microservicios desarrollados en diferentes lenguajes de programación, debido a sus estándares de comunicación como lo son REST o gRPC y sus contratos de servicio.

La interoperabilidad en arquitecturas de microservicios es crucial, dado que los servicios pueden estar construidos en diferentes lenguajes de programación o plataformas. Mantener estándares comunes en las interfaces y contratos de servicio es

fundamental para garantizar que estos servicios puedan comunicarse de manera efectiva, aunque esto también introduce complejidades adicionales en el proceso de desarrollo y mantenimiento (Flower, 2016).

Las complejidades mencionadas, inherentes en las arquitecturas de microservicios, pueden dilatar los tiempos de desarrollo, pruebas e integración, afectando las capacidades productivas de una empresa o desarrollador.

> Estos sistemas normalmente están diseñados para ser escalables y resilientes, pero eso no viene sin costo. Puede añadir complejidad al proceso de desarrollo, ya que es otro sistema que podrías necesitar ejecutar para desarrollar y probar tus servicios. También pueden ser necesarias máquinas adicionales y experiencia para mantener esta infraestructura en funcionamiento. (Newman, 2015, p.56).

Debido a lo expuesto en la cita, muchas empresas decidieron brindar al mercado herramientas para intentar mitigar los conflictos expuestos. Es por esto por lo que, herramientas web como Power Apps de Microsoft o BettyBlocks, tuvieron su génesis brindando a los desarrolladores abstracciones de la excesiva complejidad presentada por las arquitecturas distribuidas, permitiendo el desarrollo de aplicaciones de esta índole de una forma más sencilla mediante iniciativas Low-code.

A su vez, la solución propuesta referida a la simplificación del desarrollo de microservicios mediante herramientas Low-code, fue, y sigue siendo investigado por diversos autores del ámbito académico, evidenciando la presencia de la problemática tratada. Como Chaudhary y Margaria (2021) mediante su solución ideada basada en DSL y drag and drop, o también, la propuesta de Dhoke y Lokulwar (2023), soportada en una interfaz similar a la presente en la plataforma Scratch, con el objeto de permitir la programación de servicios backend mediante bloques visuales preconstruidos y plantillas.

TICs

A continuación, se listan las tecnologías utilizadas en el desarrollo del proyecto:

Lenguajes de Programación

Java: "Java es un lenguaje multiplataforma, orientado a objetos y centrado en la red que se puede utilizar como una plataforma en sí mismo" (Amazon Web Services, 2023).

JavaScript: "JavaScript es un lenguaje de programación o de secuencias de comandos que te permite implementar funciones complejas en páginas web" (Mozilla Developer Network, 2024).

Frameworks y Bibliotecas

Spring: "Spring Framework proporciona un modelo integral de programación y configuración para aplicaciones empresariales modernas basadas en Java, en cualquier tipo de plataforma de implementación" (Spring, 2024).

Apache Velocity: "Velocity es un motor de plantillas basado en Java. Permite a cualquiera utilizar un lenguaje de plantilla simple pero potente para hacer referencia a objetos definidos en código Java." (Apache Software Fundation, 2020).

Apache Jena: "Jena es un marco Java para crear aplicaciones de Web Semántica. Proporciona amplias bibliotecas de Java para ayudar a los desarrolladores a desarrollar código que maneje RDF, RDFS, RDFa, OWL y SPARQL." (Apache Software Fundation, 2024).

Thymeleaf: "El objetivo principal de Thymeleaf es aportar plantillas naturales y elegantes a tu flujo de trabajo de desarrollo HTML, que puede ser correctamente mostrado en los navegadores y también funcionar como prototipos estáticos. (The Thymeleaf Team, 2024).

Boostrap: "Bootstrap es un framework front-end utilizado para desarrollar aplicaciones web y sitios mobile first, o sea, con un layout que se adapta a la pantalla del dispositivo utilizado por el usuario" (Rock Content, 2020).

Motor de Base de Datos

PostgreSQL: "PostgreSQL, comúnmente pronunciado 'Post-GRES', es una base de datos de código abierto que tiene una sólida reputación por su fiabilidad, flexibilidad y soporte de estándares técnicos abiertos" (IBM, 2024).

Otras Tecnologías

GitHub: "GitHub es una plataforma donde puedes almacenar, compartir y trabajar junto con otros usuarios para escribir código" (GitHub, 2024).

RDF: "RDF es un modelo estándar para el intercambio de datos en la Web. RDF tiene características que facilitan la combinación de datos incluso si los esquemas subyacentes difieren" (World Wide Web Consortium, 2014).

SPARQL: "SPARQL contiene capacidades para consultar patrones de gráficos requeridos y opcionales junto con sus conjunciones y disyunciones via RDF" (World Wide Web Consortium, 2013).

HTML5: "La última versión estable de HTML, HTML5 convierte a HTML de un simple formato de marcado para estructurar documentos en una plataforma completa de desarrollo de aplicaciones" (Mozilla Developer Network, 2023).

CSS: "CSS, de las siglas en inglés Cascading Style Sheets (Hojas de Estilo en Cascada), es un lenguaje declarativo que controla el aspecto de las páginas web en el navegador" (Mozilla Developer Network, 2023).

Docker: "Docker empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución" (Amazon Web Services, 2023).

Kubernetes: "Kubernetes es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Kubernetes facilita la automatización y la configuración declarativa"(Kubernetes, 2022).

MinIO: "Minio es un servidor de almacenamiento de objetos distribuidos de alto rendimiento, que está diseñado para la infraestructura de nube privada a gran escala" (IBM, 2021).

Apache Kafka: "Apache Kafka es una plataforma de transmisión de eventos distribuida de código abierto utilizada por miles de empresas para canalizaciones de datos de alto rendimiento, análisis de transmisión, integración de datos y aplicaciones críticas" (Apache Software Fundation, 2024).

IntelliJ IDEA: "IntelliJ IDEA incorpora uno de los editores de código más potentes del sector. Comprende los entresijos de su código gracias a la indexación inicial." (JetBrains, 2024).

Postman: "Postman es una plataforma para crear y utilizar APIs. Postman simplifica cada paso del ciclo de vida y agiliza la colaboración para que puedas crear mejores APIs y más rápido" (Postman, 2024).

Trello: "Trello es una herramienta visual que permite a los equipos gestionar cualquier tipo de proyecto y flujo de trabajo, así como supervisar tareas" (Trello, 2023).

Competencia

A continuación, en la siguiente tabla se exponen y comparan las principales características de las distintas soluciones, similares a la propuesta en el presente proyecto, presentes en el mercado.

Paginas	Generación Automática de Microservicios	Soporte REST	Soporte gRPC	Integración con Sistemas Externos	Soporte para Contenedores
outsystems.com	X	X		X	X
mendix.com	X	X		X	X
appian.com	X	X		X	
microsoft.com/**/power-apps		X		X	
bettyblocks.com	X	X		X	X

Tabla 1: Comparativa de empresas con soluciones Low-code. Fuente: Elaboración Propia.

Diseño Metodológico

Herramientas Metodológicas

Durante el desarrollo del presente sistema se siguieron los lineamientos establecidos por la metodología ágil Scrum, un marco de trabajo que facilita la colaboración entre equipos para entregar productos de manera iterativa e incremental, permitiendo adaptarse rápidamente a los cambios e incentivando la mejora continua (Schwaber & Sutherland, 2010).

De esta forma, al finalizar cada Sprint, se logró obtener porciones funcionales de código, aunque el sistema completo no esté desarrollado, aprendiendo en cada iteración y aplicando el conocimiento en la siguiente a realizar. Así el producto se benefició de las características mencionadas, permitiendo incluso cambiar de tecnologías, en base al conocimiento obtenido durante el proceso de desarrollo sin represalia alguna.

Herramientas de Desarrollo

En el desarrollo del proyecto fueron utilizadas múltiples tecnologías, las cuales serán explicadas y justificadas a continuación, organizadas en 4 grupos según su actividad especifica: herramientas referidas al Front-end, aquellas destinadas al Back-end para autenticación lógica básica e intercambio con el Front-end embebido en el Gateway, por otra parte las tecnologías orientadas al Back-end para la generación automática de código y, finalmente, las tecnologías utilizadas para facilitar el despliegue y escalabilidad del sistema.

En cuanto a las herramientas empleadas en el desarrollo del Front-end se encuentra, el ya estándar conjunto de JavaScript, HTML5 y CSS, permitiendo crear un aspecto visual completo y con alta compatibilidad, además de una correcta y adaptable lógica de comunicación con el Back-end. A su vez, estas herramientas fueron potenciadas con Boostrap, que con la ayuda de sus componentes prediseñados alivianó la carga de trabajo referida al diseño estético, permitiendo gestionar múltiples funcionalidades dentro de la misma página sin descuidar este aspecto. De esta forma y con la ayuda de Thymeleaf el Front-end fue acoplado al API Gateway para evitar complejidades innecesarias.

El Back-end fue desarrollado en su gran mayoría en Java 17, haciendo uso del framework Spring y de sus conocidas librerías para una correcta, limpia y eficiente comunicación entre servicios mediante, en su mayoría, el estilo de arquitectura REST. La seguridad se administró e implementó mediante OAuth2 partiendo del servicio ofrecido por Okta, un estándar que permite a sitios web o aplicaciones acceder a recursos alojados en otras aplicaciones, en nombre y con permiso previo, del usuario. Los aspectos que requirieron persistencia de datos relacional fueron provistos de una instancia de base de datos PostgreSQL, elegida debido a su entorno open source, además de su destacable flexibilidad, integridad de datos y escalabilidad. Además, se manejaron comunicaciones asíncronas para él envió de notificaciones mediante Apache Kafka.

La lógica referida a la generación dinámica de código se administró semánticamente mediante RDF, que permitió especificar de forma correcta y estructurada las combinaciones y decisiones realizadas por el usuario en el canvas presente en el Front-end, para poder así ser enviadas al Back-end, consultadas a través del lenguaje de consulta SPARQL y administradas mediante Apache Jena en los servicios Java. Para concluir el proceso y generar el código especificado y solicitado por el usuario se decidió utilizar Apache Velocity, debido a su alta capacidad en la tarea requerida y MinIO, para el almacenamiento y descarga de los archivos resultantes, evadiendo de esta forma, dependencia a soluciones de almacenamiento en la nube específicas.

Finalmente, la administración del despliegue y escalabilidad del sistema se llevó a cabo mediante Docker, potenciado con Kubernetes. De esta forma, al igual que otras decisiones previamente expuestas, se reduce el acoplamiento a soluciones cloud brindando al sistema una mayor versatilidad y capacidad independiente de despliegue.

Recolección de Datos

Para comprender adecuadamente el problema a abordar, se inició con el análisis de fuentes bibliográficas académicas. Este enfoque permitió identificar los principales desafíos en el desarrollo de arquitecturas distribuidas, así como los costos asociados.

Otro método de recolección de datos utilizado fue el análisis de redes sociales. Esta metodología, debido al potencial público del proyecto, resultó altamente

enriquecedora gracias a plataformas como Twitter, Reddit y Medium, donde desarrolladores de software comparten y discuten ideas y experiencias sobre temas y situaciones del ámbito informático.

De esta forma, se obtuvieron dos enfoques ciertamente antagónicos, justificando de esta manera la elección de las técnicas presentadas. Estos enfoques se dividen en los estrictamente teóricos, basados en documentos académicos, y por el otro extremo, mediante redes sociales, los enfoques basados en las experiencias diarias y cotidianas de los desarrolladores, quienes, al fin y al cabo, son los perjudicados por la problemática.

Planificación del Proyecto

Para facilitar la comprensión, se presentará primero la planificación de forma separada, y luego se mostrará el diagrama de Gantt utilizado para organizar los objetivos del presente Trabajo Final de Graduación.

A continuación, se presentan las tareas especificadas en conjunto con las fechas correspondientes para, posteriormente presentar el diagrama de Gantt mencionado.

Nombre de tarea	Fecha de inici	Fecha final
	07/08/2024	09/11/2024
Temática y Relevamiento	**07/08/2024**	05/09/2024
Elección y Presentación de Temática	07/08/2024	19/08/2024
Título, Introducción y Justificación	20/08/2024	22/08/2024
Objetivo General y Espeíficos	23/08/2024	25/08/2024
Marco Teórico Referencial	26/08/2024	28/08/2024
Diseño Metodologico	29/08/2024	31/08/2024
Relevamiento	01/09/2024	03/09/2024
Proceso de Negocios	04/09/2024	05/09/2024
Propuesta y Definición del Prototipo	**06/09/2024**	23/09/2024
Diagnóstico y Propuesta	06/09/2024	08/09/2024
Objetivos, Límites y Alcances	09/09/2024	11/09/2024
Descripción del Sistema	12/09/2024	15/09/2024
Estructura de Datos	16/09/2024	18/09/2024
Prototipos de Interfaces	19/09/2024	22/09/2024
Diagrama de Arquitectura	23/09/2024	23/09/2024
Riesgos y Control	**24/09/2024**	01/10/2024
Documentación de Seguridad	24/09/2024	25/09/2024
Análisis de Costos	26/09/2024	28/09/2024
Análisis de Riesgos	29/09/2024	01/10/2024
Finalización de Entregable	**02/10/2024**	06/10/2024
Conclusiones y Anexos	02/10/2024	04/10/2024
Revisión y Corrección General	05/10/2024	06/10/2024
Desarrollo y Pruebas del Prototipo	**23/09/2024**	09/11/2024
Desarrollo del Prototipo	23/09/2024	05/11/2024
Pruebas del Prototipo	02/11/2024	09/11/2024

Figura 1: Plan de Desarrollo. Fuente: Elaboración Propia.

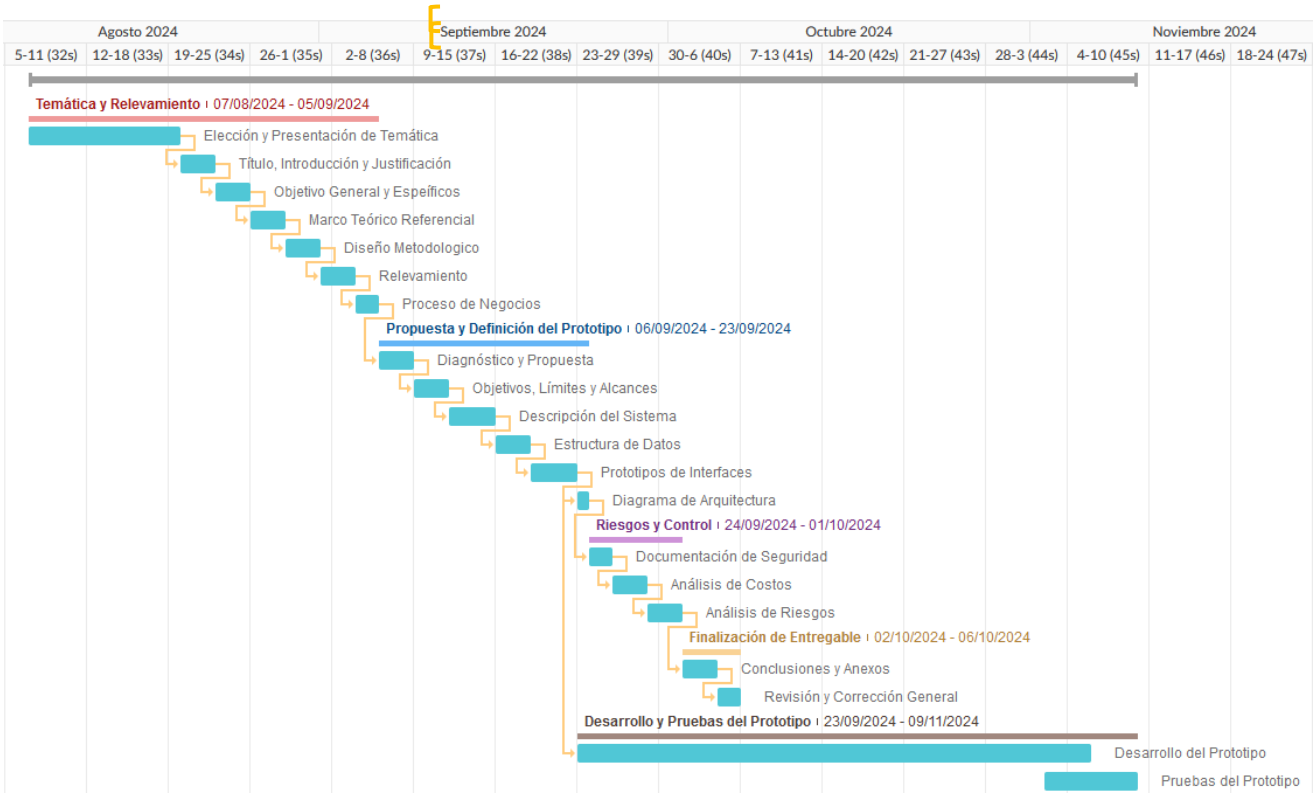

Figura 2: Diagrama de Gantt. Fuente: Elaboración Propia.

