Universidad Nacional de Loja

Facultad de la Energía, las Industrias y los de Recursos Naturales No Renovables

Maestría en Ingeniería en Software

Migración de una aplicación de escritorio de cobros a un sistema web mediante procesos DevOps. Caso de Aplicación: Empresa MONARKA

Trabajo de Titulación previo a la obtención del título de Magíster en Ingeniería en Software

AUTOR:
Edison Josué Ordoñez Monge

DIRECTOR:
Ing. Edwin René Guamán Quinche, Mg. Sc.

Loja - Ecuador
2025

Certificación

Loja, 22 de enero de 2025

Ing. Edwin René Guamán Quinche Mg. Sc.
DIRECTOR DE TRABAJO DE TITULACIÓN

CERTIFICO:

Que he revisado y orientado todo proceso de la elaboración del Trabajo de Titulación denominado: **Migración de una aplicación de escritorio de cobros a un sistema web mediante procesos DevOps. Caso de Aplicación: Empresa MONARKA**, previo a la obtención del título de **Maestría en Ingeniería en Software,** de autoría del estudiante **Edison Josue Ordoñez Monge**, con cédula de identidad Nro. **1104122922**, una vez que el trabajo cumple con todos los requisitos exigidos por la Universidad Nacional de Loja para el efecto, autorizo la presentación para la respectiva sustentación y defensa.

Firmado electrónicamente por:
EDWIN RENE GUAMAN QUINCHE

Ing. Edwin René Guamán Quinche Mg. Sc.
DIRECTOR DE TRABAJO DE TITULACIÓN

Autoría

Yo, **Edison Josué Ordoñez Monge**, declaro ser autor del Trabajo de Titulación y eximo expresamente a la Universidad Nacional de Loja y a sus representantes jurídicos de posibles reclamos y acciones legales, por el contenido de este. Adicionalmente acepto y autorizo a la Universidad Nacional de Loja la publicación del trabajo de titulación en el Repositorio Digital Institucional - Biblioteca Virtual.

Firma:

Cédula de Identidad: 1104122922

Fecha: 22/01/2025

Correo electrónico: edison.ordonez@unl.edu.ec

Teléfono: 0986480020

Carta de autorización por parte del autor, para la consulta, reproducción parcial y/o total, publicación electrónica de texto completo del Trabajo de Titulación

Yo, **Edison Josué Ordoñez Monge,** declaro ser autor del trabajo de titulación denominado: **Migración de una aplicación de escritorio de cobros a un sistema web mediante procesos DevOps. Caso de aplicación: Empresa MONARKA** como requisito para optar el título de **Magíster en Ingeniería en Software**; autorizo al sistema Bibliotecario de la Universidad Nacional de Loja para que con fines académicos muestre la producción intelectual de la Universidad, a través de la visibilidad de su contenido de la siguiente manera en el Repositorio Institucional.

Los usuarios pueden consultar el contenido de este trabajo en el Repositorio Institucional, en la redes de información del país y del exterior con las cuales tenga convenio la Universidad.

La Universidad Nacional de Loja, no se responsabiliza por el plagio o copia del Trabajo de Titulación que realice un tercero.

Para constancia de esta autorización suscribo, en la ciudad de Loja, a los veinte y dos días del mes de enero de dos mil veinte y cinco.

Firma:

Autor: Edison Josué Ordoñez Monge
Cédula de Identidad: 1104122922
Dirección: Álamos y Castaños, Barrio Pradera
Correo electrónico: edison.ordonez@unl.edu.ec
Teléfono: 0986480020

DATOS COMPLEMENTARIOS

Director del trabajo de titulación: Ing. Edwin René Guamán Quinche Mg. Sc.

Dedicatoria

Dedico este trabajo a mi amada esposa, María Rosa, por ser mi compañera fiel en todas las adversidades y por brindarme su apoyo incondicional. A mis hijas, Martina y Antonia, por su amor y comprensión durante este proceso, y a mis padres, Edison y Bertha, por creer en mis capacidades y enseñarme la importancia de perseverar. A cada uno de ustedes, mi gratitud y amor eterno, pues sin su apoyo, este logro no sería posible.

Edison Josué Ordoñez Monge

Agradecimiento

Quiero expresar mi más sincero agradecimiento a la Universidad Nacional de Loja y a su excelente cuerpo docente, por proporcionarme una formación sólida y por su compromiso con el desarrollo académico. Agradezco especialmente al Ing. Edwin René Guamán Quinche, quien, con su dirección y orientación, fue una pieza fundamental en este proyecto de titulación. También, mi gratitud al Ing. Ruperto Alexander López Lapo, por su asesoramiento y predisposición constante para guiarme en la ejecución de este trabajo. Gracias a todos ustedes por ser parte de este logro.

Edison Josué Ordoñez Monge

Índice de contenidos

Portada i
Certificación ii
Autoría iii
Carta de autorización iv
Dedicatoria v
Agradecimiento vi
Índice de contenidos vii
Índice de tablas viii
Índice de figuras ix
Índice de anexos x
1. **Título** 1
2. **Resumen** 2
Abstract 3
3. **Introducción** 4
4. **Marco teórico** 5
5. **Metodología** 12
6. **Resultados** 14
6.1. Configuración de marco de trabajo DevOps 15
6.2. Sprint 1 - Planificación y Diseño de la Arquitectura 17
6.3. Sprint 2 - Desarrollo del Backend 32
6.4. Sprint 3 - Desarrollo del Frontend e Integración 39
6.5. Sprint 4 - Pruebas Finales, Despliegue e Implementación 43
7. **Discusión** 59
8. **Conclusiones** 60
9. **Recomendaciones** 61
10. **Bibliografia** 62
11. **Anexos** 64

Índice de tablas

Tabla 1. Fechas de los sprints del proyecto 14
Tabla 2. Integrantes del proyecto 14
Tabla 3. Requerimientos funcionales del proyecto 18
Tabla 4. Requerimientos no funcionales del proyecto 19

Índice de figuras

Figura 1. Épicas de los sprints 15
Figura 2. Features de los sprints 15
Figura 3. Historias de usuario de los sprints 16
Figura 4. Arquitectura del nuevo sistema web a implementar 22
Figura 5. Diagrama de clases 23
Figura 6. Diagrama de secuencia 23
Figura 7. Diagrama de casos de uso 24
Figura 8. Arquitectura de cluster de kubernetes 26
Figura 9. Organización de MONARKA con sus respectivos repositorios 27
Figura 10. Configuración docker-compose.yaml de la api 27
Figura 11. Configuración docker-compose.yaml del client 28
Figura 12. Configuración docker-compose.yaml del keycloak 28
Figura 13. Diagrama de Integración y despliegue continuos 29
Figura 14. Manifiesto del workflow del GitHub Actions de la API 30
Figura 15. Manifiesto del workflow del GitHub Actions del Client 31
Figura 16. Configuración de la base de datos de la API 34
Figura 17. Método con el endpoint de recuperación del access token 35
Figura 18. Decorator Auth para la validación de roles 36
Figura 19. Endpoint de la API con el retorno del token 37
Figura 20. Endpoint de la API con el retorno de datos del producto 37
Figura 21. Pantalla de inicio de la aplicación 41
Figura 22. Pantalla de ingreso con Keycloak 41
Figura 23. Pantalla de registro con Keycloak 41
Figura 24. Pantalla principal de la aplicación (hacer recibos) 42
Figura 25. Pantalla de inventario de la aplicación (lista de productos) 42
Figura 26. Pantalla de inventario de la aplicación (agregar y editar productos) 42
Figura 27. Configuración general del JMeter para las cargas de la API 43
Figura 28. Reporte resumen de listar productos de la API en JMeter 44
Figura 29. Reporte resumen de crear productos de la API en JMeter 44
Figura 30. Reporte resumen de actualizar productos de la API en JMeter 45
Figura 31. Reporte resumen de encontrar producto de la API en JMeter 45
Figura 32. Comando para conectar el AKS con la computadora 46
Figura 33. Manifiesto de la Certificación como dns de Cloudflare 46
Figura 34. Manifiesto de la configuración del Argo CD del servicio de Keycloak 47
Figura 35. Manifiesto del servicio de Keycloak 47
Figura 36. Manifiesto del despliegue de Keycloak 48
Figura 37. Manifiesto del ingreso de Keycloak 49
Figura 38. Manifiesto de la configuración del Argo CD del servicio del API 49
Figura 39. Manifiesto del servicio de la API 50
Figura 40. Manifiesto del despliegue de la API 51
Figura 41. Manifiesto del ingreso de la API 52
Figura 42. Manifiesto de la configuración del Argo CD del servicio del Client 53
Figura 43. Manifiesto del servicio del Client 53
Figura 44. Manifiesto del despliegue del Client 54
Figura 45. Manifiesto del ingreso del Client 55
Figura 46. Gráficas del clúster llamado monarka-aks de Azure Kubernetes Service 56
Figura 47. Gráfica en Grafana por el servicio de Argo CD 57
Figura 48. Gráfica en Grafana por el servicio de Keycloak 57
Figura 49. Gráfica en Grafana por el servicio de la API 58
Figura 50. Gráfica en Grafana por el servicio de Client 58