Relevamiento

Relevamiento Estructural

Dado que el proyecto desarrollado se trató de una plataforma web dirigida a desarrolladores de software, tanto empleados como freelancers, no es posible establecer una localización específica, ya que este aspecto depende de cada usuario particular que acceda a la plataforma.

Sin embargo, se ha identificado mediante los métodos de recolección de datos especificados previamente, que la mayoría de los desarrolladores utilizan, propias o de la empresa empleadora, computadoras portátiles, y en menor medida, equipos de escritorio, a través de los cuales realizan sus tareas profesionales en el ámbito informático.

Relevamiento Funcional

Estructura Jerárquica

A continuación, se presenta el organigrama genérico de una empresa pequeña o mediana de desarrollo de software, debido a la naturaleza de usuarios a los que apunta el proyecto.

Se encuentran coloreados los sectores alcanzados por la plataforma desarrollada.

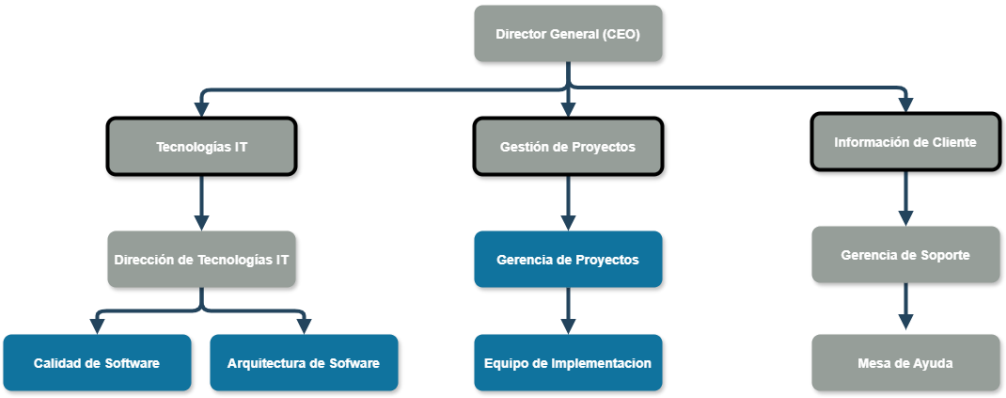

Figura 3: Organigrama Modelo de Empresa de Desarrollo de Software. Fuente: Elaboración Propia.

Funciones de las Áreas

Calidad de Software: Se encarga de asegurar que el software cumple con los requisitos previamente especificados y que este no posee errores. También implementa pruebas y controla la calidad del producto de forma continua durante el ciclo de vida del desarrollo.

Arquitectura de Software: Tiene el objetivo de definir la estructura general del sistema, incluyendo las tecnologías, patrones de diseño y las interacciones entre los diferentes servicios que componen el producto final. A su vez se enfoca en asegurar escalabilidad y eficiencia en el software.

Equipo de Implementación: Desarrolla, implementa y mantiene el producto de software. Realiza la tarea de escribir el código, solucionar errores y participar en la creación de nuevas funcionalidades indicadas por los requerimientos y especificaciones de diseño.

Gerencia de Proyectos: Establece que recursos se necesitan para realizar el proyecto y supervisa la ejecución y desarrollo del mismo, a su vez, asigna las tareas correspondientes y define sus plazos.

Procesos Relevados

Nombre de proceso: Control de Calidad del Software

Roles: Calidad de Software, Equipo de Implementación, Arquitectura de Software, Gerencia de Proyectos.

Pasos: El departamento de Calidad de Software establece y documenta los estándares y criterios de calidad que deben cumplir los desarrollos de software, de forma coordinada con el departamento de Arquitectura de Software para asegurar la alineación técnica y de diseño.

Luego, el equipo de Calidad de Sofware desarrolla un plan de pruebas detallado, el cual es compartido con el Equipo de Implementación para garantizar que todas las funcionalidades sean cubiertas y probadas mediante las pruebas especificadas.

Finalmente, el equipo de Calidad de Software realiza las pruebas y evalúa los resultados para, si es necesario realizar una retroalimentación al Equipo de Implementación. En caso contrario la Gerencia de Proyectos recibe el producto y se encarga de su liberación final.

Nombre de proceso: Diseño de Arquitectura de Software

Roles: Arquitectura de Software, Calidad de Software, Gerencia de Proyectos.

Pasos: La Gerencia de Proyectos, trabaja para comprender y documentar los diferentes requisitos del proyecto a tratar.

Una vez definidos los requisitos, el Departamento de Arquitectura de Software diseña y documenta la arquitectura de forma completa, incluyendo patrones de diseño, frameworks y tecnologías. Una vez realizado se comunica el resultado al Equipo de Calidad.

El Departamento de Calidad verifica los estándares de calidad de la arquitectura desarrollada, invitando a realizar ajustes en caso de ser necesario.

Nombre de proceso: Desarrollo de Software

Roles: Equipo de Implementación, Gerencia de Proyectos, Arquitectura de Software, Calidad de Software.

Pasos: La Gerencia de Proyectos distribuye tareas y asigna recursos al Equipo de Implementación según el cronograma y las prioridades que hayan sido asignadas a cada tarea del proyecto.

El equipo de Implementación desarrolla las funcionalidades según el diseño arquitectónico y sus especificaciones detalladas por el departamento de Arquitectura de Software, estableciendo comunicación mutua de forma constante para garantizar la comprensión y cumplimiento de los lineamientos.

Previamente a enviar los módulos desarrollados a Calidad de Software, el Equipo de Implementación realiza pruebas internas para detectar y corregir errores. Una vez solucionados se proceden a enviar los módulos completos al departamento mencionado, quienes procederán con la realización de pruebas formales.

Cualquier defecto identificado por el Equipo de Calidad es informado, mediante un informe de pruebas, al Equipo de Implementación, quienes realizan las correcciones pertinentes. Este paso se trata de un ciclo que finaliza cuando se considera que los estándares de calidad especificados han sido cumplidos.

Relevamiento de la Documentación

A continuación, se mencionan los documentos relevados:

- **Especificación de Estándares de Calidad:** Documento que describe los criterios y estándares de calidad que deben cumplir los desarrollos de software según lo definido por el equipo de Calidad de Software (Ver Anexo 1)

- **Plan de Pruebas:** Documento que detalla el plan de pruebas, incluyendo tipos de pruebas, casos de prueba, y recursos asignados. Este plan es desarrollado por el equipo de Calidad de Software (Ver Anexo 2).
- **Relevamiento de Requisitos:** Documento que lista los requisitos recopilados por la Arquitectura de Software en colaboración con la Gerencia de Proyectos. Es la base para el diseño de la arquitectura del software (Ver Anexo 3).
- **Documentación de Arquitectura:** Documento que describe en detalle la arquitectura del software, incluyendo diagramas de componentes, interacciones y tecnologías utilizadas (Ver Anexo 4).
- **Informe de Pruebas:** Documento generado por el equipo de Calidad de Software que resume los resultados de las pruebas realizadas, incluyendo defectos encontrados y acciones correctivas (Ver Anexo 5).

Proceso de Negocios

A continuación, se presenta el diagrama de flujo, referido al proceso integral llevado a cabo al momento de desarrollar un producto de software.

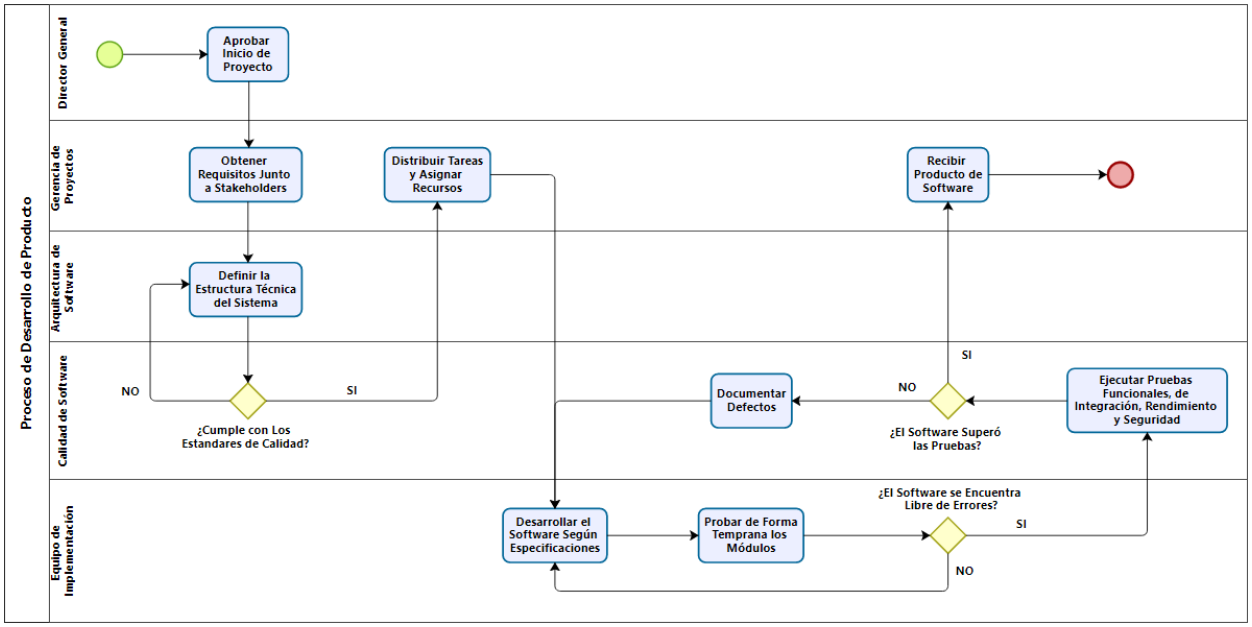

Figura 4: Diagrama de Flujo referido al Desarrollo de Producto. Fuente: Elaboración Propia.

Diagnóstico y Propuesta

Diagnóstico

Nombre del Proceso: Control de Calidad de Software	
Problemas	Causas
1. Dependencia de pruebas manuales.	1. Presencia de componentes incorrectamente desacoplados, provocando la imposibilidad de automatizar pruebas de componentes aislados. 2. Escasa documentación o ausencia de la misma referida a la composición y lógica interna de los componentes pertenecientes al producto de software.

Tabla 2: Diagnostico de proceso Control de Calidad de Software. Fuente: Elaboración Propia.

Nombre del Proceso: Diseño de Arquitectura de Software	
Problemas	Causas
1. Dificultad para adaptar la arquitectura a nuevos requisitos del proyecto.	1. Los cambios en los requisitos pueden provocar que la arquitectura original, o la mayoría de la misma, quede obsoleta o no cubra las necesidades emergentes. 2. La presencia de una arquitectura monolítica o poco modular limita la flexibilidad para adaptar partes específicas del sistema sin afectar otras. 3. Falta de la documentación de la arquitectura o inaccesibilidad a la misma.
2. Falta de coordinación en la definición de la arquitectura entre equipos.	1. Los distintos departamentos no siempre están correctamente alineados con las decisiones arquitectónicas.

Tabla 3: Diagnostico de proceso Diseño de Arquitectura de Software. Fuente: Elaboración Propia.

Nombre del Proceso: Desarrollo de Software	
Problemas	Causas
1. Sobrecarga del equipo de desarrollo por falta de reutilización software.	1. No se cuenta con una estrategia clara o un repositorio centralizado para gestionar componentes reutilizables.
2. Identificación tardía de errores en el código antes de la entrega a calidad.	1. Debido a la falta de recursos de prueba continua, los errores suelen detectarse tarde en el ciclo de desarrollo.

Tabla 4: Diagnostico de proceso de Desarrollo de Software. Fuente: Elaboración Propia.

Propuesta

El sistema desarrollado mejoró la eficiencia y claridad en el diseño y desarrollo de componentes de software. Permite al usuario crear arquitecturas de componentes altamente desacoplados y obtener el código de cada componente con documentación centralizada, accesible y clara. Esto facilita la coordinación entre equipos y asegura la alineación técnica en todas las fases del proyecto. La plataforma incorpora pruebas automáticas desde el inicio de la creación de componentes, reduciendo significativamente la dependencia de pruebas manuales y mejorando los procesos de testing y calidad. Además, el desacople lógico de los componentes fomenta su reutilización, acelerando el desarrollo y mejorando la adaptabilidad de las arquitecturas resultantes.

Objetivos, Límites y Alcance del Prototipo

Objetivos del Prototipo

Desarrollar un prototipo de sistema que permita el diseño de arquitecturas distribuidas mediante la combinación de componentes predefinidos, generando automáticamente el código ajustado a las especificaciones del usuario para su posterior descarga.

Límites

Desde la configuración y combinación de componentes predefinidos hasta la generación y descarga automática del código correspondiente según las especificaciones del usuario.

Alcances

A continuación, se listan los procesos comprendidos por el prototipo tecnológico:

- Registro e ingreso de usuarios.
- Diseño de arquitecturas mediante la combinación de microservicios.
- Gestión de composición de microservicios.
- Guías de usuario.
- Generación dinámica de componentes.
- Generación de documentación técnica.
- Validación y pruebas de la arquitectura generada.
- Descarga de la arquitectura resultante

Descripción del Sistema

Product Backlog

A continuación, se presenta el Product Backlog del prototipo a desarrollar. Cabe aclarar que las historias de usuario expuestas corresponden únicamente a las funcionalidades básicas necesarias para exponer el propósito del sistema. Es por este motivo que únicamente se presentan componentes personalizables del tipo ecommerce, ya que así se logran exponer las funcionalidades referidas a la combinación y personalización de componentes compatibles dentro del canvas de la aplicación.

ID	Historia de Usuario	Prioridad	Puntos de Historia	Dependencias
HU-001	Registro a la plataforma vía Google	Alta	5	-
HU-002	Iniciar sesión vía Google	Alta	2	HU-001
HU-003	Cerrar sesión de usuario	Media	2	HU-002
HU-004	Diseño visual drag-and-drop	Media	5	-
HU-005	Plantilla de servicio de usuarios	Alta	5	-
HU-006	Servicio de usuarios	Alta	2	HU-005

HU-007	Plantilla de servicio de catálogo de productos	Alta	5	-
HU-008	Servicio de catálogo de productos	Alta	3	HU-007
HU-009	Plantilla de servicio de carrito de compras	Alta	5	-
HU-010	Servicio de carrito de compras	Alta	3	HU-009
HU-011	Plantilla de servicio de ordenes	Alta	5	-
HU-012	Servicio de órdenes	Alta	3	HU-011
HU-013	Plantilla de servicio de envíos	Alta	5	-
HU-014	Servicio de envíos	Media	3	HU-013
HU-015	Plantilla de servicio de notificación	Alta	8	-
HU-016	Servicio de notificaciones	Media	3	HU-015
HU-017	Interactuar con componentes disponibles	Media	5	HU-006
HU-018	Compatibilidad de componentes	Media	5	HU-017
HU-019	Eliminar componentes del diseño	Baja	3	HU-018
HU-020	Seleccionar persistencia en componentes	Baja	3	HU-016
HU-021	Configurar endpoints en componentes	Media	3	HU-016
HU-022	Seleccionar metodología de comunicación	Media	8	HU-016
HU-023	Especificar seguridad JWT	Alta	3	HU-016
HU-024	Especificar configuración global	Media	5	HU-016
HU-025	Especificar API Gateway	Media	8	HU-016
HU-026	Traducción de JSON a RDF	Alta	5	HU-025
HU-027	ZIP de la arquitectura resultante	Baja	8	HU-026
HU-028	Estado de proceso de generación dinámica	Baja	5	HU-027
HU-029	Acceder a archivos ZIP	Baja	3	HU-027
HU-030	Dockerfiles de componentes	Alta	3	HU-026
HU-031	Guía de tareas pendientes	Baja	5	HU-025
HU-032	Documentación de componentes	Media	5	HU-025
HU-033	Guía de uso de la plataforma	Media	5	HU-032

Tabla 5: Product Backlog. Fuente: Elaboración Propia.