Índice de anexos

Anexo 1. Features detallados por funcionalidad 64
Anexo 2. Historias de usuarios 68
Anexo 3. Certificación de traducción del resumen 71

1. Título

Migración de una aplicación de escritorio de cobros a un sistema web mediante procesos DevOps. Caso de Aplicación: Empresa MONARKA

2. Resumen

El presente trabajo de titulación se enfoca en transformar el sistema de cobros que maneja la empresa MONARKA, la cual se dedica a la venta de zapatos, accesorios y ropa de mujer, migrando de una aplicación de escritorio a una plataforma web actualizada, a través del empleo de procesos DevOps para optimizar su desarrollo y despliegue. El problema principal surge de las restricciones tecnológicas y arquitectónicas del obsoleto sistema legacy que actualmente maneja la empresa, limitan su escalabilidad y afectan su eficiencia operativa.

La migración propuesta se fundamenta en una arquitectura moderna que utiliza NestJS y TypeScript para el backend, junto con Next.js y React para el frontend, implementando Keycloak para la gestión de autenticación y autorización. La solución incorpora un pipeline de Integración Continua y Entrega Continua (CI/CD) utilizando GitHub Actions, Docker y ArgoCD para el despliegue en Azure Kubernetes Service, asegurando así la automatización de pruebas y la calidad del código en cada etapa del desarrollo.

Garantizar la conservación de las funcionalidades actuales, proteger la seguridad de los datos, mejorar la experiencia del usuario y mantener la operación continua del negocio durante la transición son retos claves que el proyecto enfrenta. El implementar prácticas DevOps y metodologías ágiles, asegura una migración estructurada y efectiva, que permiten validaciones constantes y ajustes según la demanda del negocio.

Se espera que los resultados finales aumenten la capacidad de escalabilidad del sistema, que incluya una mayor facilidad de acceso para los usuarios, una disminución de los costos de mantenimiento y una arquitectura que sea capaz de adaptarse para integrar nuevas funcionalidades de manera rápida. Además de actualizar la infraestructura tecnológica de MONARKA, también se logra establecer una base sólida para su desarrollo futuro en un mercado retail, que cada vez se torna más competitivo y digitalizado.

***Palabras clave**: Migración de sistemas, DevOps, Aplicaciones web, Integración continua, Entrega continua, Kubernetes.*

Abstract

This thesis is centered on the transformation of the billing system managed by MONARKA., which sells shoes, accessories, and women's clothing, migrating from a desktop application to an updated web platform, using DevOps processes to optimize its development and deployment. The main problem arises from the technological and architectural restrictions of the obsolete legacy system currently managed by the company, which limit its scalability and affect its operational efficiency.

The proposed migration is based on a modern architecture that uses NestJS and TypeScript for the backend, along with Next.js and React for the front end, implementing Keycloak for authentication and authorization management. The solution incorporates a Continuous Integration and Continuous Delivery (CI/CD) pipeline using GitHub Actions, Docker, and Argo CD for deployment in Azure Kubernetes Service, thus ensuring test automation and code quality at each stage of development.

Maintaining current functionalities, ensuring data security, enhancing user experience, and supporting continuous business operations during the transition are key challenges faced by the project. Implementing DevOps practices and agile methodologies ensures a structured and effective migration, allowing for constant validation and adjustments based on business demand.

The results are expected to increase the system's scalability capacity, including greater ease of access for users, reduced maintenance costs, and an architecture that can adapt to integrate new functionalities quickly. In addition to updating MONARKA's technological infrastructure, it also establishes a solid foundation for its future development in a retail market that is becoming increasingly competitive and digitalized.

Keywords*: System migration, DevOps, Web applications, Microservices, Continuous integration, Continuous delivery, Kubernetes.*

3. Introducción

Actualmente, se ha llegado a ver a la transformación digital como una necesidad indispensable para las empresas que quieren mantenerse competitivas en el mercado. Para entrar en contexto, MONARKA, es una prestigiosa empresa la cual se enfoca en la comercialización de zapatos, accesorios y ropa femenina, que pretende desafiantemente transformar su sistema de cobros por uno moderno, convirtiendo de una aplicación típica de escritorio hacia una plataforma web actual y, sobre todo, que sea escalable.

La constante evolución de las tecnologías web y el aumento de la demanda de accesibilidad remota han hecho notar las deficiencias de los sistemas legacy, particularmente en aspectos de términos de escalabilidad, la facilidad de mantenimiento y la destreza de adaptarse a nuevos requerimientos de la empresa. Esta situación, en concordancia con la necesidad de mejorar procesos operativos y ofrecer una mejor experiencia al usuario, destaca la importancia de poner en práctica una migración tecnológica muy bien estructurada y planificada.

Este trabajo de fin de titulación tiene un enfoque en el desarrollo e implementación de una solución que resuelve los desafíos técnicos asociados a la migración, y establece las bases para una infraestructura tecnológica actual y sostenible. La propuesta integra tecnologías de vanguardia como NestJS y TypeScript para el backend, Next.js 14 y React para el frontend, implementando además prácticas DevOps que garantizan un desarrollo y despliegue eficientes mediante herramientas como GitHub Actions, Docker y ArgoCD en Azure Kubernetes Service.

El proceso de modernización o transformación tecnológica busca tanto preservar las funcionalidades críticas del sistema actual y fortalecer las capacidades operativas de MONARKA, logrando mejorar la accesibilidad, obtener una disminución de los costos de mantenimiento y establecer una arquitectura flexible que ayude a la adaptación ágil a las necesidades futuras de la empresa. La implementación de metodologías ágiles y prácticas DevOps asegura una transición controlada y eficiente, minimizando los riesgos operativos mientras se maximizan los beneficios de la modernización tecnológica.

4. Marco teórico

En el contexto actual de transformación digital, la migración de aplicaciones de escritorio a servicios web es un proceso crucial para las empresas que buscan mejorar su eficiencia operativa, escalabilidad y accesibilidad. La empresa MONARKA, dedicada a la venta de zapatos, accesorios y ropa de mujer, enfrenta la necesidad de modernizar su sistema de cobros para mantenerse competitiva en un mercado cada vez más digitalizado. Este marco teórico explorará los fundamentos, tecnologías, metodologías y beneficios asociados con la migración de sistemas, proporcionando una base sólida para la implementación de este proyecto en MONARKA.

Antecedentes y Evolución de los Sistemas de Cobros

Historia de los Sistemas de Cobros. Los sistemas de cobros han evolucionado significativamente desde los métodos manuales hasta los sistemas automatizados de escritorio y, más recientemente, los servicios web. Originalmente, las transacciones comerciales se registraban manualmente, lo que implicaba un alto riesgo de errores humanos y una gestión ineficiente del tiempo. Con la llegada de las computadoras, surgieron los primeros sistemas de cobros basados en software de escritorio, que ofrecían una mejora significativa en la precisión y eficiencia de las transacciones[1].

Estos sistemas de escritorio se caracterizaban por ser aplicaciones monolíticas, donde todas las funcionalidades estaban integradas en un único software que corría en la computadora local del usuario. Aunque estos sistemas mejoraron considerablemente la gestión de cobros, presentaban limitaciones en términos de escalabilidad, accesibilidad y capacidad de adaptación a nuevas tecnologías[1].

El papel de los Sistemas de Cobros en Empresas Retail. En el sector retail, donde opera MONARKA, los sistemas de cobros son fundamentales para la gestión eficiente de las ventas y el control financiero. Un sistema de cobros robusto y confiable permite a las empresas procesar transacciones de manera rápida y precisa, mejorar la satisfacción del cliente y mantener un registro detallado de las operaciones financieras. La eficiencia en la gestión de cobros se traduce en una mayor fluidez en las operaciones diarias y en una mejor capacidad para tomar decisiones basadas en datos precisos y actualizados[2].

Con la creciente adopción de tecnologías digitales, muchas empresas del sector retail han comenzado a migrar sus sistemas de cobros de escritorio a plataformas web, buscando aprovechar las ventajas que ofrecen en términos de escalabilidad, accesibilidad remota y flexibilidad para integrar nuevas funcionalidades[3].

Conceptos Fundamentales

Migración de Sistemas. La migración de sistemas es el proceso de trasladar una aplicación o sistema existente a un nuevo entorno o plataforma. Este proceso puede implicar la reingeniería del software, la refactorización del código, o la simple transferencia de datos y funcionalidades a un nuevo entorno. En el caso de MONARKA, la migración implica trasladar su sistema de cobros de una aplicación de escritorio a un servicio web[4].

Uno de los principales desafíos en la migración de sistemas es asegurar que la nueva plataforma mantenga o mejore la funcionalidad, rendimiento y seguridad del sistema original. Además, es crucial minimizar la interrupción del servicio durante la migración, especialmente en un sistema crítico como el de cobros[4].

Sistemas Web. Los sistemas web son aplicaciones que se ejecutan en un servidor y se acceden a través de un navegador web. A diferencia de las aplicaciones de escritorio, instaladas localmente en el ordenador del usuario, los sistemas web permiten el acceso remoto desde cualquier dispositivo conectado a Internet. Esto ofrece mayor flexibilidad y escalabilidad, ya que el sistema puede utilizarlo varios usuarios a la vez, sin las limitaciones de las aplicaciones de escritorio.

Los sistemas web modernos suelen estar construidos utilizando arquitecturas de microservicios, lo que permite una mayor modularidad y facilidad para integrar nuevas funcionalidades. Además, los sistemas web pueden ser fácilmente actualizados y mantenidos, ya que las actualizaciones se realizan en el servidor y no requieren intervención del usuario final.

DevOps. DevOps [5] es una cultura y conjunto de prácticas que combina el desarrollo de software (Dev) y las operaciones de tecnología de la información (Ops). Su objetivo es reducir el ciclo de desarrollo de sistemas, aumentar la frecuencia de entrega de software y mantener la alta calidad de las versiones, su práctica permite acelerar la entrega a través de la automatización, la colaboración, los comentarios rápidos y la mejor iterativa. Para MONARKA, la adopción de prácticas DevOps permitirá un flujo de trabajo más continuo y colaborativo entre los equipos de desarrollo y operaciones, lo que facilitará el despliegue y actualización constante del sistema de cobros .

Tecnologías Involucradas

NestJS. NestJS[6] es un framework progresivo de Node.js para la construcción de aplicaciones eficientes, escalables y mantenibles. Está diseñado para aprovechar al máximo las capacidades de TypeScript[7], proporcionando una estructura modular que facilita la

organización y gestión del código. En el contexto de la migración del sistema de cobros de MONARKA, NestJS será utilizado para desarrollar el backend del servicio web, aprovechando su robustez y facilidad de integración con otros servicios y bases de datos.

Una de las principales ventajas de NestJS es su compatibilidad con una arquitectura de microservicios, lo que permite a MONARKA construir un sistema altamente escalable y flexible. Además, su enfoque modular facilita el mantenimiento y la evolución del sistema a medida que las necesidades de la empresa cambian.

Next.js. Next.js[8] es un framework de desarrollo web basado en React[9], que permite la creación de aplicaciones web modernas con soporte para renderizado del lado del servidor (SSR)[10] y generación de sitios estáticos. Este framework será utilizado para el desarrollo del frontend del nuevo sistema de cobros de MONARKA, ofreciendo una experiencia de usuario optimizada y accesible desde cualquier dispositivo.

Next.js es especialmente adecuado para proyectos como este, ya que facilita la implementación de interfaces de usuario dinámicas y responsivas, mejorando la usabilidad del sistema. Además, su capacidad para integrar fácilmente con APIs y microservicios lo convierte en una excelente elección para un proyecto basado en una arquitectura moderna.

Keycloak. Keycloak [11] es un sistema de gestión de identidades y accesos de código abierto para la gestión de identidades y acceso, que facilita la autenticación y autorización de usuarios en aplicaciones web, elimina los costos de licencias asociados con otros sistemas comerciales, lo que resulta atractivo para startups y pequeñas empresas, además, reduce el tiempo y el esfuerzo para integrar la autenticación y autorización en sus aplicaciones. Para el sistema de MONARKA, Keycloak será esencial para implementar una capa de seguridad robusta, gestionando de manera centralizada los permisos y roles de los usuarios. Esto es fundamental en un sistema de cobros donde la seguridad de la información financiera y los datos de los clientes es prioritaria.

GitHub Actions. GitHub Actions [12] es una plataforma de integración y despliegue cambios de código de forma continua que te permite la automatización de flujos de trabajo de desarrollo, puedes crear flujos de trabajo y crear y probar cada solicitud de cambios en tu repositorio o desplegar solicitudes de cambios fusionadas a producción. Esto es útil para mantener la calidad del código y asegurar que las actualizaciones se implementen de forma segura y controlada. En el caso de MONARKA, GitHub Actions facilitará la implementación de un flujo DevOps, permitiendo pruebas y despliegues continuos.

Docker. Docker[13] es una plataforma de virtualización de contenedores que permite empaquetar aplicaciones junto con sus dependencias. Esto garantiza que el sistema se ejecute

de la misma manera en cualquier entorno, lo cual es crucial para proyectos como el de MONARKA, donde la consistencia y portabilidad del sistema son clave para evitar problemas de compatibilidad.

Otras Tecnologías Complementarias. Además de las tecnologías ya mencionadas, se utilizarán otras tecnologías y herramientas para garantizar el éxito del proyecto. Jest[14] se utilizará para realizar pruebas de unidad e integración, asegurando que cada componente del sistema funcione correctamente. Apache JMeter[15] será utilizado para realizar pruebas de rendimiento, garantizando que el sistema pueda manejar la carga esperada sin problemas.

Tecnologías de Contenedores y Despliegue

Azure Container Registry. Azure Container Registry [16] es un servicio de Microsoft Azure que permite almacenar y gestionar imágenes de contenedores de forma segura en la nube, además permite crear, proteger, examinar y administrar imágenes de contenedores y artefactos con una instancia de la distribución OCI con replicación geográfica y totalmente administrada. ACR es útil para proyectos que necesitan escalar sus aplicaciones utilizando contenedores, proporcionando una ubicación centralizada para almacenar imágenes Docker. En el caso de MONARKA, ACR facilitará la integración continua al permitir que las imágenes de la aplicación y sus componentes se gestionen de manera eficiente y estén siempre disponibles para los entornos de producción y desarrollo.

Argo CD. Argo CD [17] es una herramienta de entrega continua para Kubernetes que permite gestionar la implementación de aplicaciones mediante la metodología GitOps. Esto significa que los cambios en el código fuente de la aplicación pueden reflejarse automáticamente en el clúster de Kubernetes mediante Argo CD, proporcionando una forma automatizada y controlada de gestionar despliegues, se implementa como un controlador de Kubernetes que monitorea continuamente las aplicaciones en ejecución y compara el estado actual en vivo con el estado de destino deseado. En el contexto del sistema de MONARKA, Argo CD permitirá realizar actualizaciones de manera segura y eficiente, con un control detallado sobre las versiones implementadas y la capacidad de revertir cambios si es necesario.

Kubernetes. Kubernetes [18] es una plataforma de orquestación de contenedores de código abierto que automatiza la implementación, el escalado y la administración de aplicaciones en contenedores. Kubernetes facilita la administración de cargas de trabajo distribuidas y proporciona alta disponibilidad, escalabilidad y mantenimiento, tiene un ecosistema gran y en rápido crecimiento que facilita la automatización y la configuración declarativa. En este proyecto, Kubernetes permitirá implementar la aplicación de MONARKA

en una infraestructura en la nube, gestionando eficientemente los recursos y asegurando una operatividad continua.

Azure Kubernetes Service. Azure Kubernetes Service [19] es un servicio de Microsoft Azure que facilita la implementación, administración y escalabilidad de aplicaciones en Kubernetes. AKS es ideal para proyectos que requieren una gestión simplificada de Kubernetes, ya que Azure se encarga de la configuración y mantenimiento de la infraestructura de clústeres, un caso común para su uso es para realizar la migración mediante “lift-and-shift" a contenedores con AKS. Para MONARKA, AKS será fundamental para implementar la aplicación en un entorno escalable y confiable, asegurando que el sistema de cobros pueda manejar altos volúmenes de transacciones y usuarios sin interrupciones.