Historias de Usuario

ID	**HU-001**	**Nombre**	**Registro a la plataforma vía Google**
Descripción		Como usuario, quiero registrarme en el sistema mediante mi cuenta de Google, para poder crear mi perfil.	
Criterios de Aceptación		• Dado un usuario de Google existente, cuando este decida registrarse, el sistema debe redireccionarlo a una pestaña	

	para brindar los permisos necesarios, permitiendo el registro e informando al usuario, una vez completada la acción. • Dado un usuario de Google previamente registrado mediante su cuenta en la plataforma, cuando este sea provisto para registrarse, el sistema debe informar que ya existe una cuenta registrada con ese usuario, invitándolo a iniciar sesión mediante un hipervínculo a la sección correspondiente.		
Prioridad	Alta	Puntos de historia estimados	5

Tabla 6: HU-001 Registro a la plataforma vía Google. Fuente: Elaboración Propia.

ID	**HU-002**	**Nombre**	**Iniciar sesión vía Google**
Descripción		Como usuario, quiero iniciar sesión en el sistema mediante mi cuenta de Google, para acceder a la plataforma y a mis arquitecturas previas.	
Criterios de Aceptación		• Dado un usuario previamente registrado mediante su cuenta de Google, cuando ingrese mediante la cuenta mencionada, el sistema debe otorgar acceso, redirigiéndolo al canvas de la plataforma. • Dado un usuario que no se encuentre registrado mediante su cuenta de Google, cuando intente ingresar mediante la cuenta no registrada, el sistema debe informar que no se encuentra registrado, invitándolo a hacerlo mediante un hipervínculo a la sección correspondiente.	
Prioridad		Alta	Puntos de historia estimados 2

Tabla 7: HU002 Iniciar sesión vía Google. Fuente: Elaboración Propia.

ID	**HU-003**	**Nombre**	**Cerrar sesión de usuario**
Descripción		Como usuario, quiero cerrar sesión en el sistema mediante mi cuenta de Google, para evitar accesos no deseados a mi perfil.	

Criterios de Aceptación	• Dado un usuario ingresado en el sistema mediante credenciales validas, cuando decida cerrar su sesión, el sistema debe anular las credenciales temporales que permiten su acceso en la sesión actual.		
Prioridad	Media	Puntos de historia estimados	2

Tabla 8: HU-003 Cerrar sesión de usuario. Fuente: Elaboración Propia.

ID	**HU-004**	**Nombre**	**Diseño visual drag-and-drop**
Descripción		Como usuario, quiero arrastrar componentes al canvas para poder diseñar una arquitectura de forma visual.	
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión, cuando el usuario presione y arrastre al canvas la representación visual de un componente, el sistema debe moverlo al área donde el usuario deposite el elemento.	
Prioridad		Media	Puntos de historia estimados	5

Tabla 9: HU-004 Diseño visual drag-and-drop. Fuente: Elaboración Propia.

ID	**HU-005**	**Nombre**	**Plantilla de Servicio de Usuarios**
Descripción		Como usuario, quiero que el sistema utilice una plantilla predefinida para la creación automática de servicios de usuarios, de manera que se generen todos los componentes necesarios para gestionar usuarios y permisos sin intervención manual.	
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, cuando genere una arquitectura que contenga "Servicio de Usuarios" dentro del canvas de diseño, el sistema debe asociar automáticamente el servicio a una plantilla predefinida para la gestión de usuarios. • Dado un usuario que ha iniciado sesión en la plataforma, cuando combine cierto servicio compatible a “Servicio de Usuarios”, el sistema debe adaptar la generación de código de forma acorde para una correcta interacción.	

Prioridad	Alta	Puntos de historia estimados	5

Tabla 10: HU-005 Plantilla de servicio de usuarios. Fuente: Elaboración Propia.

ID	**HU-006**	**Nombre**	**Servicio de usuarios**
Descripción		Como usuario, quiero disponer de un servicio de manejo de usuarios para incorporar en mi arquitectura, para poder manejar su información y permisos.	
Criterios de Aceptación		• Dado que un usuario ha iniciado sesión en la plataforma, cuando arrastre el componente visual "Usuarios" al canvas de diseño, el sistema debe asociar automáticamente el servicio de órdenes a la plantilla de código predefinida.	
Prioridad		Alta	Puntos de historia estimados 3

Tabla 11: HU-006 Servicio de usuarios. Fuente: Elaboración Propia.

ID	**HU-007**	**Nombre**	**Plantilla de Servicio de catálogo de productos**
Descripción		Como usuario, quiero que el sistema utilice una plantilla predefinida para la creación automática de un servicio de catálogo de productos, para que pueda exponer mi inventario actual.	
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, cuando genere una arquitectura que contenga "Servicio de Catálogo de Productos" dentro del canvas de diseño, el sistema debe asociar automáticamente el servicio a una plantilla predefinida para la gestión de usuarios. • Dado un usuario que ha iniciado sesión en la plataforma, cuando combine cierto servicio compatible a “Servicio de Catálogo de Productos”, el sistema debe adaptar la generación de código de forma acorde para una correcta interacción.	
Prioridad		Alta	Puntos de historia estimados 5

Tabla 12: HU-007 Plantilla de servicio de catálogo de productos. Fuente: Elaboración Propia.

ID	HU-008	Nombre	Servicio de catálogo de productos	
Descripción		Como usuario, quiero disponer de un servicio de catálogo de productos para incorporar en mi arquitectura, para poder exponer mi stock actual al público.		
Criterios de Aceptación		• Dado que un usuario ha iniciado sesión en la plataforma, cuando arrastre el componente visual "Catálogo de Productos" al canvas de diseño, el sistema debe asociar automáticamente el servicio de órdenes a la plantilla de código predefinida.		
Prioridad		Alta	Puntos de historia estimados	3

Tabla 13: HU-008 Servicio de catálogo de productos. Fuente: Elaboración Propia.

ID	HU-009	Nombre	Plantilla de Servicio de carrito de compras	
Descripción		Como usuario, quiero que el sistema utilice una plantilla predefinida para la creación automática de un servicio de carrito de compras, para que los usuarios puedan realizar compras de múltiples productos.		
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, cuando genere una arquitectura que contenga "Servicio de Carrito de Compras" dentro del canvas de diseño, el sistema debe asociar automáticamente el servicio a una plantilla predefinida para la gestión de usuarios. • Dado un usuario que ha iniciado sesión en la plataforma, cuando combine cierto servicio compatible a “Servicio de Carrito de Compras”, el sistema debe adaptar la generación de código de forma acorde para una correcta interacción.		
Prioridad		Alta	Puntos de historia estimados	5

Tabla 14: HU-009 Plantilla de servicio carrito de compras. Fuente: Elaboración Propia.

ID	HU-010	Nombre	Servicio de carrito de compras	
Descripción		Como usuario, quiero disponer de un servicio de carrito de compras para incorporar en mi arquitectura, para que los usuarios de mi sistema puedan realizar compras de múltiples artículos simultáneamente.		
Criterios de Aceptación		• Dado que un usuario ha iniciado sesión en la plataforma, cuando arrastre el componente visual "Carrito de Compras" al canvas de diseño, el sistema debe asociar automáticamente el servicio de órdenes a la plantilla de código predefinida.		
Prioridad		Alta	Puntos de historia estimados	3

Tabla 15: HU-010 Servicio de carrito de compras. Fuente: Elaboración Propia.

ID	HU-011	Nombre	Plantilla de Servicio de órdenes	
Descripción		Como usuario, quiero que el sistema utilice una plantilla predefinida para la creación automática de un servicio de órdenes, para gestionar el proceso de emisión y administración de órdenes.		
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, cuando genere una arquitectura que contenga "Servicio de Ordenes" dentro del canvas de diseño, el sistema debe asociar automáticamente el servicio a una plantilla predefinida para la gestión de usuarios. • Dado un usuario que ha iniciado sesión en la plataforma, cuando combine cierto servicio compatible a “Servicio de Ordenes”, el sistema debe adaptar la generación de código de forma acorde para una correcta interacción.		
Prioridad		Alta	Puntos de historia estimados	5

Tabla 16: HU-011 Plantilla de servicio de órdenes. Fuente: Elaboración Propia.

ID	HU-012	Nombre	Servicio de órdenes	
Descripción		Como usuario, quiero disponer de un servicio capaz de administrar órdenes para incorporar a mi arquitectura, para manejar las ordenes emitidas dentro del sistema y actuar en consecuencia.		
Criterios de Aceptación		• Dado que un usuario ha iniciado sesión en la plataforma, cuando arrastre el componente visual "Órdenes" al canvas de diseño, el sistema debe asociar automáticamente el servicio de órdenes a la plantilla de código predefinida.		
Prioridad		Alta	Puntos de historia estimados	3

Tabla 17: HU-012 Servicio de órdenes. Fuente: Elaboración Propia.

ID	HU-013	Nombre	Plantilla de Servicio de envíos	
Descripción		Como usuario, quiero que el sistema utilice una plantilla predefinida para la creación automática de un servicio de envíos, de manera que pueda gestionar el estado de los envíos.		
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, cuando genere una arquitectura que contenga "Servicio de Envíos" dentro del canvas de diseño, el sistema debe asociar automáticamente el servicio a una plantilla predefinida para la gestión de usuarios. • Dado un usuario que ha iniciado sesión en la plataforma, cuando combine cierto servicio compatible a “Servicio de Envíos”, el sistema debe adaptar la generación de código de forma acorde para una correcta interacción.		
Prioridad		Alta	Puntos de historia estimados	5

Tabla 18: HU-013 Plantilla de servicio de envíos. Fuente: Elaboración Propia.

ID	HU-014	Nombre	Servicio envíos
Descripción		Como usuario, quiero disponer de un servicio capaz de administrar el estado de los envíos de mi sistema para incorporar a mi arquitectura, para poder rastrear y actualizar el estado de los envíos.	

Criterios de Aceptación	• Dado que un usuario ha iniciado sesión en la plataforma, cuando arrastre el componente visual "Envíos" al canvas de diseño, el sistema debe asociar automáticamente el servicio de órdenes a la plantilla de código predefinida.		
Prioridad	Media	Puntos de historia estimados	3

Tabla 19: HU-014 Servicio de envíos. Fuente: Elaboración Propia.

ID	**HU-015**	**Nombre**	**Plantilla de Servicio de notificaciones**
Descripción		Como usuario, quiero que el sistema utilice una plantilla predefinida para la creación automática de un servicio de notificaciones, de manera que pueda enviar alertas y mensajes a los usuarios.	
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, cuando genere una arquitectura que contenga "Servicio de Notificaciones" dentro del canvas de diseño, el sistema debe asociar automáticamente el servicio a una plantilla predefinida para la gestión de usuarios. • Dado un usuario que ha iniciado sesión en la plataforma, cuando combine cierto servicio compatible a “Servicio de Notificaciones”, el sistema debe adaptar la generación de código de forma acorde para una correcta interacción.	
Prioridad		Alta	Puntos de historia estimados	8

Tabla 20: HU-015 Plantilla de servicio de notificaciones. Fuente: Elaboración Propia.

ID	**HU-016**	**Nombre**	**Servicio de notificaciones**
Descripción		Como usuario, quiero disponer de un servicio de notificaciones para incorporar en mi arquitectura, para enviar mensajes y alertas a los usuarios del sistema sobre eventos importantes.	
Criterios de Aceptación		• Dado que un usuario ha iniciado sesión en la plataforma, cuando arrastre el componente visual "Notificaciones" al canvas de diseño, el sistema debe asociar automáticamente el servicio de órdenes a la plantilla de código predefinida.	

Prioridad	Media	Puntos de historia estimados	3

Tabla 21: HU-016 Servicio de notificaciones. Fuente: Elaboración Propia.

ID	**HU-017**	**Nombre**	**Interactuar con componentes disponibles**
Descripción		Como usuario, quiero visualizar los componentes disponibles para poder seleccionarlos y combinarlos en el canvas según mis necesidades.	
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, cuanto deposite un componente en el canvas y lo desee unir con otro compatible, el sistema debe crear una unión grafica mediante una línea para expresar la relación exitosa. • Dado un usuario que ha iniciado sesión en la plataforma, cuando deposite un componente en el canvas y lo desee unir con otro no compatible, el sistema debe presentar mediante "pop up" que la combinación no es legal, invitando al usuario a revisar la documentación de la plataforma.	
Prioridad		Media	Puntos de historia estimados 5

Tabla 22: HU-017 Visualizar componentes disponibles. Fuente: Elaboración Propia.

ID	**HU-018**	**Nombre**	**Compatibilidad de componentes**
Descripción		Como usuario, quiero ver con que componentes es compatible el seleccionado, para combinarlos correctamente en el canvas.	
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, cuando interactúe sobre el botón "+" anexado al componente presente en el canvas, el sistema debe destacar los botones "+" del resto de componentes compatibles presentes en el canvas. • Dado un usuario que ha iniciado sesión en la plataforma, cuando destaque las compatibilidades de un componente y vuelva a tocar el canvas, el sistema debe desmarcar las conexiones previamente remarcadas.	

Prioridad	Media	Puntos de historia estimados	5

Tabla 23: HU-018 Compatibilidad de componentes. Fuente: Elaboración Propia.

ID	**HU-019**	**Nombre**	**Eliminar componentes del diseño**
Descripción		Como usuario, quiero eliminar un componente del canvas para quitarlo de la arquitectura diseñada.	
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, cuando ubique el ratón sobre un componente desplegado en el canvas, el sistema debe presentar un icono “X” referido a la opción de eliminar el componente. • Dado un usuario que ha iniciado sesión en la plataforma, cuando interactúe con el botón “X” de un componente en el canvas, el sistema debe consultar si desea eliminar el componente, en caso positivo ejecuta la acción y en caso negativo quita el pop-up de consulta.	
Prioridad		Baja	Puntos de historia estimados 3

Tabla 24: HU-019 Eliminar componentes del diseño. Fuente: Elaboración Propia.

ID	**HU-020**	**Nombre**	**Seleccionar persistencia en componentes**
Descripción		Como usuario, quiero configurar la base de datos de un componente para poder elegir la que mejor se adapta a los requerimientos de mi arquitectura.	
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, cuando interactúe con la base de datos predeterminada adjunta al componente dentro del canvas, el sistema debe desplegar las opciones alternativas presentes para el servicio, permitiendo modificar la predeterminada.	
Prioridad		Baja	Puntos de historia estimados 3

Tabla 25: HU-020 Seleccionar persistencia en componentes. Fuente: Elaboración Propia.

ID	HU-021	Nombre	Configurar endpoints en componentes
Descripción		Como usuario, quiero configurar la URL de los endpoints de un componente para adecuarlo a mis requerimientos de infraestructura.	
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, cuando deposite el ratón sobre un componente en el canvas, el sistema debe desplegar un botón representando una "tuerca" para acceder a un pop-up con las configuraciones del servicio. • Dado un usuario que ha iniciado sesión en la plataforma, cuando interactúe con el botón de configuración de un microservicio, el sistema debe presentar la opción de modificar el prefijo de los endpoints del microservicio, aplicando los cambios mediante un botón "aceptar". • Dado un usuario que ha iniciado sesión en la plataforma, cuando ingrese una cadena no valida como URL como prefijo de endpoints y presione "aceptar", el sistema debe notificar que no se realizó ningún cambio debido a caracteres inválidos, manteniendo al usuario en el pop-up mostrando la cadena previa al ingreso ilegal.	
Prioridad		Media	Puntos de historia estimados: 3

Tabla 26: HU-021 Configurar endpoints en componentes. Fuente: Elaboración Propia.

ID	HU-022	Nombre	Seleccionar metodología de comunicación
Descripción		Como usuario, quiero seleccionar la metodología de comunicación entre dos componentes para que se adapten a las necesidades de mi sistema.	
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, al crear una conexión entre 2 componentes compatibles, el sistema debe presentar en medio de la conexión un símbolo que al interactuar con el desplegara una lista con los tipos de conexión compatibles entre ambos microservicios permitiendo seleccionar la opción deseada.	

Prioridad	Media	Puntos de historia estimados	8

Tabla 27: HU-022 Seleccionar metodología de comunicación. Fuente: Elaboración Propia.