Prometheus. Prometheus [20] es un conjunto de herramientas de alerta y monitoreo de sistemas de código abierto independiente de cualquier empresa; recopila y almacena sus métricas como datos de series de tiempo, es decir, la información de las métricas se almacena con la marca de tiempo en la que se registró, junto con pares clave-valor opcionales llamados etiquetas. No depende del almacenamiento distribuido, los nodos de servidor individuales son autónomos, la recopilación de series temporales se realiza a través de un modelo de extracción a través de HTTP, el envío de series temporales se realiza mediante una puerta de enlace intermedia. Para la empresa MONARKA, servirá para brindar con confiabilidad, es un sistema al que se recurre durante una interrupción del servicio para diagnosticar problemas rápidamente, además, se puede confiar en él cuando otras partes de su infraestructura fallan y no necesita configurar una infraestructura extensa para usarlo.

Grafana. Grafana [21] es un software de código abierto que permite consultar, visualizar, generar alertas y explorar sus métricas, registros y rastros donde sea que estén almacenados. Los complementos de fuentes de datos le permiten consultar fuentes de datos, incluidas bases de datos de series temporales como Prometheus y CloudWatch, herramientas de registro como Loki y Elasticsearch, bases de datos NoSQL/SQL como Postgres, herramientas de CI/CD como GitHub. Es escrita en lenguaje Go, basada en la licencia Apache 2.0, el usario puede interpetar los datos fácilmente gracias al uso de gráficas e informes en tiempo real.

Metodologías Ágiles

Scrum. Scrum [22] es un framework ágil que facilita el desarrollo de proyectos complejos mediante la división del trabajo en ciclos de trabajo iterativos llamados sprints. En el proyecto de migración de MONARKA, Scrum permitirá una planificación flexible y la

capacidad de realizar ajustes continuos en función del progreso y la retroalimentación del equipo.

La estructura de Scrum, que incluye roles claros como el Product Owner, el Scrum Máster y el equipo de desarrollo, asegura que todas las partes interesadas estén alineadas y que el proyecto se mantenga en el camino correcto. Además, las reuniones diarias y las revisiones de sprint permiten identificar y resolver problemas rápidamente, minimizando los riesgos y mejorando la eficiencia del equipo.

Kanban. Kanban[23] es otra metodología ágil que complementa a Scrum al proporcionar una visualización clara del flujo de trabajo y ayudar a gestionar las tareas de manera más eficiente. En este proyecto, Kanban se utilizará para gestionar las pruebas y validaciones continuas, asegurando que cada tarea se complete antes de avanzar a la siguiente fase.

La combinación de Scrum y Kanban permite un enfoque ágil y adaptable, asegurando que el proyecto se mantenga flexible y responda a los cambios y necesidades emergentes durante el proceso de migración.

Beneficios de la Migración a Sistemas Web

Escalabilidad. Uno de los principales beneficios de la migración de sistemas de escritorio a servicios web es la escalabilidad. Los sistemas web, especialmente aquellos basados en una arquitectura de microservicios, pueden escalar fácilmente para manejar un mayor volumen de transacciones y usuarios sin comprometer el rendimiento. Esto es crucial para MONARKA, ya que le permitirá crecer y expandir sus operaciones sin enfrentar los cuellos de botella que suelen presentarse en los sistemas de escritorio[24].

Accesibilidad y Usabilidad. La accesibilidad es otro beneficio clave de los sistemas web. A diferencia de los sistemas de escritorio, que requieren una instalación local y están limitados al dispositivo donde se ejecutan, los sistemas web pueden ser accedidos desde cualquier dispositivo con conexión a Internet. Esto mejora la usabilidad y permite a los empleados de MONARKA acceder al sistema de cobros desde cualquier ubicación, facilitando el trabajo remoto y la flexibilidad operativa[25].

Eficiencia Operativa. La eficiencia operativa se ve significativamente mejorada con la migración a un sistema web. La capacidad de automatizar procesos, integrar fácilmente con otras plataformas y realizar actualizaciones sin interrupciones permite a MONARKA optimizar sus operaciones diarias. Además, un sistema web moderno puede ofrecer análisis en tiempo

real y reportes detallados, lo que ayuda a la empresa a tomar decisiones más informadas y a mejorar continuamente sus procesos[25].

Estudios y Referencias Previas

Revisión de Literatura. Numerosos estudios han explorado los beneficios y desafíos de la migración de sistemas de escritorio a servicios web. La mayoría de estos estudios destacan la importancia de una planificación cuidadosa y la selección de tecnologías adecuadas para garantizar el éxito de la migración. En particular, se han documentado casos de éxito en el sector retail, donde empresas similares a MONARKA han logrado mejoras significativas en su eficiencia operativa y satisfacción del cliente tras migrar a sistemas web[24].

Teorías Relacionadas. Existen varias teorías tecnológicas que respaldan la migración y modernización de software. La teoría de la adopción de la tecnología, por ejemplo, sugiere que las empresas que adoptan nuevas tecnologías de manera efectiva pueden lograr una ventaja competitiva significativa. Además, la teoría de la escalabilidad en sistemas distribuidos proporciona un marco para entender cómo los sistemas web pueden soportar el crecimiento de las empresas sin sacrificar el rendimiento[24].

Conclusiones del Marco Teórico

Síntesis de Conceptos Clave. El marco teórico presentado abarca los conceptos, tecnologías y metodologías fundamentales para la migración de un sistema de cobros de escritorio a un servicio web en MONARKA. La combinación de tecnologías modernas como NestJS y Next.js, junto con metodologías ágiles como Scrum y Kanban, proporciona una base sólida para un proyecto exitoso.

Importancia del Marco Teórico en el Proyecto. Este marco teórico comprende los desafíos y beneficios asociados a la migración de sistemas y guía la selección de tecnologías y enfoques metodológicos que maximizan el éxito del proyecto en MONARKA.

Preparación para el Desarrollo Práctica. Con un marco teórico bien fundamentado, el proyecto de migración está bien posicionado para pasar a la fase de desarrollo práctico, donde se implementarán las tecnologías y metodologías descritas para alcanzar los objetivos establecidos y mejorar significativamente la eficiencia operativa y la satisfacción del cliente en MONARKA.

5. Metodología

La metodología para la migración del sistema de cobros de escritorio a una plataforma web en la empresa MONARKA se ha diseñado en varias fases, empleando tecnologías modernas y frameworks ágiles que aseguren la eficiencia y calidad del proyecto. El proceso está estructurado en las siguientes fases, con un flujo de trabajo basado en la implementación de CI/CD, despliegue en Kubernetes y monitoreo avanzado.

Fases del Proyecto

Fase de Planificación y Análisis. La migración del sistema de escritorio a una plataforma web inicia con una revisión detallada de los requerimientos del sistema actual. En esta fase, se realizaron entrevistas con usuarios clave y se revisó el programa de escritorio existente para identificar y priorizar las funcionalidades críticas que deben ser mantenidas o mejoradas.

Con los datos recopilados, se definió el alcance del proyecto, incluyendo las funcionalidades a migrar, los objetivos de rendimiento y las mejoras en escalabilidad y usabilidad. La planificación se estructuró mediante el framework ágil Scrum, permitiendo iteraciones rápidas y revisiones continuas del progreso. Se establecieron sprints con objetivos claros, asignación de recursos y criterios de aceptación específicos para cada tarea, facilitando un seguimiento cercano del desarrollo.

Fase de Diseño y Arquitectura. En esta fase, se diseñó la arquitectura del nuevo sistema web. Se eligió NestJS para el backend, por su estructura modular y su compatibilidad con TypeScript, lo cual facilita el desarrollo y mantenimiento del código. Para el frontend, se utilizó Next.js, aprovechando sus capacidades de renderizado del lado del servidor (SSR) y generación de sitios estáticos, que mejoran la velocidad y la experiencia del usuario (UX).

Se desarrolló un prototipo de la interfaz de usuario en Next.js, centrado en la usabilidad y experiencia del usuario. Este prototipo fue evaluado en pruebas con usuarios clave para asegurar que cumple con las necesidades operativas de la empresa. Asimismo, se definió una estrategia de implementación de CI/CD mediante GitHub Actions, la cual facilita el control de versiones y automatización del despliegue.

Fase de Desarrollo e Implementación. Durante el desarrollo, se trabajó en sprints bajo Scrum, implementando la lógica del backend en NestJS y el frontend en Next.js. El backend se

construyó con una arquitectura de estructura modular, facilitando la escalabilidad y la futura integración con otros sistemas. La interfaz web se diseñó para ser responsive y accesible desde cualquier dispositivo. Y la autenticación se la ejecutó mediante keycloak, una herramienta open source, facilitando el proceso de ingreso y registro de usuarios.

Se adoptó un enfoque Kanban para gestionar las pruebas de unidad, integración y aceptación de usuario, con participación de los usuarios clave en pruebas de usabilidad y rendimiento. Las pruebas se automatizan utilizando herramientas como Jest para pruebas unitarias y JMeter para pruebas de carga y rendimiento.

Fase de Pruebas Finales y Despliegue. Previo al despliegue, se llevaron a cabo pruebas exhaustivas de todo el sistema, utilizando herramientas como Jest para validar la funcionalidad del backend y JMeter para simular carga y evaluar el rendimiento. El despliegue se realizó de forma progresiva, comenzando con un grupo de usuarios reducido para monitorear el rendimiento y satisfacción, antes del lanzamiento completo.

Para el despliegue en producción, se registró la imagen del Dockerfile en Azure Container Registry y se implementó en un clúster de Kubernetes gestionado mediante Argo CD. La contenedorización con Docker y Kubernetes permitió un escalado automático y mayor resiliencia.

Herramientas y Tecnologías Utilizadas. Para garantizar que cada fase se ejecute de forma rigurosa y eficiente, se emplearon tecnologías y metodologías modernas. El uso de NestJS y Next.js como principales frameworks de desarrollo, junto con Docker, Kubernetes y herramientas de CI/CD como GitHub Actions, permitió una estructura de despliegue y monitoreo avanzado.

La implementación de CI/CD incluyó GitHub Actions para la automatización del ciclo de vida de la aplicación, integrando tests automáticos y despliegues. Las imágenes Docker fueron registradas en Azure Container Registry y gestionadas mediante Kubernetes y Argo CD para asegurar un despliegue ágil y controlado. Para el monitoreo del sistema en producción, se implementaron Prometheus y Grafana, proporcionando visualización y métricas en tiempo real para supervisar el estado del sistema y anticipar posibles problemas de rendimiento o disponibilidad.

6. Resultados

Siguiendo la metodología descrita en el apartado anterior y respetando el cronograma del proyecto, se planificó el desarrollo del producto en cuatro sprints, con una duración aproximada de cuatro semanas cada uno, completando un ciclo total de cuatro meses. En los resultados se detalla la implementación de esta metodología, combinada con el marco de trabajo Azure DevOps, para abordar y resolver los desafíos del proyecto de la empresa MONARKA.

Tabla 1. Fechas de los sprints del proyecto

Sprint	Fecha de Inicio	Fecha de culminación
Sprint 1	15 - ago - 2024	10 - sep - 2024
Sprint 2	11 - sep - 2024	10 - oct - 2024
Sprint 3	11 - oct - 2024	7 - nov - 2024
Sprint 4	8 - nov - 2024	13 - dic - 2024

Por otra parte el equipo de trabajo está conformado por dos personas, lo cual se describe en la tabla 2.

Tabla 2. Integrantes del proyecto

Rol	Persona a cargo
Product Owner	Maria Rosa Cueva
Desarrollador	Edison Josue Ordonez

6.1. Configuración de marco de trabajo DevOps

6.1.1. Épicas. Para abarcar todo lo propuesto en la metodología descrita se crearon 4 épicas, definidas como Planificación y Diseño de Arquitectura, Desarrollo del Backend, Desarrollo del Frontend e Integración y Pruebas Finales, Despliegue e Implementación.

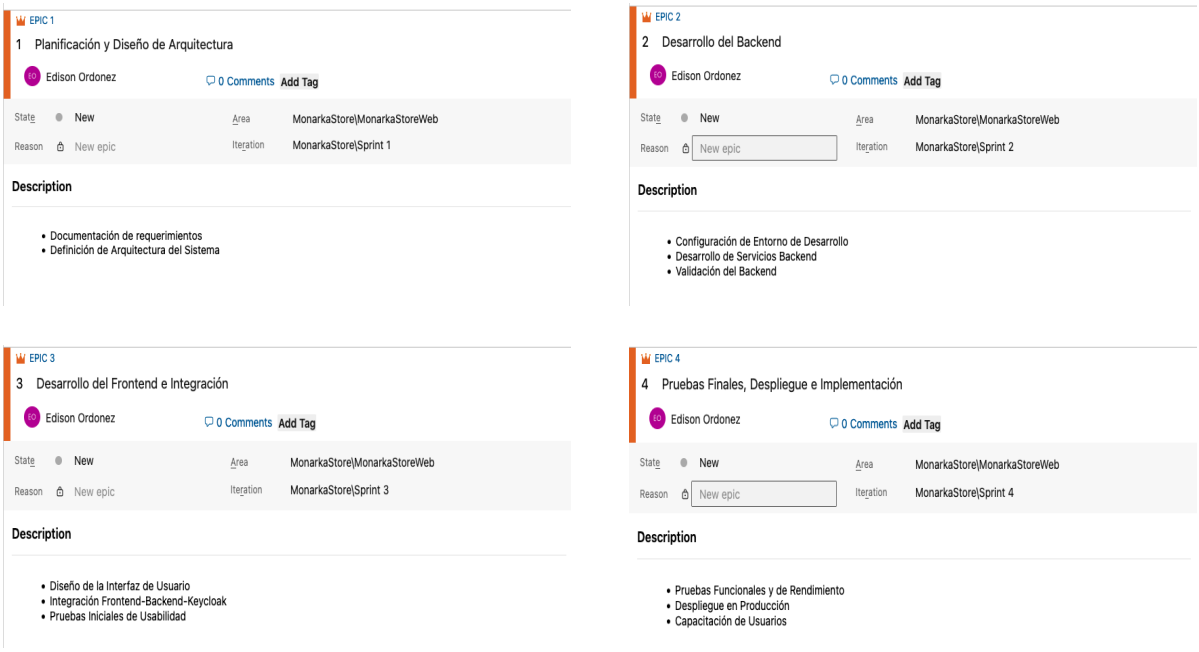


Figura 1. Épicas de los sprints

6.1.2. Features. En cuanto a los features, se observa en la Figura 2 la lista de las funcionalidades principales, para observar las funcionalidades a detalle se encuentra en el Anexo 1.

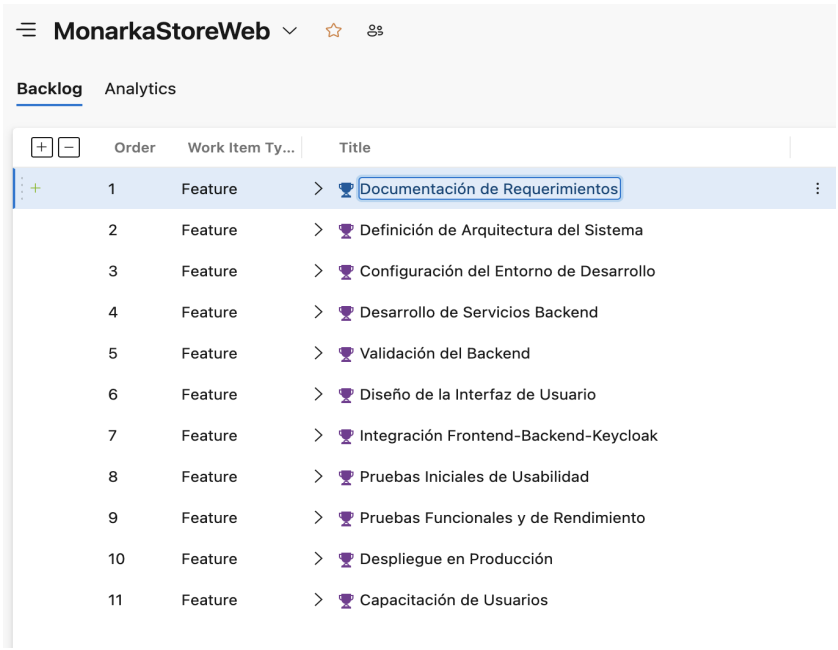


Figura 2. Features de los sprints

6.1.3. Historias de Usuario. En cuanto a las historias de usuario, se observa en la Figura 3 la lista de las historias de usuario, para observar las historias de usuario a detalle se encuentra en el Anexo 2.