ID	**HU-023**	**Nombre**	**Especificar seguridad JWT**
Descripción		Como usuario, quiero especificar la utilización de JWT como medio de seguridad para que la seguridad de la arquitectura se ajuste a mis requerimientos.	
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, al acceder al canvas, el sistema debe exponer en la esquina superior derecha una opción global para asegurar la arquitectura mediante JWT, permitiendo desactivarlo o activarlo.	
Prioridad		Alta	Puntos de historia estimados 3

Tabla 28: HU-023 Especificar seguridad JWT. Fuente: Elaboración Propia.

ID	**HU-024**	**Nombre**	**Especificar configuración global**
Descripción		Como usuario, quiero especificar la utilización de un servidor de configuración global para que la arquitectura se ajuste a mis requerimientos.	
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, al acceder al canvas, el sistema debe exponer en la esquina superior derecha, debajo de “JWT” una opción para administrar la totalidad de las configuraciones de los servicios mediante un configuration server, permitiendo desactivarlo o activarlo.	
Prioridad		Media	Puntos de historia estimados 5

Tabla 29: HU-024 Especificar configuración global. Fuente: Elaboración Propia.

ID	**HU-025**	**Nombre**	**Especificar API Gateway**
Descripción		Como usuario, quiero especificar la generación de un API gateway para que los componentes interactúen de forma directa	

Criterios de Aceptación	• Dado un usuario que ha iniciado sesión en la plataforma, al acceder al canvas, el sistema debe exponer en la esquina superior derecha, debajo de "Config Server" una opción para administrar la totalidad de las interacciones de la arquitectura mediante un API gateway, permitiendo desactivarlo o activarlo.		
Prioridad	Media	Puntos de historia estimados	8

Tabla 30: HU-025 Especificar API Gateway. Fuente: Elaboración Propia.

ID	**HU-026**	**Nombre**	**Traducción de JSON a RDF**
Descripción		Como usuario, quiero que el sistema convierta automáticamente los datos que especifican la arquitectura generada de formato JSON a RDF para garantizar la una especificación estructurada.	
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, al solicitar el procesamiento de la arquitectura resultante, el sistema debe representar la información mediante un JSON, traduciéndola posteriormente en contenido semántico RDF para brindar estructura y posibilidad de realizar consultas.	
Prioridad		Alta	Puntos de historia estimados 5

Tabla 31: HU-026 Traducción de JSON a RDF. Fuente: Elaboración Propia.

ID	**HU-027**	**Nombre**	**ZIP de la arquitectura resultante**
Descripción		Como usuario, quiero solicitar un archivo ZIP desde el canvas con la arquitectura diseñada para obtener los archivos que la componen y así configurarlo en mi entorno local.	
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, al configurar una arquitectura y solicitar su generación, el sistema debe convertir el código generado en un archivo ZIP, informando al usuario cuando se haya completado el proceso, invitándolo a descargarlo desde su perfil de usuario.	

Prioridad	Baja	Puntos de historia estimados	8

Tabla 32: HU-027 ZIP de la arquitectura resultante. Fuente: Elaboración Propia.

ID	**HU-028**	**Nombre**	**Estado de proceso de generación dinámica**
Descripción		Como usuario, quiero conocer en qué estado se encuentra la generación dinámica de mi arquitectura para estimar cuando concluirá.	
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, al configurar una arquitectura y solicitar su generación, el sistema debe presentar un pop-up indicando al usuario en qué estado se encuentra el proceso, mostrando los estados: o Creando los pom.xml o Generando el código o Comprimiendo la arquitectura o Proceso finalizado	
Prioridad		Baja	Puntos de historia estimados 5

Tabla 33: HU-028 Estado de proceso de generación dinámica. Fuente: Elaboración Propia.

ID	**HU-029**	**Nombre**	**Acceder a archivos ZIP**
Descripción		Como usuario, quiero acceder a mis archivos ZIP generados previamente para poder volver a descargarlos	
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, al seleccionar el símbolo de perfil presente en la esquina superior derecha, el sistema debe redireccionar al usuario a una pantalla donde se presentan sus últimas 5 arquitecturas generadas, permitiendo descargarlas con un simple clic en el botón adosado a la derecha de cada una.	
Prioridad		Baja	Puntos de historia estimados 3

Tabla 34: HU-029 Acceder a archivos ZIP previos. Fuente: Elaboración Propia.

ID	HU-030	Nombre	Dockerfiles de componentes
Descripción		Como usuario, quiero disponer de Dockerfiles asociados a los componentes de la arquitectura para facilitar su despliegue.	
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, al obtener el zip resultante de una arquitectura, el sistema debe incluir dentro de ese archivo, un Dockerfile individual por cada componente de la arquitectura, para permitir su despliegue y el de su componente de persistencia.	
Prioridad		Alta	Puntos de historia estimados 3

Tabla 35: HU-030 Dockerfiles de componentes. Fuente: Elaboración Propia.

ID	HU-031	Nombre	Guía de tareas pendientes
Descripción		Como usuario, quiero obtener una guía de tareas pendientes (TO-DO's) en el código generado para ajustar y personalizar la arquitectura descargada.	
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, al obtener el zip resultante de una arquitectura, el sistema debe incluir dentro de ese archivo, un archivo .txt explicando los TO-DO’s presentes en el código y su motivo de existencia.	
Prioridad		Baja	Puntos de historia estimados 5

Tabla 36: HU-031 Guía de tareas pendientes. Fuente: Elaboración Propia.

ID	HU-032	Nombre	Documentación de componentes
Descripción		Como usuario, quiero acceder a la documentación de componentes para entender su estructura y funcionamiento detallado.	
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, al presionar el apartado presente referido a la documentación en la zona superior de la pantalla, el sistema debe redireccionarlo a una pantalla con un menú lateral donde se listen los componentes disponibles.	

	• Dado un usuario que ha iniciado sesión en la plataforma, al presionar un componente especifico en la documentación, el sistema debe exponer la misma, especificando su funcionalidad, compatibilidades y endpoints.		
Prioridad	Media	Puntos de historia estimados	5

Tabla 37: HU-032 Documentación de componentes. Fuente: Elaboración Propia.

ID	**HU-033**	**Nombre**	**Guía de uso de la plataforma**	
Descripción		Como usuario, quiero acceder a una guía de uso de la plataforma para comprender su funcionamiento general y sacarle el máximo provecho.		
Criterios de Aceptación		• Dado un usuario que ha iniciado sesión en la plataforma, al presionar el apartado presente referido a la instrucción de uso de la plataforma en la zona superior de la pantalla, el sistema debe dirigirlo a la pantalla correspondiente, donde se presente una breve guía de uso de la plataforma.		
Prioridad		Media	Puntos de historia estimados	5

Tabla 38: HU-033 Guía de uso de la plataforma. Fuente: Elaboración Propia.

Sprint Backlog

En la siguiente tabla, se presenta el Backlog referido al primer Sprint del prototipo a desarrollar con una duración establecida de 14 días.

Sprint	Historia de Usuario	Tareas	Prioridad	Estimado (días)	Estado
1	Registro a la plataforma vía Google (HU-001)	Definir e implementar metodología y estructura de persistencia de datos	Alta	1	Hecho
1		Codificar e integrar módulo de registro y conexión con Google OAUTH2	Alta	2	Hecho
1		Diseñar interfaz grafica	Media	1	Hecho
1		Realizar testing unitario al módulo desarrollado	Alta	1	Hecho

1	Iniciar sesión vía Google (HU-002)	Codificar e integrar módulo de inicio de sesión	Alta	1	Hecho
1		Diseñar interfaz grafica	Media	½	Hecho
1		Realizar testing unitario al módulo desarrollado	Alta	½	Hecho
1	Cerrar sesión de usuario (HU-003)	Codificar e integrar módulo de cierre de sesión	Media	1	Hecho
1		Diseñar interfaz grafica	Media	1	Hecho
1		Realizar testing unitario al módulo desarrollado	Media	1	Hecho
1	Plantilla de servicio de usuarios (HU-005)	Diseñar esquema y compatibilidades para la plantilla de servicio de usuarios	Alta	1	Hecho
1		Codificar plantilla de servicio de usuarios	Alta	2	Hecho
1		Realizar testing unitario a los componentes generados producto de la utilización de la plantilla desarrollada	Alta	1	Hecho

Tabla 39: Sprint Backlog. Fuente: Elaboración Propia.

Estructura de Datos

El proyecto posee cuatro estructuras de datos en las cuales soporta sus procesos e interacciones, brindándole al sistema desde una correcta persistencia de información, hasta una notable mejoría en cuanto eficiencia general del sistema y la posibilidad de enviar notificaciones de forma asíncrona a usuarios dentro de la plataforma.

A continuación, se presenta el diagrama DER de la base de datos relacional PostgreSQL, la cual cumple con la administración y gestión de usuarios y sus correspondientes identificadores, incluyendo los provistos por el servidor de identificación OAUTH2.

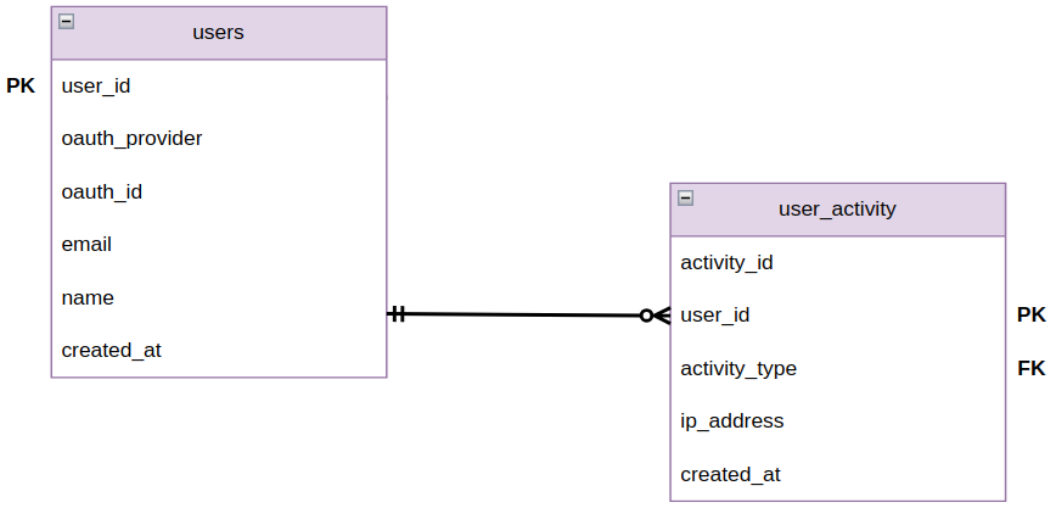

Figura 5: DER base de datos relacional para gestión de usuarios. Fuente: Elaboración Propia.

A continuación, se presentan los diccionarios de datos de la base de datos relacional recién presentada:

Tabla: users			
Campo	Longitud	Tipo de Dato	Descripción
user_id	19	Numérico	Identificador de usuario
oauth_provider	16	Alfabético	Proveedor OAUTH
oauth_id	19	Numérico	Numero identificador provisto por OAUTH
email	254	Alfabético	Email provisto por proveedor OAUTH
name	254	Alfabético	Nombre provisto por proveedor OAUTH
created_at	19	Timestamp	Momento de creación de la cuenta

Tabla 40: Diccionario tabla users. Fuente: Elaboración Propia.

Tabla: user_activity			
Campo	Longitud	Tipo de Dato	Descripción
activity_id	19	Numérico	Identificador de sesión
user_id	19	Numérico	Identificador de usuario
activity_type	254	Alfabético	Tipo de actividad realizada
ip_address	15	Numérico	Direccion IPv4
created_at	19	Timestamp	Momento de realización de la actividad

Tabla 41: Diccionario tabla user_activity. Fuente: Elaboración Propia.

Durante el diseño del sistema, se detectó que incluir un almacenamiento temporal del tipo “caché” podría brindar una reducción temporal en la respuesta del sistema hacia el usuario, aprovechando los procesamientos realizados previamente. Es por esto que fue implementado un almacenamiento temporal con REDIS, el cual busca reducir las llamadas al API de Spring Initializr, analizando las llamadas previas en un tiempo cercano en busca a una igual a la que se desea realizar.

El siguiente diagrama ejemplifica el proceso mencionado:

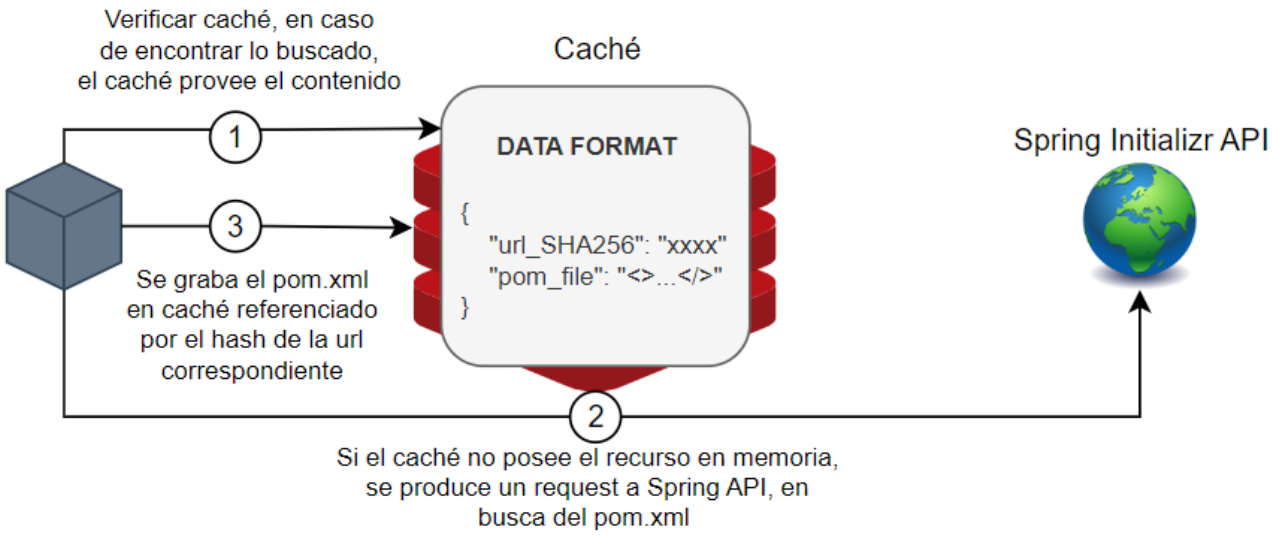

Figura 6: Explicación de funcionalidad de sistema cache. Fuente: Elaboración Propia.

De esta forma, en muchos casos, se logra ahorrar el valioso tiempo consumido por una llamada hacia una API externa.

La simple estructura de datos del caché mencionado se expone a continuación:

init_request

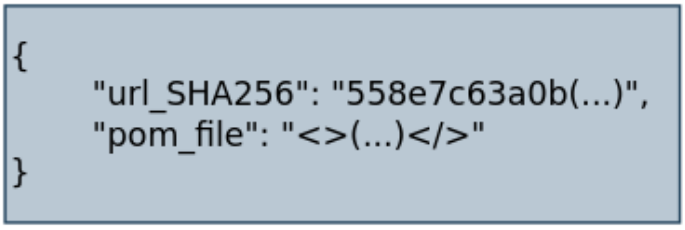

Figura 7: Estructura de datos de caché para la generación de poms. Fuente: Elaboración Propia.

A continuación, se presenta el diccionario de la estructura de datos presentada:

Campo	Longitud	Tipo de Dato	Descripción
url_SHA256	64	Alfabético	Hash SHA-256 de la url utilizada para la petición hacia la API de Spring Initializr
pom_file	65535(max)	Alfabético	Contenido del archivo pom.xml resultante del request

Tabla 42: Diccionario de datos presentes en caché. Fuente: Elaboración Propia.