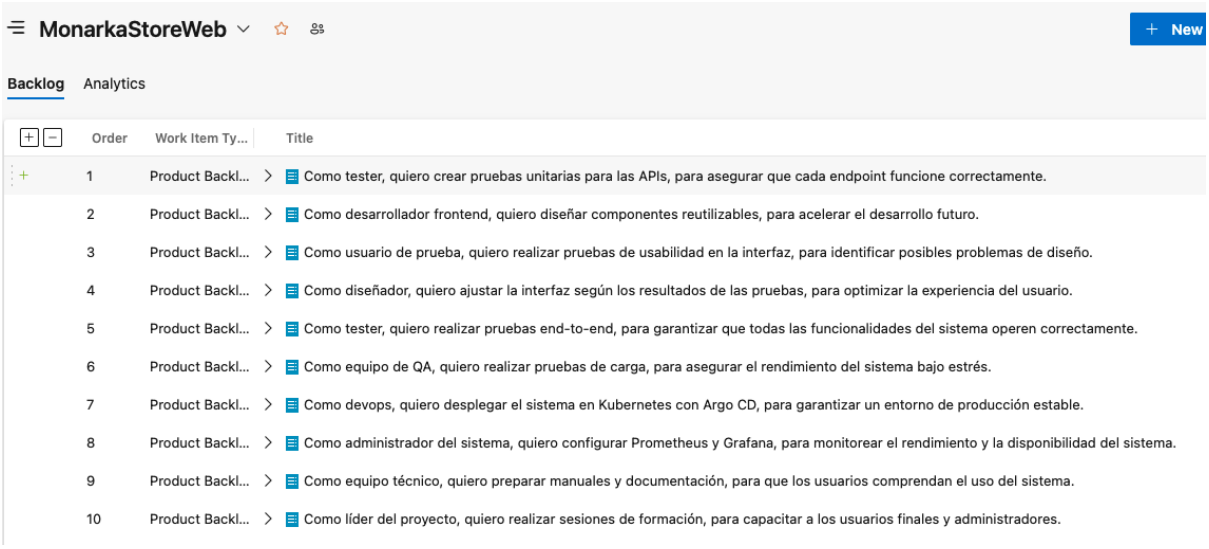


Figura 3. Historias de usuario de los sprints

6.2. Sprint 1 - Planificación y Diseño de la Arquitectura

El primer sprint, titulado “Planificación y Diseño de la Arquitectura”, se centró en la elaboración y definición de la documentación de los requerimientos funcionales y no funcionales, abarcando el alcance del sistema y sus funcionalidades clave. Durante este sprint, también se diseñó la arquitectura del sistema, incluyendo los diagramas correspondientes, y se definieron las bases para la arquitectura de integración y despliegue continuo.

6.2.1. Entrevistas con Usuarios Clave. Se realizaron entrevistas con usuarios clave del sistema de cobros para entender los procesos manuales existentes, así como los puntos de dolor y las expectativas sobre la automatización. Los usuarios aportaron retroalimentación sobre cómo debería mejorar el sistema en términos de usabilidad y tiempos de respuesta.

6.2.2. Revisión del Programa de Escritorio Existente. Se revisó la funcionalidad del programa de escritorio actual, destacando las áreas críticas que deben ser mantenidas, mejoradas, o simplificadas en la migración al nuevo sistema web. Entre ellas, se priorizaron funcionalidades de facturación, inventario y acceso seguro, esenciales para la operación de la empresa.

6.2.3. Requerimientos. Una vez realizadas las entrevistas y la revisión del escritorio existente se procedió a describir los requerimientos funcionales y no funcionales del proyecto.

6.2.3.1. Requisitos Funcionales. Estos son los que describen las características del sistema en general, funciones y comportamientos a producir para el proyecto, se describen los requerimientos en el siguiente cuadro.

Tabla 3. Requerimientos funcionales del proyecto

Requerimiento	Descripción	Importancia
Autenticación de usuarios	Permitir que los usuarios inicien sesión con un nombre de usuario y contraseña.	Alta
Procesamiento de pagos	Gestionar cobros y generación de recibos.	Alta
Gestión de inventario	Actualizar, eliminar y consultar productos de calzado, accesorios y ropa.	Alta
Categorías de productos	Organizar productos por categoría para facilitar la búsqueda.	Media
Visualización de inventario en tiempo real	Consultar disponibilidad de productos en tiempo real.	Alta
Calculadora de impuestos	Calcular automáticamente impuestos según el cambio de políticas fiscales.	Alta

6.2.3.2. Requisitos No Funcionales. Estos requerimientos demuestran cómo debe funcionar el sistema en términos de rendimiento, seguridad y confiabilidad entre otros.

Tabla 4. Requerimientos no funcionales del proyecto

Requerimiento	Descripción	Importancia
Disponibilidad	Garantizar la disponibilidad del sistema 99.9% del tiempo.	Alta
Seguridad	Implementar cifrado, autenticación de dos factores y control de acceso por roles.	Alta
Portabilidad	Compatibilidad con navegadores y dispositivos móviles.	Media
Mantenibilidad	Código modular y documentado para facilitar actualizaciones.	Media
Cumplimiento normativo	Cumplir con las normativas fiscales locales y de facturación electrónica.	Alta

6.2.4. Herramientas. Para abordar un sistema de cobros basado en web, con un pipeline de CI/CD y todas las especificaciones necesarias para MONARKA, se propone herramientas tecnológicas que abarcan todas las fases del proyecto.

6.2.4.1. Planificación y Gestión del Proyecto

Azure DevOps. Para coordinar y gestionar eficazmente las tareas del equipo, utilizamos Azure DevOps, una herramienta que facilita la implementación de metodologías ágiles como Scrum. A través de Azure DevOps, el equipo puede organizar y asignar tareas, planificar sprints, y establecer prioridades en cada fase de desarrollo. Además, la capacidad de generación de informes de Jira permite realizar un seguimiento del avance del proyecto, asegurando que todos los entregables se cumplan en los tiempos previstos.

6.2.4.2. Desarrollo Backend

NestJS. Para el desarrollo del backend, utilizamos NestJS debido a su estructura modular y su compatibilidad con patrones de diseño eficientes como controladores y servicios. NestJS permite construir una arquitectura escalable y modular, facilitando la incorporación de nuevas funcionalidades a medida que el proyecto crece. Su compatibilidad con TypeScript y su capacidad para construir aplicaciones orientadas a microservicios lo hacen ideal para el sistema de cobros de MONARKA.

Prisma. La manipulación de datos en PostgreSQL se gestiona a través de Prisma, un ORM que facilita la creación de consultas y migraciones automáticas de la base de datos. Prisma permite un acceso seguro y eficiente a la base de datos, mientras que su capacidad para definir y validar esquemas de datos ayuda a minimizar errores en la manipulación de la información, mejorando la confiabilidad y consistencia en el almacenamiento de datos.

PostgreSQL. Para la base de datos, PostgreSQL se selecciona como una solución robusta y segura, ideal para manejar datos sensibles como los relacionados con transacciones de cobros. La capacidad de PostgreSQL para gestionar consultas complejas y asegurar transacciones garantiza que los datos del sistema se mantengan organizados, integrados y confiables en cada proceso de la aplicación.

6.2.4.3. Desarrollo Frontend

Next.js. La experiencia del usuario es crucial, por lo que Next.js se emplea para el desarrollo del frontend, aprovechando su capacidad de renderizado del lado del servidor (SSR) y generación de sitios estáticos. Esta herramienta permite reducir los tiempos de carga, mejorar el SEO y optimizar la navegación. Su flexibilidad para crear interfaces modernas y responsive contribuye a una mejor experiencia de usuario.

React Query. Para la gestión de datos, React Query se utiliza en el frontend para sincronizar datos en tiempo real entre el cliente y el servidor. Esta herramienta minimiza las llamadas repetitivas a la API, mejorando la eficiencia de la aplicación y asegurando que la información visible esté siempre actualizada, sin necesidad de recargar o refrescar manualmente los datos.