También se definió una estructura de datos encargada de almacenar las arquitecturas diseñadas por los usuarios en la plataforma y su código correspondiente generado de forma automática. De esta forma, mediante MinIO, el sistema tiene una plataforma donde trabajar dinámicamente con el código, permitiendo a los usuarios pueden descargar un archivo .zip con la arquitectura diseñada. La estructura de datos mencionada cumple con el siguiente formato:

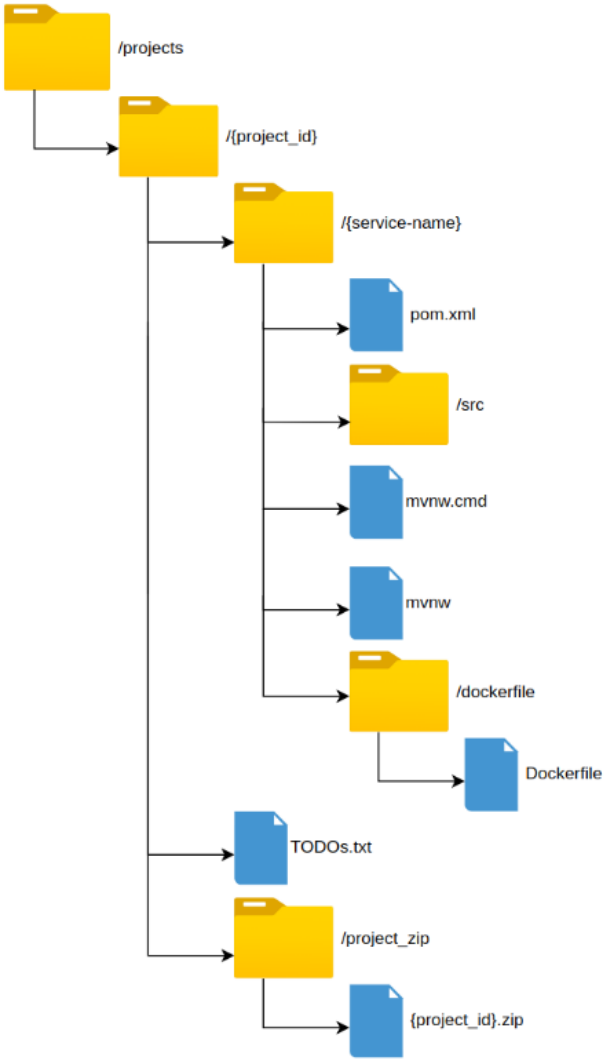

Figura 8: Estructura de archivos para almacenar y generar código. Fuente: Elaboración Propia.

.

Además, el sistema requirió para un correcto funcionamiento en procesos asíncronos la inclusión de un sistema de mensajería, en este caso se optó por utilizar Apache Kafka. La herramienta mencionada cumple con dos tareas cruciales para el sistema:

- Informar de la presencia de una nueva arquitectura a generar y brindar su especificación.
- Mantener actualizado al usuario sobre el proceso de generación del código que solicitó, informándole en qué estado se encuentra la generación del producto final.

- Los tópicos utilizados para las funciones mencionadas son **service-generation-request** y **service-generation-status** respectivamente, los cuales interaccionan con los servicios del sistema de la siguiente forma:

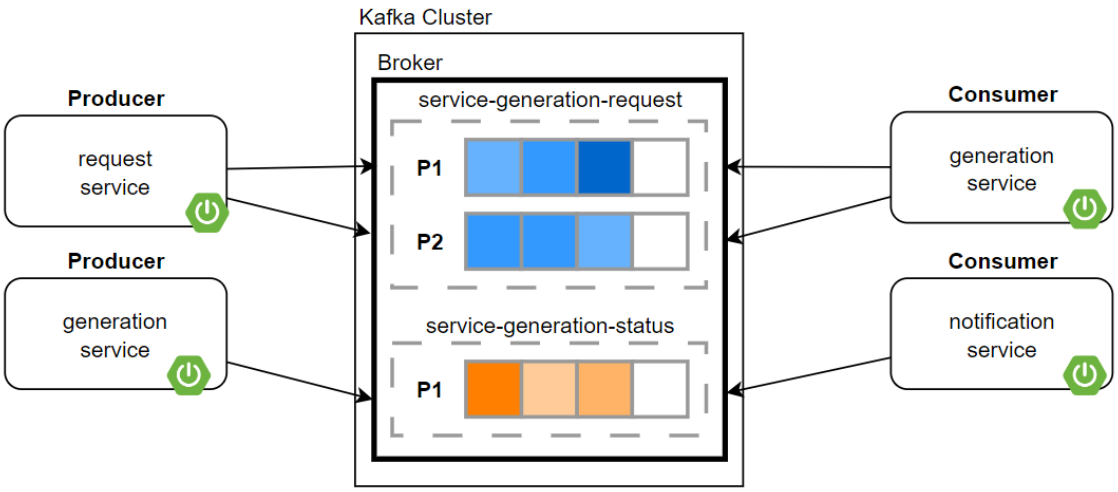

Figura 9: Diagrama de explicación del sistema asíncrono de mensajes. Fuente: Elaboración Propia.

Como se puede apreciar en el diagrama presentado, el servicio está compuesto por un único bróker y service-generation-request posee dos particiones a comparación de service-generation-status, que solo posee una. Esta decisión está basada en la posibilidad de procesamiento paralelo en las solicitudes generadas por los usuarios que brinda la presencia de más de una partición, algo no tan necesario en el caso de los mensajes de progreso de la generación de código.

Se procede a exponer el modelo de datos de cada tópico mencionado.

service-generation-status:

Mediante la informacion contenida por el topico expuesto, el sistema es capaz de mantener al usuario informado en materia del progreso transcurrido referido a la generacion de codigo del sistema que diseñó en el canvas.

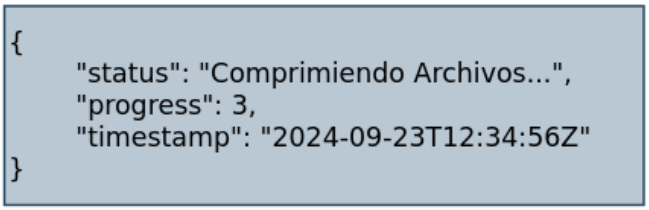

Figura 10: Estructura de datos del tópico ‘service-generation-status’. Fuente: Elaboración Propia.

A continuación, se presenta el diccionario de la estructura de datos presentada:

Campo	Longitud	Tipo de Dato	Descripción
status	32	Alfabético	Mensaje informativo del proceso de generación dinámica de código
progress	1	Numérico	Rango de números del 1 al 3, indicando en que paso del proceso se encuentra la generación dinámica de código
timestamp	19	Alfabético	Momento de actualización de estado de la actividad

Tabla 43: Diccionario de datos de tópico ‘service-generation-status’. Fuente: Elaboración Propia.

service-generation-request:

La información transportada por el presente tópico definido es crítica para el funcionamiento del sistema, ya que en base a él JSON ejemplificado a continuación, el sistema recibe de forma asíncrona las nuevas tareas a procesar y construye el código necesario solicitado por el usuario. Es importante aclarar que el JSON posee muchos atributos que para el prototipo podrían parecer innecesarios ya que no permiten personalización, pero de esta forma el sistema se encuentra listo para ampliar las personalizaciones en un hipotético caso de un desarrollo completo.

Figura 11: Estructura de datos del tópico ‘service-generation-request’. Fuente: Elaboración Propia.

A continuación, se presenta el diccionario de la estructura de datos presentada:

Campo	Tipo de Dato	Descripción
projectName	Alfabético	Nombre del proyecto general
versión	Numérico	Versión del proyecto, 1.0 por defecto
services	Alfabético	Contiene y define los componentes de la arquitectura
name	Alfabético	Nombre del componente
type	Alfabético	Tipo de componente, es decir de que trata el servicio
versión	Numérico	Versión de la plantilla utilizada
description	Alfabético	Descripción del componente
endpoints	Alfabético	Sección referida a los endpoints del componente
path	Alfabético	Url base del componente
dependencies	Alfabético	Dependencias del componente
connections	Alfabético	Conexiones entre componentes especificadas en el canvas
source	Alfabético	Punto A de la conexión
target	Alfabético	Punto B de la conexión
protocol	Alfabético	Protocolo de la comunicación
type	Alfabético	Metodología utilizada para la comunicación
databases	Alfabético	Contiene los elementos de persistencia de los servicios
name	Alfabético	Nombre del elemento de persistencia
owner	Alfabético	Servicio que interactúa con el elemento de persistencia
type	Alfabético	Herramienta utilizada
version	Numérico	Versión de la herramienta utilizada
infrastructure	Alfabético	Contiene los elementos de configuración e infraestructura
name	Alfabético	Nombre del componente de infraestructura
type	Alfabético	Tipo de componente de infraestructura
port	Numérico	Puerto del componente de infraestructura
security	Alfabético	Contiene los elementos de seguridad global del sistema
auth	Alfabético	Especifica como se realiza la autenticación en el sistema
type	Alfabético	Metodología de autenticación
configurations	Alfabético	Contiene información para la personalización de seguridad
secretKey	Alfabético	Clave secreta utilizada para asegurar el sistema

Tabla 44: Diccionario de tópico ‘service-generation-request’. Fuente: Elaboración Propia.

Prototipos de Interfaces de Pantallas

Al acceder a la plataforma web, se solicita de forma obligatoria ingresar por medio de la cuenta de Google perteneciente al usuario.

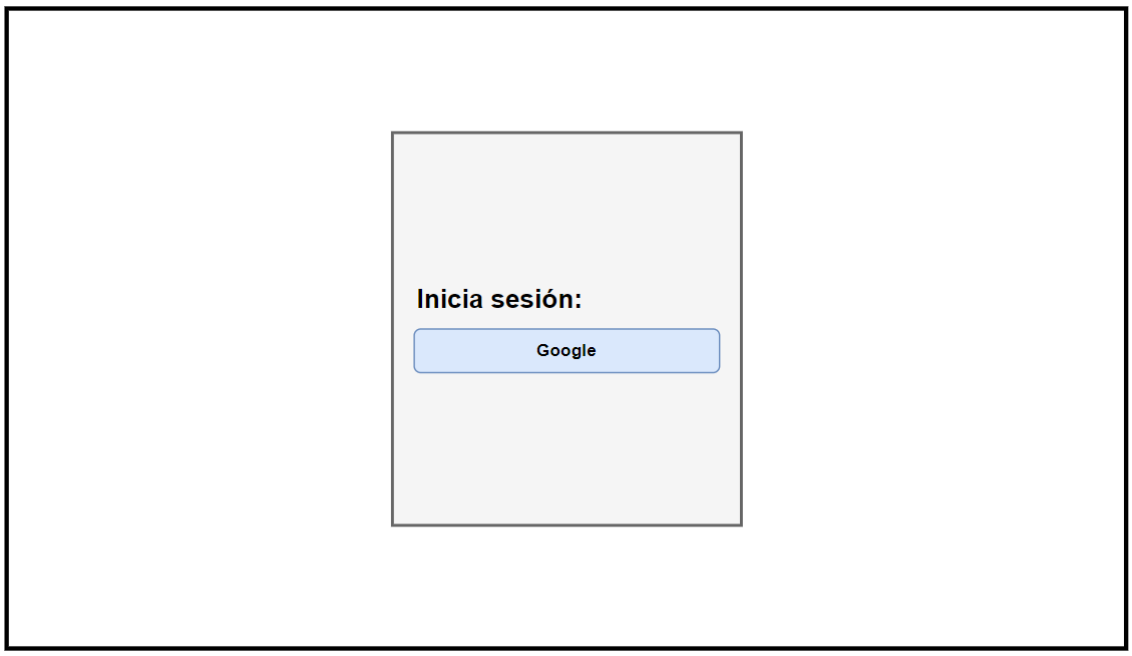

Figura 12: Pantalla de inicio de sesión. Fuente: Elaboración Propia.

Una vez es validado y/o registrado el usuario en el sistema, el mismo es enviado a la página principal de la plataforma, donde se encuentra su funcionalidad *core*, referida al diseño de sistemas distribuidos.

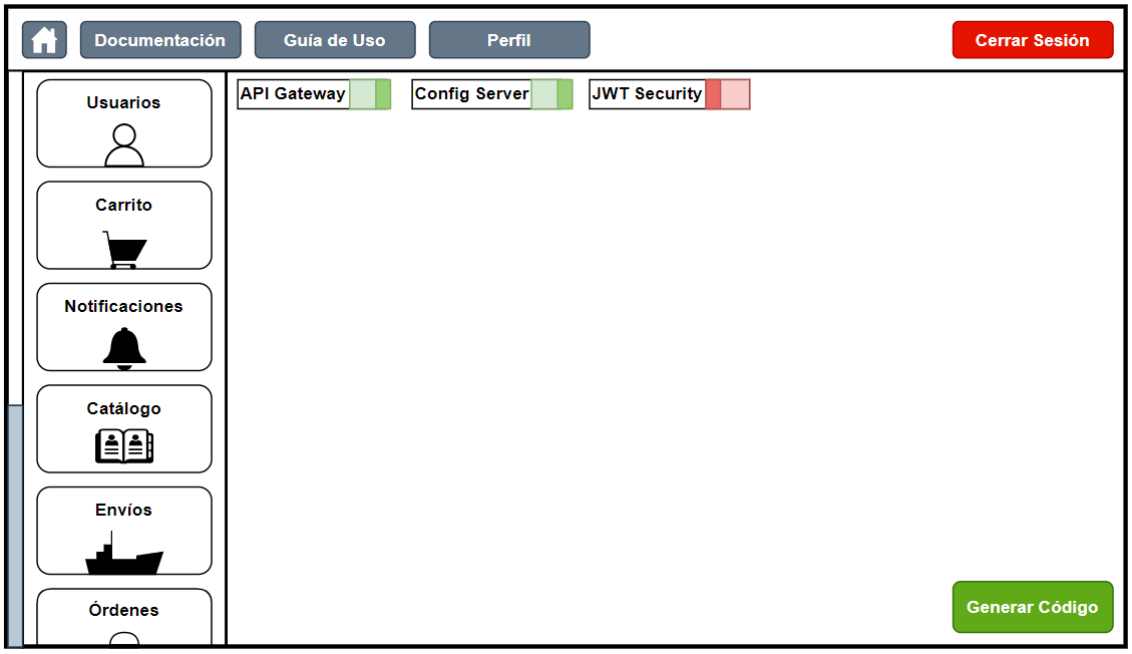

Figura 13: Pantalla de inicio. Fuente: Elaboración Propia.

Dentro de la pantalla principal, el usuario es capaz de arrastrar los componentes participes de la arquitectura al canvas, para comenzar a diseñar el sistema.

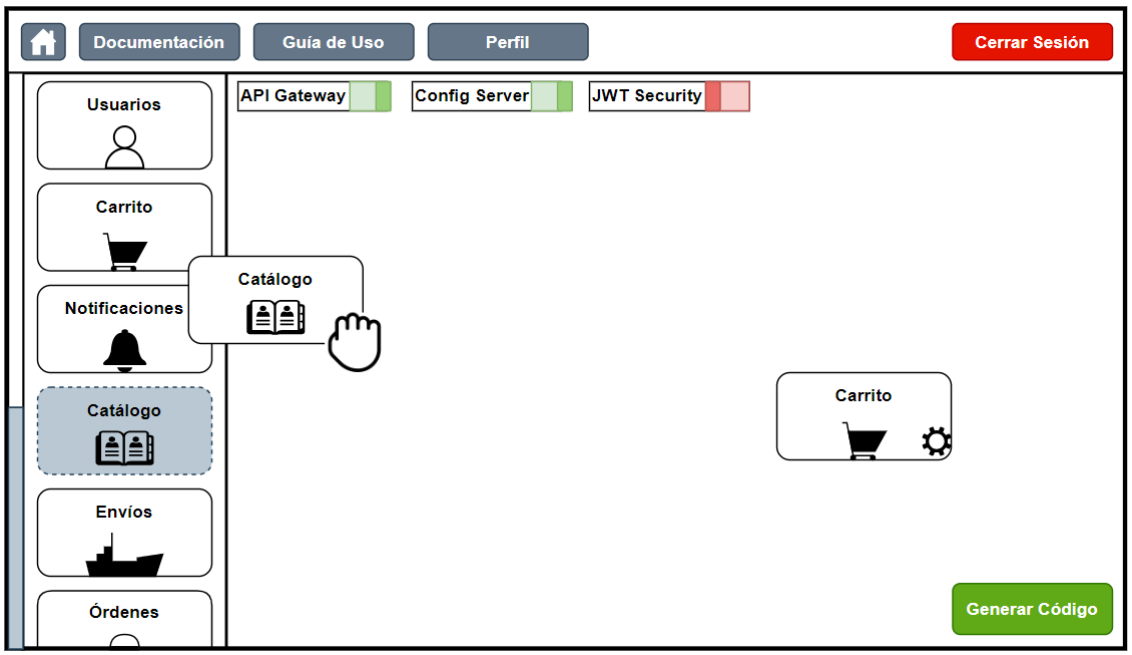

Figura 14: Funcionalidad Drag-and-Drop. Fuente: Elaboración Propia.