6.2.4.4. Autenticación y Autorización

Keycloak. Para gestionar el acceso seguro al sistema, Keycloak es una solución completa que ofrece autenticación y autorización mediante estándares como OAuth2 y OpenID Connect. La implementación de Keycloak permite administrar roles y permisos de los usuarios, asegurando un sistema de acceso sólido, flexible y escalable. Además, su compatibilidad con autenticación multifactor añade una capa adicional de seguridad a los datos y funcionalidades de la aplicación.

6.2.4.5. Pruebas y Validación

Jest. En el backend, Jest se utiliza para realizar pruebas unitarias y de integración que aseguren que cada componente funcione correctamente. Las pruebas automatizadas ayudan a identificar y corregir errores en las primeras etapas de desarrollo, lo que reduce el riesgo de problemas durante el despliegue. Jest también genera reportes de cobertura de código, facilitando un control exhaustivo de la calidad.

React Testing Library. Para el frontend, React Testing Library se emplea para simular interacciones del usuario y validar que la interfaz reaccione como se espera. La biblioteca permite realizar pruebas centradas en la experiencia del usuario, asegurando que la funcionalidad del sistema sea intuitiva y responsiva, y que el diseño cumpla con los estándares de accesibilidad.

JMeter. JMeter es esencial para las pruebas de carga y rendimiento, permitiendo simular múltiples usuarios en diferentes escenarios. Esto asegura que el sistema pueda manejar el volumen de usuarios esperado sin afectar su rendimiento. JMeter ayuda a identificar y mitigar los cuellos de botella en el sistema, optimizando la infraestructura antes de que el sistema entre en producción.

6.2.4.6. Integración y Entrega Continua (CI/CD)

GitHub Actions. Para implementar CI/CD, GitHub Actions automatiza el ciclo de desarrollo, facilitando la ejecución de pruebas automáticas, construcción de contenedores Docker y despliegue en Kubernetes. Esta herramienta permite mantener un flujo de trabajo continuo y ágil, asegurando que cada nueva actualización o corrección se integre de manera eficiente y rápida al proyecto.

Docker. Docker permite que la aplicación y sus dependencias se empaqueten en contenedores, lo cual garantiza que se mantengan consistentes en diferentes entornos. Docker facilita el despliegue y escalabilidad de la aplicación, asegurando que el entorno de desarrollo sea idéntico al de producción, lo que reduce riesgos de errores en la implementación.

Azure Container Registry (ACR). Para almacenar las imágenes de Docker de la aplicación, ACR ofrece una solución segura y escalable en la nube de Microsoft Azure. Esta integración permite gestionar los contenedores y facilita la implementación de la aplicación en Kubernetes, asegurando la seguridad y accesibilidad de los contenedores para el despliegue.

6.2.4.7. Orquestación y Despliegue en la Nube

Kubernetes. Kubernetes se utiliza para gestionar la orquestación de los contenedores, distribuyéndolos en un clúster para asegurar alta disponibilidad y escalabilidad. Kubernetes no solo facilita la gestión de servicios y cargas de trabajo, sino que permite realizar despliegues continuos y rollback en caso de fallas, asegurando que el sistema se mantenga disponible y estable en todo momento.

Argo CD. En conjunto con Kubernetes, Argo CD permite automatizar el despliegue de nuevas versiones de la aplicación directamente desde el repositorio de código. Con Argo CD, el clúster de Kubernetes puede sincronizarse automáticamente con el estado deseado definido

en el repositorio, lo que asegura que cualquier cambio sea desplegado de manera controlada y monitorizada.

Azure Kubernetes Service (AKS). Para el despliegue en la nube, AKS proporciona un entorno Kubernetes gestionado, que facilita la escalabilidad, el balance de carga y la gestión de recursos en producción. AKS permite además una integración con Argo CD, simplificando la administración y supervisión de los microservicios de MONARKA en la nube, y asegura que el sistema esté preparado para manejar tráfico de usuarios en un entorno productivo.

6.2.4.8. Monitoreo y Logging

Grafana. Para la visualización de métricas y monitoreo del rendimiento, Grafana proporciona paneles personalizados que permiten observar el uso de recursos, tiempo de respuesta y otros indicadores clave. Esta herramienta ayuda a los administradores a detectar problemas de rendimiento y ajustar los recursos para optimizar el sistema en tiempo real.

Prometheus. Para la recolección y almacenamiento de métricas en tiempo real, Prometheus ofrece un sistema robusto que captura el rendimiento del clúster de Kubernetes y sus aplicaciones. La capacidad de Prometheus para enviar alertas facilita una respuesta rápida ante cualquier anomalía, asegurando que el sistema se mantenga estable y eficiente.

6.2.5. Fase de Diseño y Arquitectura

6.2.5.1. Arquitectura del Nuevo Sistema Web. La arquitectura del sistema se diseñó en sistema modular utilizando NestJS para el backend y Next.js para el frontend. Este diseño modular facilitó la creación de pipelines específicos para cada servicio, optimizando el despliegue continuo y permitiendo escalabilidad horizontal.

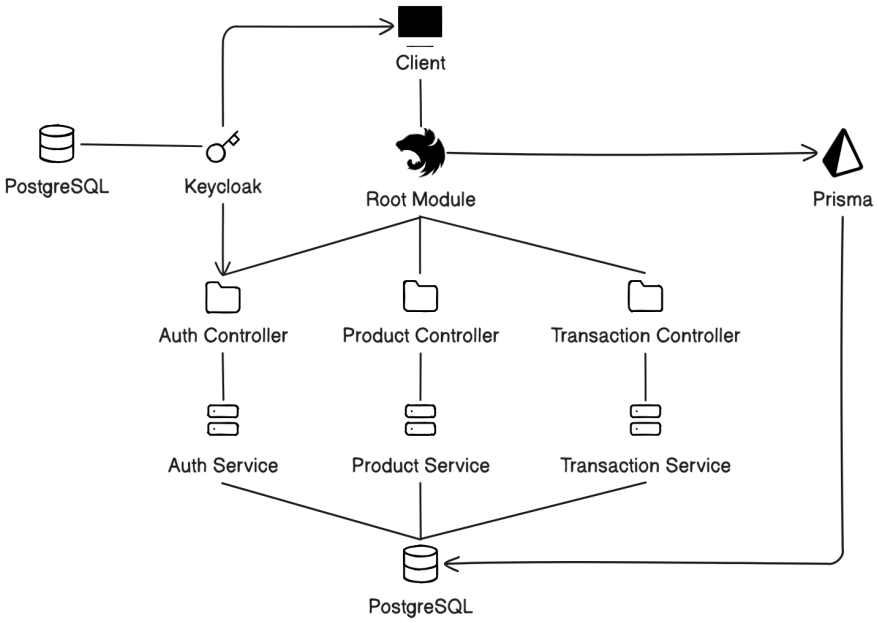


Figura 4. Arquitectura del nuevo sistema web a implementar

6.2.5.2. Diagramas de arquitectura.

Diagrama de Clases. Este diagrama modela la estructura del sistema, refiere las clases, atributos, métodos y relaciones de la aplicación MONARKA.

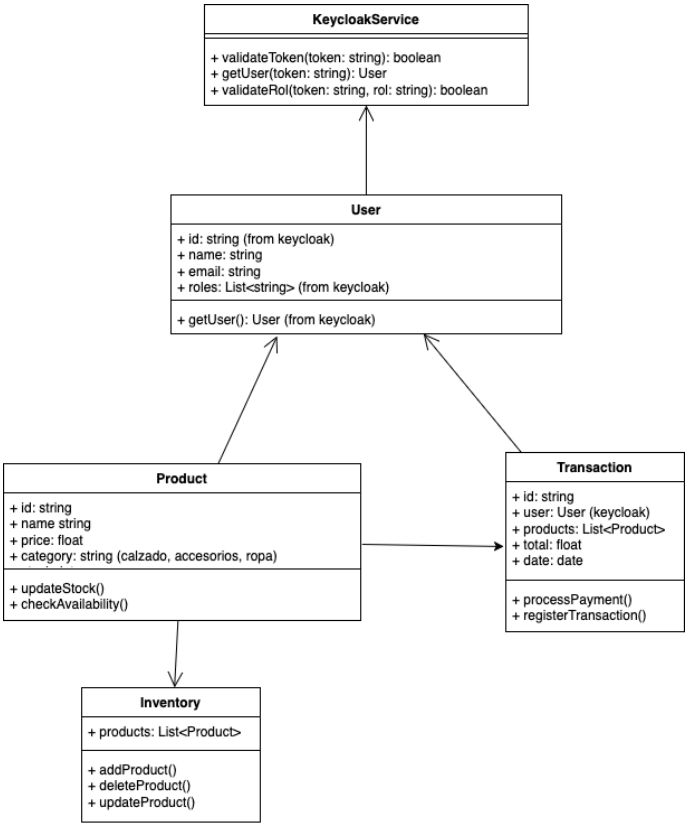


Figura 5. Diagrama de clases

Diagrama de Secuencia. Este diagrama muestra cómo los objetos interactúan entre sí a lo largo del tiempo para cumplir un caso de uso de la aplicación MONARKA.