Los componentes depositados sobre el canvas a su vez, si su compatibilidad lo permite, se pueden unir entre sí, para generar funcionalidades más complejas mediante la interacción de los mismos.

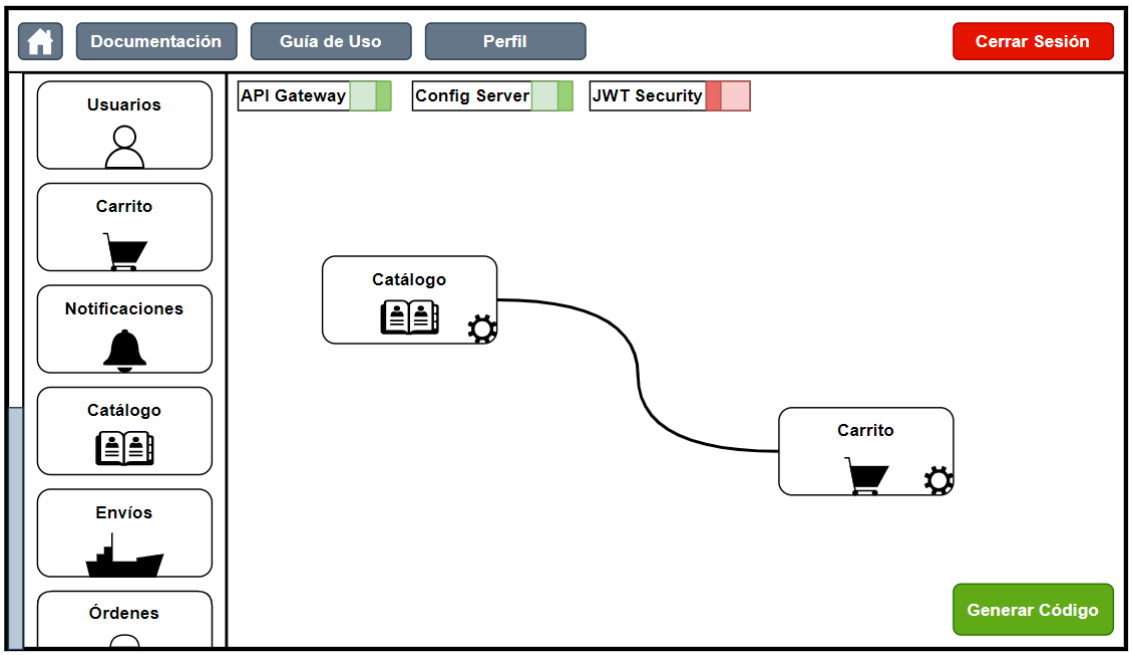

Figura 15: Funcionalidad de unión de componentes. Fuente: Elaboración Propia.

Para permitir la personalización de cada componente depositado en el canvas, el usuario debe presionar el engranaje correspondiente para acceder al menú de configuración.

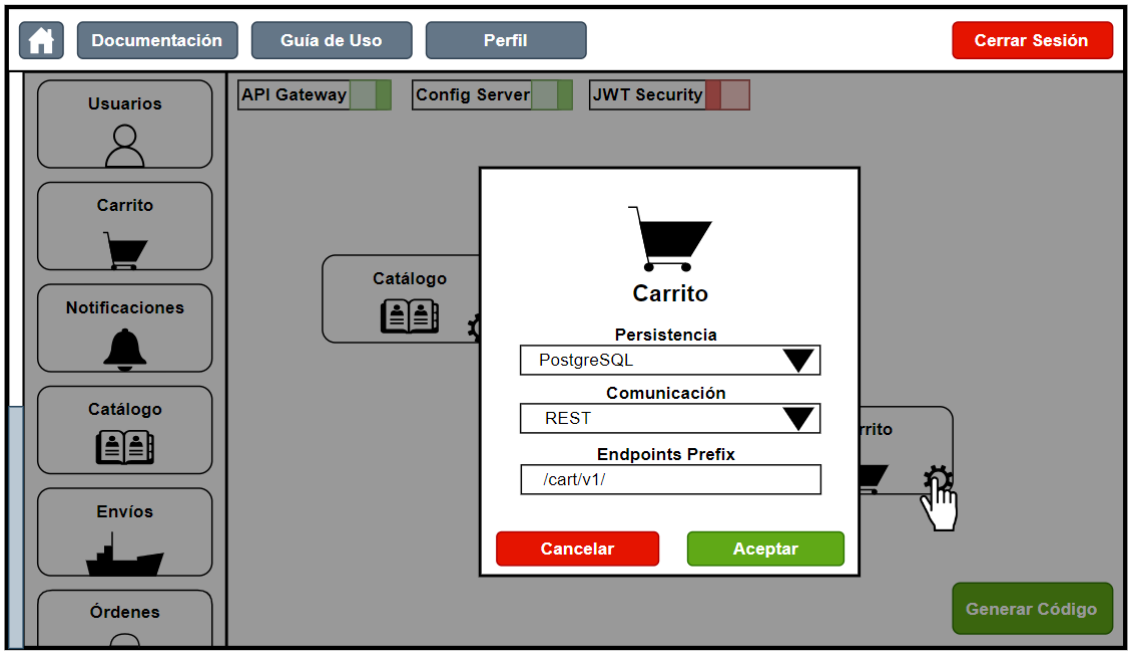

Figura 16: Pantalla de configuración de componente. Fuente: Elaboración Propia.

Una vez que el usuario diseñó y configuró a su gusto una arquitectura especifica, este puede generar el código correspondiente presionando el botón verde situado en la esquina inferior derecha del canvas, donde recibirá en tiempo real el progreso que lleva el proceso de generación dinámica de código.

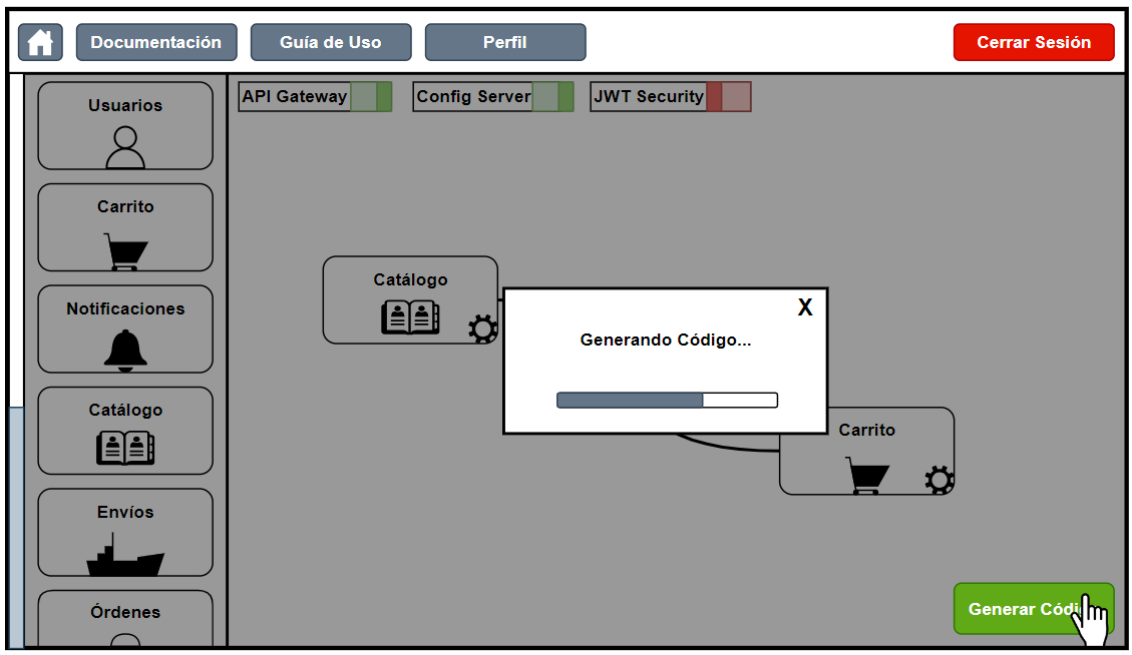

Figura 17: Pantalla de proceso de generación de código. Fuente: Elaboración Propia.

Para que el usuario conozca los diferentes componentes utilizables en el canvas, su composición, funcionamiento, metodología de persistencia y compatibilidades, puede acceder a la sección de documentación, donde se encontrará con la información necesaria para comprender cada pieza disponible.

Figura 18: Pantalla de documentación. Fuente: Elaboración Propia.

Si el usuario desea comprender el funcionamiento de la plataforma en sí, existe un apartado denominado Guía de Uso, donde puede acceder a la información mencionada.

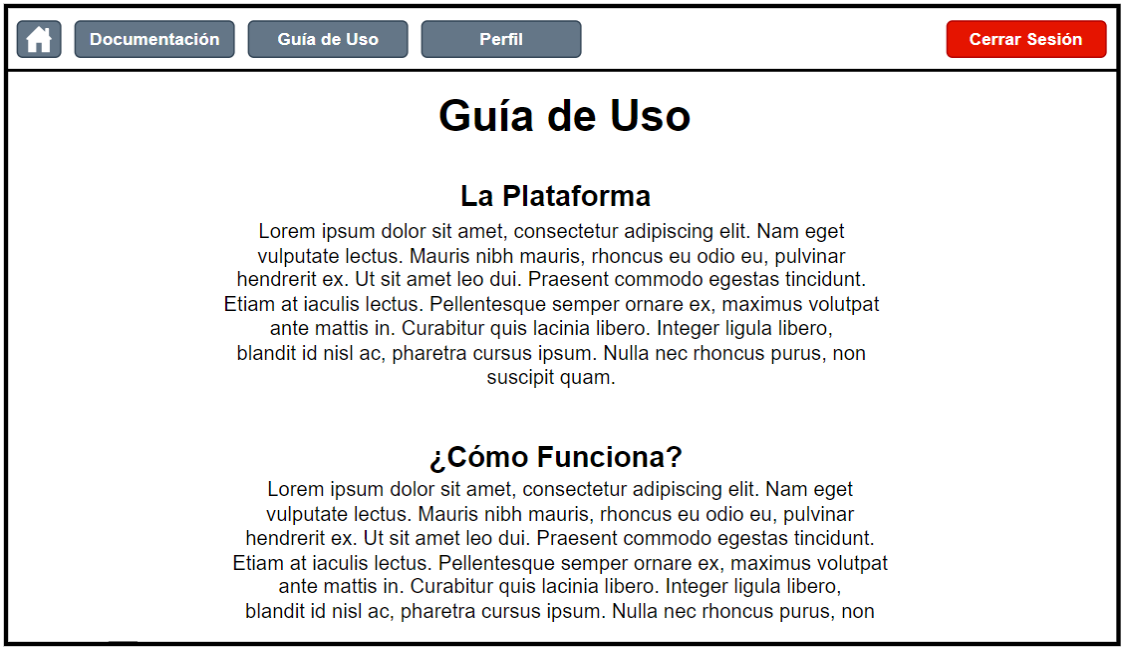

Figura 19: Pantalla de guía de uso. Fuente: Elaboración Propia.

Cuando el usuario del sistema genera arquitecturas en el canvas, las ultimas 5 son registradas y accesibles para su descarga en el apartado Perfil, donde se expone información básica de utilidad sobre las arquitecturas mencionadas y un enlace para su correspondiente descarga.

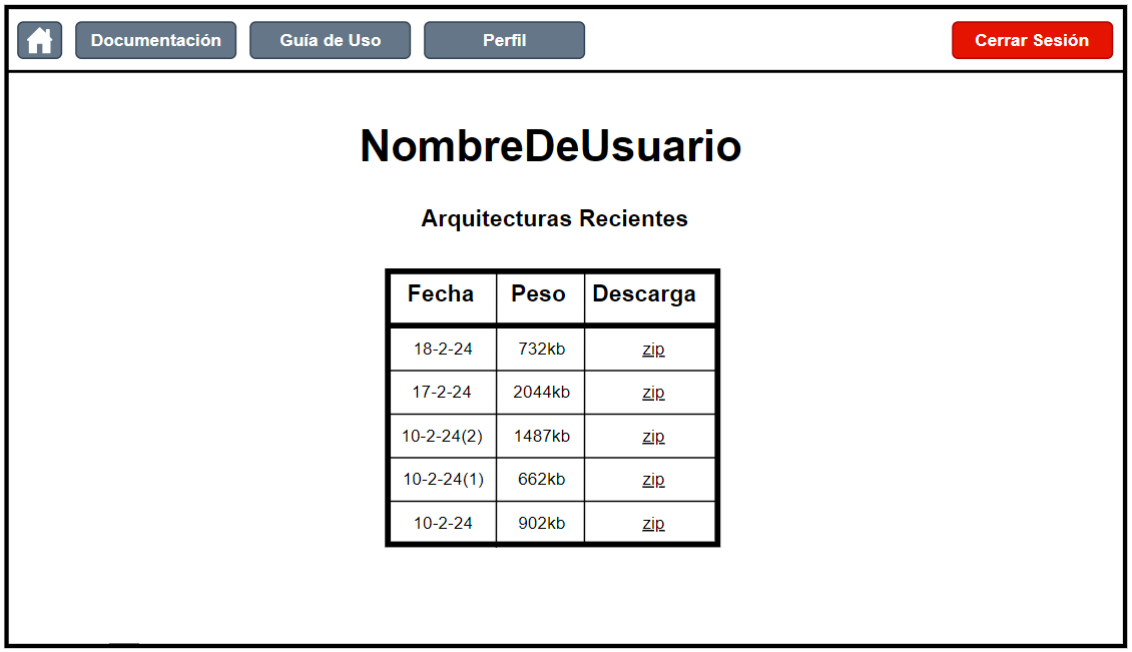

Figura 20: Pantalla de perfil de usuario. Fuente: Elaboración Propia.

Finalmente, para concluir su actividad en la plataforma, el usuario puede cerrar su sesión mediante el botón presente en la esquina superior derecha, luego de confirmar definitivamente su deseo de realizar tal acción.

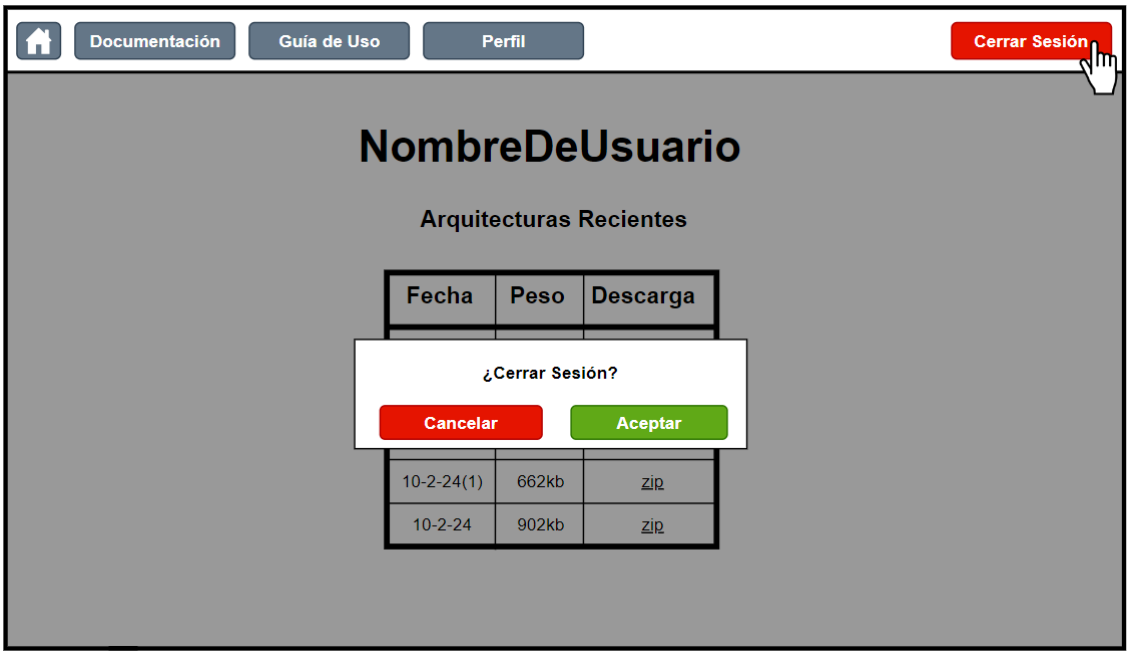

Figura 21: Pantalla de cierre de sesión. Fuente: Elaboración Propia.

Diagrama de Arquitectura

A continuación, se presenta el diagrama de arquitectura correspondiente al proyecto tratado en el presente documento. Debido a las decisiones tomadas apreciables en el diagrama, se logró un sistema altamente escalable y eficiente para el fin que persigue.

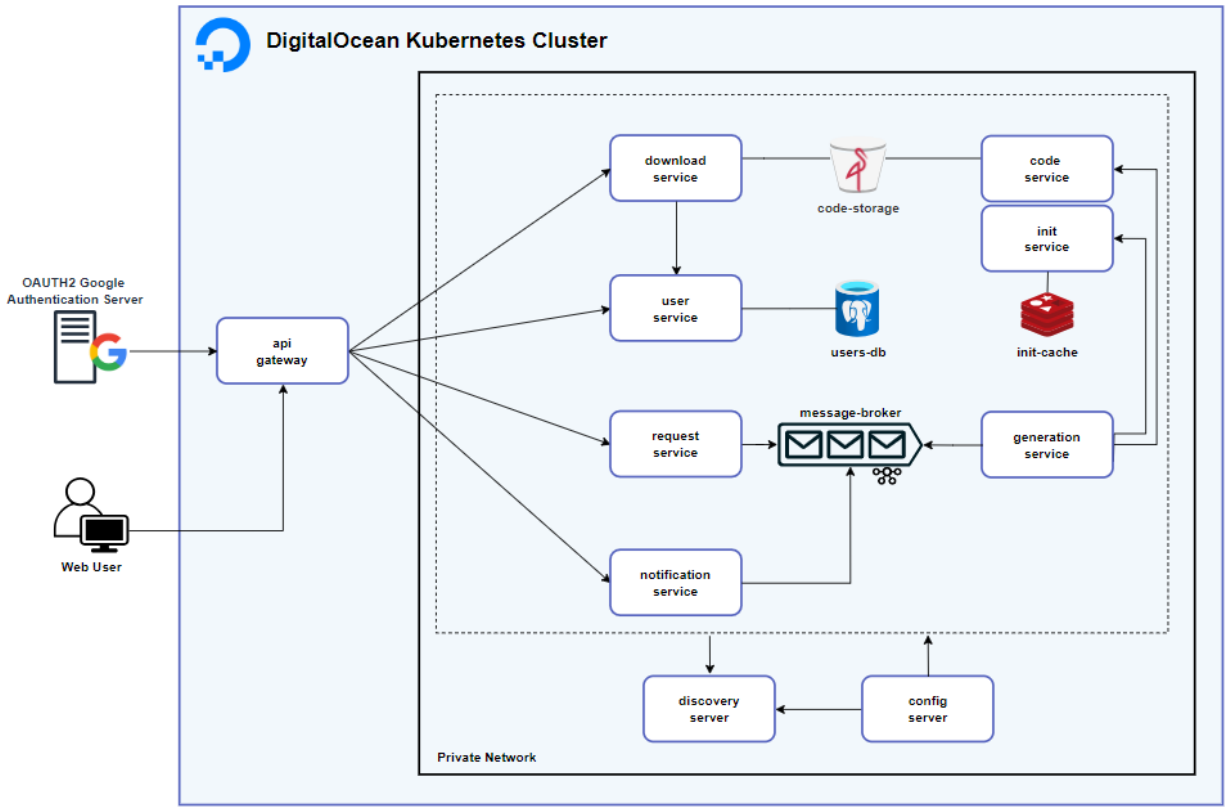

Figura 22: Diagrama de arquitectura. Fuente: Elaboración Propia.

Seguridad

Acceso a la Aplicación

Como fue expresado en reiteradas ocasiones a lo largo del presente documento, el proyecto gestiona a los usuarios mediante OAuth2, específicamente mediante el inicio de sesión vía Google, siendo una alternativa muy conveniente para una aplicación como la desarrollada, debido a su permiso de acceso al público general.

Si bien por motivos de respaldo, auditoria y gestión de recursos el propio proyecto almacena la información de los usuarios registrados en la plataforma, las contraseñas son manejadas por Google, debido a que es mediante un usuario de la plataforma mencionada que el sistema concede el acceso al proyecto. Por lo tanto, los usuarios se deben adecuar a los requerimientos de Google para la creación de sus credenciales. A continuación, se listan los requerimientos al día de la fecha dispuestos por Google para la creación de un email bajo su dominio y su respectiva clave de acceso.

Requerimientos para la creación de un email:

- Debe seguir el formato estándar de una dirección de correo electrónico, es decir nombre@dominio.com. En este caso, el dominio es @gmail.com.
- El nombre de usuario, es decir el texto previo al '@', debe contener entre 6 y 30 caracteres.
- Únicamente se permiten caracteres del tipo a-z y A-Z, además de números 0-9 y los caracteres especiales como el punto (.), el guion bajo (_) y el medio (-).
- La dirección de correo no debe estar en uso en Gmail. Es decir que cada dirección de correo debe ser única.

Requerimientos para la especificación de una contraseña:

- La contraseña debe contener más de 8 caracteres.
- Si bien no existe una limitación relacionada, se recomienda la utilización de minúsculas, mayúsculas, números y caracteres especiales.
- No debe formar parte de la lista de contraseñas consideradas comunes por Google.

A su vez, Google almacena las contraseñas de los usuarios utilizando funciones de hash que convierten la contraseña en una cadena de texto irreversible, además de añadir un valor aleatorio denominado 'salt' para dificultar los intentos de ataque por fuerza bruta. También cabe destacar la existencia de la posibilidad para los usuarios de Google de activar la autenticación de dos factores, conocida como 2FA, añadiendo una capa extra de seguridad al requerir un segundo elemento para acceder a la cuenta de usuario.

Respecto a los roles de usuario expuestos a continuación, debido a la naturaleza del proyecto y el desarrollo únicamente de las funcionalidades core para el prototipo, el usuario administrador no tiene ninguna ventaja o capacidad diferencial. Aun así, fue añadido de forma "lógica" en el sistema, con el fin de disponer del mismo pensando en futuras actualizaciones que provean características que requieran de un control extra dentro de la aplicación.

Perfiles presentes en la aplicación:

- **Usuario común:** Tiene acceso a la totalidad de funcionalidades presentes en el prototipado tecnológico, es decir, la especificación, configuración y descarga de servicios referenciados a su perfil de usuario y la capacidad de consulta tanto a documentación presente en la plataforma, como a la guía de uso.
- **Usuario administrador:** Como fue explicado de forma previa, actualmente el usuario administrador no posee permisos diferenciales respecto al usuario común, aun así, se encuentra presente en el sistema teniendo en cuenta posibles actualizaciones.

Política de Respaldo de Información

Con el fin de respaldar la información referida tanto al código de la aplicación, como a la información producida producto de su uso y ejecución, se almacenan 2 copias del código fuente de la aplicación y 3 copias de los datos de usuario.

Los datos de usuario, en primera instancia son almacenados en el propio hosting donde se almacena el volumen del contenedor referido al motor de base de datos. En este caso se trata del proveedor de servicios cloud DigitalOcean. Como segunda instancia de back-up, el sistema ejecuta un proceso cada día a las 00:00 (GMT-3), con el fin de persistir una copia del contenido de la base de datos en el servidor local de las oficinas de desarrollo. Finalmente, y con el objetivo de maximizar la integridad de los datos de usuario, se persiste de forma semanal una copia de los datos de usuario en un disco duro externo, siendo este almacenado en una ubicación confidencial, en un edificio diferente al servidor local, y de conocimiento únicamente por parte de los cargos gerenciales de la empresa.

En cuanto al código fuente de la aplicación, este es tratado y persistido en Github, donde los desarrolladores trabajan en él. A su vez, como se mencionó anteriormente, las versiones funcionales son respaldadas y persistidas al momento de su completitud en dos instancias. En primera instancia el código es almacenado de forma manual en el servidor local de la empresa, para finalmente, en segunda instancia, ser almacenado en un disco duro externo reservado en una ubicación confidencial, en un edificio diferente al que se

encuentran las oficinas de la empresa, y de conocimiento únicamente por los cargos gerenciales de la misma.

El servidor local mencionado tanto en el respaldo de código fuente como de datos de usuario se trata de un NAS presente en las oficinas de desarrollo, configurado con RAID 10, para obtener un alto nivel de redundancia además de un notable rendimiento, asegurando una destacable disponibilidad de la información incluso en caso de incidentes, brindando una veloz recuperación del contenido persistido.

Por parte de DigitalOcean la disponibilidad también es tenida en cuenta y, de hecho, es uno de los factores que incidieron en que el sistema sea ejecutado sobre sus servicios en la nube, ya que, en los productos utilizados para el despliegue y ejecución del proyecto desarrollado, la plataforma mencionada presume de un tiempo de actividad del 99,99% (DigitalOcean, 2024).

Análisis de Costos

A continuación, se presentan desglosados, los costos estimados del proyecto en referencia a los recursos humanos necesarios para el desarrollo del sistema informático. Los mismos fueron obtenidos de la web del Consejo Profesional de Ciencias Informáticas de la provincia de Córdoba el día 21 de octubre del año 2024 (CPCIPC, 2024).

Rol	Honorarios	Meses	Subtotal (AR$)
Analista Programador Senior	$1.697.430,72	3	$5.092.292,16
Desarrollador Backend	$1.985.445,37	3	$5.956.335,93
Desarrollador Frontend	$1.883.828,08	2	$3.767.656.16
Analista Testing de Aplicaciones	$1.646.481,38	2	$3.292.962,76
		Total Desarrollo	$18.109.247,01

Tabla 45: Análisis de costos RRHH. Fuente: Elaboración Propia.

Ya presentados los valores referidos a la mano de obra, a continuación, se presentan los costos operativos considerados necesarios para el correcto despliegue y funcionamiento del proyecto.

Recurso	Cant.	Fuente	Subtotal AR$	Mensual AR$
* Kubernetes Basic Node (DigitalOcean) - 8gb RAM - 4 vCPU - 160GB storage	2	https://www.digitalocean.com/pricing	-	$94,464
Nombre de dominio .com	1	https://www.hostinger.com.ar/dominios	-	$2.144
NAS Drive - S.O Linux - Compatible RAID 10 - Capacidad 4 HDD	1	https://www.compel.com.ar/almacenamiento/almacenamiento/nas-drive-au-4b-25-35a335696.html	$816.851	-
HDD 3T NAS	4	https://www.compel.com.ar/almacenamiento/hdd-internos/hdd-3t-sea-35-nas-ironwol-328858.html	$636.660	-
		Total Costo Inicial	$1.453.511	
			Total Costos Fijos	$96.608

Tabla 46: Análisis de costos operativos. Fuente: Elaboración Propia.

* Valor original en USD, convertido a AR$ considerando 1 USD equivalente a 948 AR$ en base a la cotización brindada por el Banco Central de la República Argentina el día 21 de octubre del año 2024 (BCRA, 2024).

En cuanto a los costos referidos al software utilizado para el desarrollo del proyecto, se tomó la decisión de recurrir a plataformas de código abierto, accediendo a planes gratuitos, para ahorrar costos en cuanto a licenciamiento. Aun así, se exponen estas herramientas a modo informativo.

Herramienta	Subtotal (AR$)
PostgreSQL	$0
Apache Kafka	$0
MinIO	$0
Docker	$0
Kubernetes	$0
Spring Framework	$0
Total Licencias de Software	$0

Tabla 47: Análisis de costos herramientas de desarrollo. Fuente: Elaboración Propia.

A modo de conclusión del análisis de costos, se expone el resumen de los mismos excluyendo los valores de los salarios previamente detallados.

	Capital Humano	Software y Licencias	Infraestructura y Hardware	Total
Costo Inicial (AR$)*	$18.109.247,01	$0	$1.453.511	19.562.758,01
Costo Fijo Mensual (AR$)**	$0	$0	$96.608	$96.608

Tabla 48: Resumen de costos excluyendo RHHH. Fuente: Elaboración Propia.

* Comprende todo costo relacionado con los primeros tres meses de actividad, es decir, hasta concluir el desarrollo del sistema, excluyendo el despliegue del mismo.

** Comprende los costos de mantener el sistema, una vez desarrollado, desplegado en la nube.

Análisis de Riesgos

A continuación, se presentan descriptos y detallados los riesgos que se pueden presentar durante el transcurso del desarrollo del proyecto, dividiendo los mismos en diferentes tablas según su razón de origen. En ellas se presenta tanto la probabilidad como el impacto, valores que basaran su significado en la matriz que se expone luego.

Riesgos Técnicos

ID	Tipo	Riesgo	Probabilidad	Impacto	Causa
1	Técnico	El tiempo de respuesta de la plataforma no cumple con los estándares de respuesta esperados	0,40	4	Ineficiencias en la arquitectura o falta de optimización en las comunicaciones
2	Técnico	Vulnerabilidad de seguridad en servicios que interactúan con bases de datos.	0,70	2	Falta de sanitización y validación adecuada de las entradas de usuario

Tabla 49: Riesgos Técnicos. Fuente: Elaboración Propia.

Riesgos del Proyecto

ID	Tipo	Riesgo	Probabilidad	Impacto	Causa

3	Proyecto	Dependencia de API's de terceros para integración o creación de servicios que no estén disponibles	0,80	3	Falta de control sobre la calidad y disponibilidad de servicios utilizados en el sistema
4	Proyecto	Dificultad para conseguir el personal técnico necesario para llevar a cabo el desarrollo del sistema	0,35	3	Temática muy específica debido al objetivo final del sistema, enfocado en sistemas distribuidos
5	Proyecto	Recursos insuficientes para el desarrollo del proyecto	0,45	4	Falta de inversión en recursos humanos y/o tecnológicos
6	Proyecto	El campo de investigación no brinda información útil sobre la cual partir	0,60	2	El campo de investigación de microservicios puede ser incompleto debido a su reciente inclusión

Tabla 50: Riesgos del Proyecto. Fuente: Elaboración Propia.

Una vez expuestos los riesgos identificados del proyecto, se procede con la siguiente matriz de riesgo previamente mencionada, con el fin de ponderar las probabilidades de ocurrencia y sus impactos relacionados.

			IMPACTO				
			Insignificante 1	Menor 2	Significativo 3	Mayor 4	Severo 5
PROBABILIDAD	Casi Seguro	**0,9**	0,9	1,8	2,7	3,6	4,5
	Probable	**0,7**	0,7	1,4	2,1	2,8	3,5
	Moderado	**0,5**	0,5	1	1,5	2	2,5
	Poco Probable	**0,3**	0,3	0,6	0,9	1,2	1,5
	Raro	**0,1**	0,1	0,2	0,3	0,4	0,5

Figura 23: Matriz de Riesgos. Fuente: Elaboración Propia.

En base a la matriz presentada, se desarrollaron tanto las tablas previamente expuestas como la definida a continuación, referida al análisis cuantitativo de los riesgos.

Riesgo	Probabilidad de Ocurrencia	Impacto	Grado de Exposición	Porcentaje	Porcentaje Acumulado
Dependencia de API's de terceros para integración o creación de servicios que no estén disponibles	0,80	3	2,40	%25,40	%25,40
Recursos insuficientes para el desarrollo del proyecto	0,45	4	1,80	%19,06	%44.46
El tiempo de respuesta de la plataforma no cumple con los estándares de respuesta esperados	0,40	4	1,60	%16,93	%61.39
Vulnerabilidad de seguridad en servicios que interactúan con bases de datos.	0,70	2	1,40	%14,81	%76,20
El campo de investigación no brinda información útil sobre la cual partir	0,60	2	1,20	%12,70	%88,90
Dificultad para conseguir el personal técnico necesario para llevar a cabo el desarrollo del sistema	0,35	3	1,05	%11,10	%100

Tabla 51: Análisis cuantitativo de riesgos. Fuente: Elaboración Propia.

Una vez presentado el grado de exposición al riesgo correspondiente para cada riesgo detectado, es momento de utilizar el Principio de Pareto para centrar la atención en los aspectos importantes y críticos, ignorando los más triviales. A continuación, se expone el grafico correspondiente.

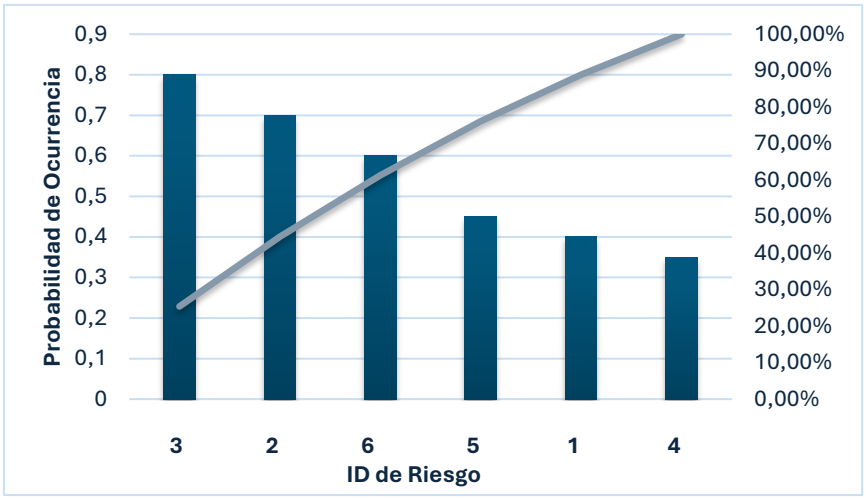

Figura 24: Principio de Pareto de la Exposición al Riesgo. Fuente: Elaboración Propia.

Una vez identificados los riesgos más amenazantes utilizando el Principio de Pareto, se elaboraron planes de contingencia específicos para mitigar dichas amenazas. A continuación, se presentan los detalles de estos planes.

Riesgo	Plan de Contingencia
Dependencia de API's de terceros para integración o creación de servicios que no estén disponibles	Desarrollar soluciones internas mínimas viables para reducir la dependencia crítica de API's externas. Establecer contratos con proveedores clave para asegurar tiempos de respuesta y disponibilidad, y definir mecanismos de monitorización para detectar fallos y actuar rápidamente.
Recursos insuficientes para el desarrollo del proyecto	Establecer desde el inicio un plan de asignación flexible de recursos, que permita redistribuir esfuerzos en función de las prioridades del proyecto. Proponer fases adicionales o ajustes en el alcance del proyecto para adaptarse al presupuesto disponible.
El tiempo de respuesta de la plataforma no cumple con los estándares de respuesta esperados	Realizar una revisión exhaustiva de la arquitectura y aplicar optimizaciones puntuales en las áreas críticas. Integrar soluciones de monitoreo en tiempo real para detectar problemas de rendimiento y ajustar la capacidad del sistema.
Vulnerabilidad de seguridad en servicios que interactúan con bases de datos.	Implementar capas adicionales de seguridad, como firewalls a nivel de aplicación y encriptación de los datos sensibles. Realizar revisiones periódicas de código centradas en la seguridad y aplicar parches inmediatos ante cualquier vulnerabilidad descubierta.

Tabla 52: Planes de contingencia. Fuente: Elaboración Propia.

Conclusiones

A partir de la identificación de un patrón recurrente en el desarrollo de aplicaciones de microservicios, basado en que se suelen programar servicios con funcionalidades y lógica similares, se ideó, diseñó e implementó un sistema con el objetivo de ofrecer una alternativa que mejore la productividad del desarrollo, aportando múltiples beneficios para el desarrollador y por consecuencia, para el cliente y/o empleador del mismo.

El proyecto resultó en una plataforma web con una interfaz simple e intuitiva, construida con múltiples tecnologías y basando su núcleo lógico en plantillas para la generación dinámica de código. Esta herramienta logró brindar a los usuarios la capacidad de diseñar arquitecturas distribuidas de microservicios basadas en componentes comunes, arrastrando y conectando piezas sobre un canvas interactivo. De manera sencilla, se pueden generar miles de líneas de código, con el respaldo de documentación adecuada. Al descargar el código resultante de su diseño sobre el canvas, el usuario obtiene componentes totalmente compatibles entre sí, con configuraciones correctas, y una comunicación fluida y libre de errores entre los servicios.

A lo largo del desarrollo del sistema y de la elaboración de este documento, tuve la oportunidad de aplicar y profundizar en los conocimientos adquiridos durante mi carrera, aprendiendo de manera práctica en cada etapa del proceso, desde la concepción del proyecto hasta la documentación final. Este trabajo también me permitió enfrentar el verdadero desafío que implica construir un sistema desde cero, combinando diversas tecnologías de forma coherente y funcional. En este proceso, no solo consolidé lo aprendido, sino que también comprendí las complejidades que implican lograr una solución robusta y eficiente, y la satisfacción que genera tangibilizar en un producto final de software lo que, en un comienzo, no fue más que una idea.

Demo

A continuación, se encuentra listado un enlace con destino a Google Drive donde se encuentra el código correspondiente al prototipo desarrollado, con instructivos para la puesta en marcha del sistema en cuestión.

Enlace: https://drive.google.com/drive/folders/1CiOxIPMH1h3d0xEvkFjUOkFmgELMhOA2?usp=sharing

Referencias

Amazon Web Services. (2023). *What is Docker?*. Recuperado de (https://aws.amazon.com/es/docker/)

Amazon Web Services. (2023). *What is Java?*. Recuperado de (https://aws.amazon.com/es/what-is/java/)

Apache Software Fundation. (2020). *What is Velocity?*. Recuperado de (https://velocity.apache.org/)

Apache Software Fundation. (2024). *About Jena*. Recuperado de (https://jena.apache.org/about_jena/about.html)

Apache Software Fundation. (2024). *Apache Kafka*. Recuperado de (https://kafka.apache.org/)

BCRA. (2024). *Cotizaciones por fecha*. Recuperado de (http://www.bcra.gob.ar/PublicacionesEstadisticas/Cotizaciones_por_fecha_2.asp)

Brown, T., & Smith, L. (2023). *The impact of Low-code platforms and intuitive interfaces on software development efficiency*. Journal of Software Innovation, volumen 18, 75-89

Chaudhary, H. A. A. & Ahmed, T. (2021). *Integration of micro-services as components in modeling enviroments for low code development.* ISP RAS, volumen. 33, 19-30. Doi: 10.15514/ISPRAS-2021-33(4)-2

CPCIPC. (2024). Honorarios Recomendados. Recuperado de (https://cpcipc.org.ar/honorarios-recomendados/)

Dhoke, P. & Lokulwar, P. (2023). *Evaluating the Impact of No-Code/Low-Code Backend Services on API Development and Implementation: A Case Study Approach.* 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), 1-5. Doi: 10.1109/ICCCNT56998.2023.10306945

DigitalOcean. (2024). *DigitalOcean Managed Kubernetes.* Recuperado de (https://www.digitalocean.com/products/kubernetes)

Elgheriani, N. S. & Ahmed, N. A. (2022). *Microservices vs. monolithic architectures.* International Journal of Applied Sciences and Technology, volumen 4, 501-514. doi: 10.47832/2717-8234.12.47

GitHub. (2024). *About GitHub and Git*. Recuperado de (https://docs.github.com/es/get-started/start-your-journey/about-github-and-git)

IBM. (2021). *Minio*. Recuperado *de (https://www.ibm.com/docs/es/cloud-private/3.2.x?topic=private-minio)*

IBM. (2024). *PostgreSQL*. Recuperado de (https://www.ibm.com/mx-es/topics/postgresql)

JetBrains. (2024). *IntelliJ IDEA features*. Recuperado de (https://www.jetbrains.com/es-es/idea/features/)

Kubernetes. (2022). *¿Qué es Kubernetes?*. Recuperado de (https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/)

Lewis, J., & Fowler, M. (2014). *Microservices, a definition of this new architectural term.* Recuperado de: https://martinfowler.com/articles/microservices.html

Lopez, B. M., & Garcia, J. L. (2019). *Impacto de arquitecturas de microservicios en el desarrollo web* (Tesis de maestría). Universidad Politécnica de Madrid, Madrid. Recuperada de https://oa.upm.es/55917/1/TESIS_MASTER_BRUNO_MARTIN_LOPEZ.pdf

Misic, B., Novkovic, M., Ramac, R., & Mandic, V. (2017). *Do the microservices improve the agility of software development teams*?. International Scientific Conference on Industrial Systems, volumen 17, 170-175.

Mozilla Developer Network. (2023). *CSS*. Recuperado de (https://developer.mozilla.org/es/docs/Glossary/CSS)

Mozilla Developer Network. (2023). *HTML5*. Recuperado de (https://developer.mozilla.org/es/docs/Glossary/HTML5)

Mozilla Developer Network. (2024). *What is JavaScript?*. Recuperado de (https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/What_is_JavaScript)

Newman, S. (2015). *Building microservices: Designing fine-grained systems*. Newton: O'Reilly Media.

Postman. (2024). *What is Postman?*. Recuperado de (https://www.postman.com/product/what-is-postman/)

Richardson, C. (2019). *Microservices patterns: With examples in Java*. New York: Manning Publications.

Rock Content. (2020). *What is Bootstrap?*. Recuperado de (https://rockcontent.com/es/blog/bootstrap/)

Said, M., Ezzati, A., & Arezki, S. (2023). Microservice-specific language, a step to the low-code platforms. Lecture Notes in Networks and Systems, volumen 637, 817-828. doi: 10.1007/978-3-031-26384-2_72

Schwaber, K., & Sutherland, J. (2010). *Scrum: The art of doing twice the work in half the time*. Houston: Crown Business.

Spring. (2024). *Spring Framework*. Recuperado de https://spring.io/projects/spring-framework

The Thymeleaf Team. (2024). *Thymeleaf*. Recuperado de (https://www.thymeleaf.org/)

Trello. (2023). *Trello Tour*. Recuperado de (https://trello.com/es/tour)

Vincent, P., Lijima, K., Driver, M., Wong, J., & Natis, Y. (2019). *Gartner magic quadrant for enterprise low-code application platforms*. Gartner, Inc. https://www.gartner.com/en/documents/3956079

World Wide Web Consortium. (2014). *RDF 1.1 Concepts and Abstract Syntax*. Recuperado de (https://www.w3.org/TR/rdf11-concepts/)

World Wide Web Consortium. (2013). *SPARQL 1.1 Query Language*. Recuperado de (https://www.w3.org/TR/sparql11-query/)

Anexo

Anexo 1, Estándares de Calidad:

Estándares de Calidad para el Desarrollo de Software

Propósito

El presente documento tiene como objetivo definir los estándares de calidad a seguir durante los desarrollos a realizar en la organización. Los estándares mencionados aseguran que los productos de software cumplan los requisitos obtenidos en conjunto con el cliente, brindando robustes y mantenibilidad.

Introducción

[Explicación del propósito de los estándares desarrollados en el presente documento]

Objetivo

[Justificación del propósito de establecer este tipo de estándares en la organización.]

Alcance

[Definición del alcance de los estándares de calidad expresados a continuación]

Normas y Directrices

[Especificación de las normas y directrices a seguir. Como puede ser ISO/IEC 25010 para la calidad de software]

Roles y Responsabilidades

[Especificación de personas responsables de mantener e implementar los estándares]

Procesos de Control de Calidad

[Detallada descripción de los procesos mediante los cuales se controlará y medirá la calidad del software, como pueden ser revisiones de código, ciertas pruebas, etc]

Métricas y Herramientas

[Indica que métricas y herramientas se van a emplear para medir la calidad del software]

Anexo

Anexo 2, Plan de Pruebas:

Plan de Pruebas

Propósito

El presente documento tiene como objetivo la definición del enfoque, alcance, recursos y cronograma a seguir de las actividades de prueba que se llevaran a cabo en el proyecto [nombre del proyecto]. De esta forma, se busca cumplir con un software desarrollado bajo los requisitos de calidad seguidos y con un correcto funcionamiento.

Introducción

[Descripción específica del propósito del plan de pruebas]

Alcance de Pruebas

[Definición de los componentes, módulos y funcionalidades donde se aplicarán las pruebas]

Tipos de Pruebas

[Especificación y definición de los tipos de prueba a realizar]

Criterios de Aceptación

[Criterios que determinan si el software supero o no las pruebas con éxito]

Planificación de Pruebas

[Detalle de la cronología de las pruebas, especificando a su vez, quienes las realizarán]

Recursos Necesarios

[Listado de recursos, herramientas de software y personal necesario para llevar a cabo las pruebas especificadas.]

Informe de Resultados

[Detallada documentación de los resultados de pruebas, adjuntos a sus correspondientes errores y a la especificación de la gravedad de los mismos]

Anexo

Anexo 3, Relevamiento de Requisitos:

Especificaciones Técnicas - [Nombre del Proyecto]

Propósito

El presente documento tiene como objetivo la descripción minuciosa de los requisitos técnicos y funcionales que el software debe cumplir en el marco del proyecto [Nombre del Proyecto]. Las especificaciones detalladas a continuación, establecen una guía para el diseño, desarrollo, pruebas e implementación del software.

Introducción

[Explicación general del proyecto y sus objetivos en términos tecnológicos]

Requisitos Funcionales

[Listado detallado de las funcionalidades que el sistema debe cumplir]

Requisitos No Funcionales

[Requisitos no relacionados a la funcionalidad directa del software, incluyen los referidos a rendimiento, usabilidad, escalabilidad, plataforma, etc]

Descripción Técnica

[Detalles técnicos del proyecto, donde se incluyen los lenguajes y frameworks a utilizar, las bases de datos, entre otros]

Diagramas

[Exposición de diagramas técnicos representativos de la estructura del software, como pueden ser diagramas de clase, diagramas de flujo de datos, etc]

Criterios de Aceptación Técnica

[Definición de los parámetros técnicos a cumplir para considerar el software correcto, en aspectos referidos a tiempos de respuesta, concurrencia máxima de usuarios bajo un correcto funcionamiento del sistema, etc]

Anexo

Anexo 4, Documentación de Arquitectura:

Documentación de Arquitectura - [Nombre del Proyecto]

Propósito

El presente documento tiene como objetivo la descripción de la arquitectura del sistema para el proyecto [Nombre del Proyecto]. La documentación especificada a continuación proporciona una visión general de la estructura del sistema, sus interacciones entre componentes y las decisiones arquitectónicas clave que guiarán su desarrollo.

Introducción

[Explicación de la arquitectura general del sistema, su propósito y aspectos clave]

Visión General de la Arquitectura

[Descripción del sistema a nivel modular o de capas, en conjunto de una vista de alto nivel]

Componentes Principales

[Especificación y descripción de los componentes del sistema, como APIs, persistencia de datos, Front-end, Back-end, etc]

Interacción entre Componentes

[Detalle de interacción entre los diferentes componentes que conforman el sistema, se pueden utilizar para este propósito diagramas de secuencia o interacción]

Decisiones de Arquitectura

[Explicación de las decisiones tomadas a lo largo del presente documento respecto a la arquitectura]

Requerimientos de Escalabilidad y Seguridad

[Consideraciones referidas a la escalabilidad del sistema y sus medidas de seguridad implementadas en la arquitectura]

Anexo

Anexo 5, Informe de Pruebas:

Informe de Pruebas

Propósito

El presente documento tiene como objetivo la descripción de la arquitectura del sistema para el proyecto [Nombre del Proyecto]. La documentación especificada a continuación proporciona una visión general de la estructura del sistema, sus interacciones entre componentes y las decisiones arquitectónicas clave que guiarán su desarrollo.

Introducción

[Resumen del propósito específico del informe, las pruebas realizadas y los objetivos de cada una]

Alcance de las pruebas

[Especificación de las partes del sistema que fueron objeto de pruebas, adjuntas a sus objetivos]

Resumen de las Pruebas Realizadas

[Descripción de los resultados generales basados en las pruebas llevadas a cabo, además de una breve exposición de los métodos de pruebas utilizados]

Resultados de las Pruebas

[Exposición detallada de los hallazgos clave, incluyendo el número de defectos encontrados, su gravedad aparente y la forma en la que fueron abordados]

Incidencias Detectadas

[Listado de los problemas detectados durante las pruebas, diferenciando entre problemas críticos, mayores y menores]

Conclusiones y Recomendaciones

[Especificación sobre la aptitud del software para ser liberado o sus correcciones requeridas]

Anexo