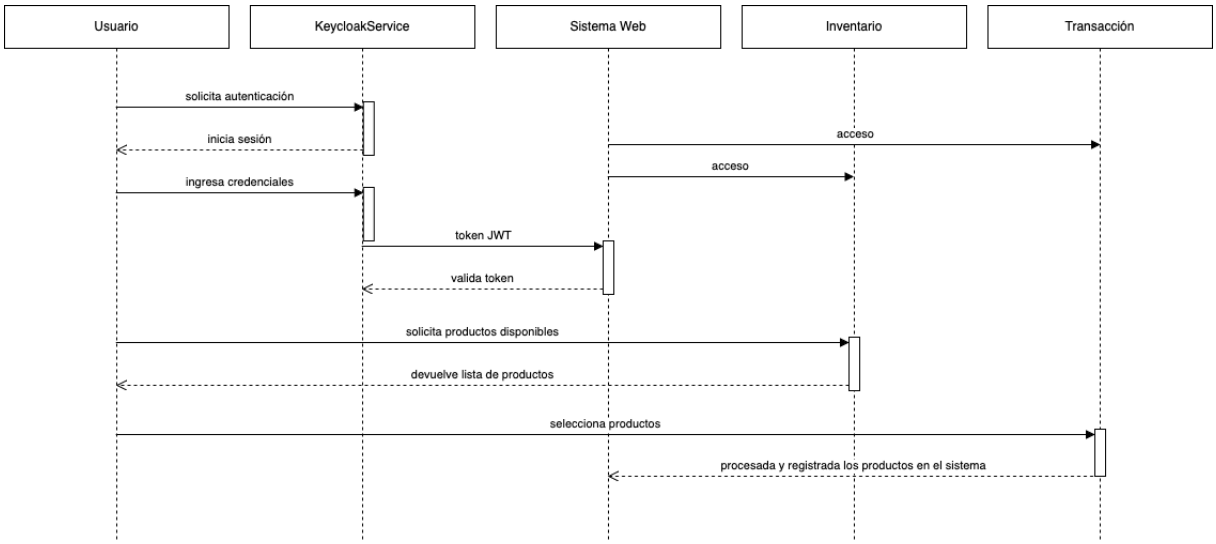


Figura 6. Diagrama de secuencia

Diagrama de Casos de Uso. Este diagrama representa las funcionalidades del sistema desde la perspectiva del usuario de la empresa MONARKA.

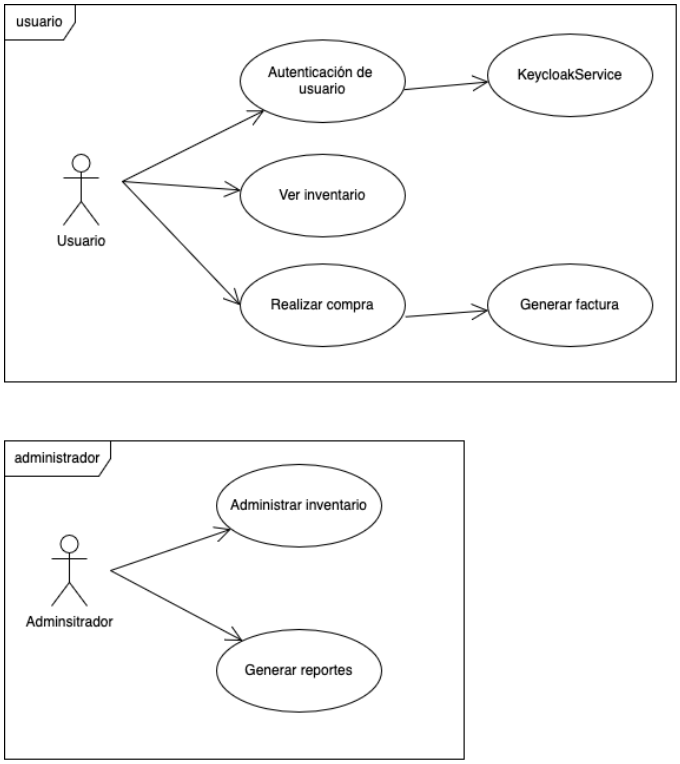


Figura 7. Diagrama de casos de uso

6.2.5.2. Arquitectura Kubernetes. La estructura del sistema se centra en la distribución de los servicios en contenedores, gestionados y orquestados por Kubernetes. Esta arquitectura permite una alta disponibilidad, escalabilidad, y facilita la automatización de despliegues y el manejo de configuraciones. A continuación, describimos los elementos clave de esta fase con un enfoque en Kubernetes.

6.2.5.2.1. Diseño del Entorno en Kubernetes

Estructura de Namespace y Aislamiento de Servicios. Para garantizar la organización y el aislamiento entre componentes, se define un conjunto de namespaces en el clúster de Kubernetes. Cada namespace alberga un conjunto de servicios específicos del sistema, tales como frontend (Next.js), backend (NestJS) y auth (Keycloak). Esta estructura no solo ayuda a mantener un entorno limpio y organizado, sino que también facilita la asignación de recursos, mejora la seguridad mediante el aislamiento de contenedores, y permite aplicar políticas de red específicas a cada servicio o módulo.

Despliegue de Aplicaciones como Deployments y Servicios. Los componentes clave de la aplicación, como el backend, frontend y sistema de autenticación, se despliegan en

Kubernetes mediante objetos Deployment, que gestionan el número de réplicas y aseguran la disponibilidad mediante reinicios automáticos en caso de fallos. Cada Deployment se asocia con un objeto Service que expone los contenedores dentro del clúster y permite la comunicación interna. Para el tráfico externo, se utiliza un Service de tipo LoadBalancer o NodePort, asegurando que los usuarios puedan acceder al frontend o a la API desde fuera del clúster.

6.2.5.2.2. Gestión de Despliegue y Sincronización con Argo CD

Implementación de Argo CD para Despliegue Controlado. Argo CD permite aplicar un enfoque GitOps al despliegue y actualización de la aplicación. La configuración deseada del clúster se define en un repositorio Git, y Argo CD supervisa y sincroniza el estado del clúster con este repositorio. De esta forma, cualquier cambio en el repositorio, como una nueva imagen de contenedor o actualización de configuración, se refleja automáticamente en el entorno de Kubernetes. Este enfoque garantiza una gestión de versiones precisa y una trazabilidad completa de cada cambio aplicado al clúster.

Configuración de Ingress para Enrutamiento y SSL. Un Ingress de NGINX en Kubernetes se utiliza para gestionar el tráfico externo hacia los servicios internos. Esta configuración facilita el enrutamiento mediante reglas que dirigen el tráfico a cada servicio (como el frontend o Keycloak) según la URL solicitada. Además, el Ingress se configura con certificados SSL, protegiendo el tráfico con HTTPS y asegurando la seguridad de los datos transmitidos. NGINX maneja también los encabezados de proxy, permitiendo que el backend identifique las solicitudes externas correctamente.

6.2.5.2.3. Escalabilidad y Disponibilidad de la Arquitectura

Autoescalado Horizontal y Mantenimiento de Alta Disponibilidad. Kubernetes permite configurar un autoescalado horizontal para cada Deployment en función de métricas como el uso de CPU o memoria. Esto significa que, en momentos de alta demanda, Kubernetes puede aumentar automáticamente el número de réplicas de un servicio para asegurar que el sistema responde eficientemente, mejorando la experiencia del usuario sin interrupciones. Asimismo, Kubernetes monitorea y reemplaza los contenedores que fallan, manteniendo la disponibilidad en todo momento.

Seguridad y Gestión de Configuraciones. Los secretos y variables de entorno, tales como credenciales de base de datos o claves de API, se gestionan mediante Secrets y ConfigMaps. Esto asegura que la información sensible no se codifique en los contenedores y se mantenga segura. Los Network Policies se configuran para controlar el tráfico entre los pods, limitando la comunicación a solo los servicios necesarios y reduciendo posibles vectores de ataque.

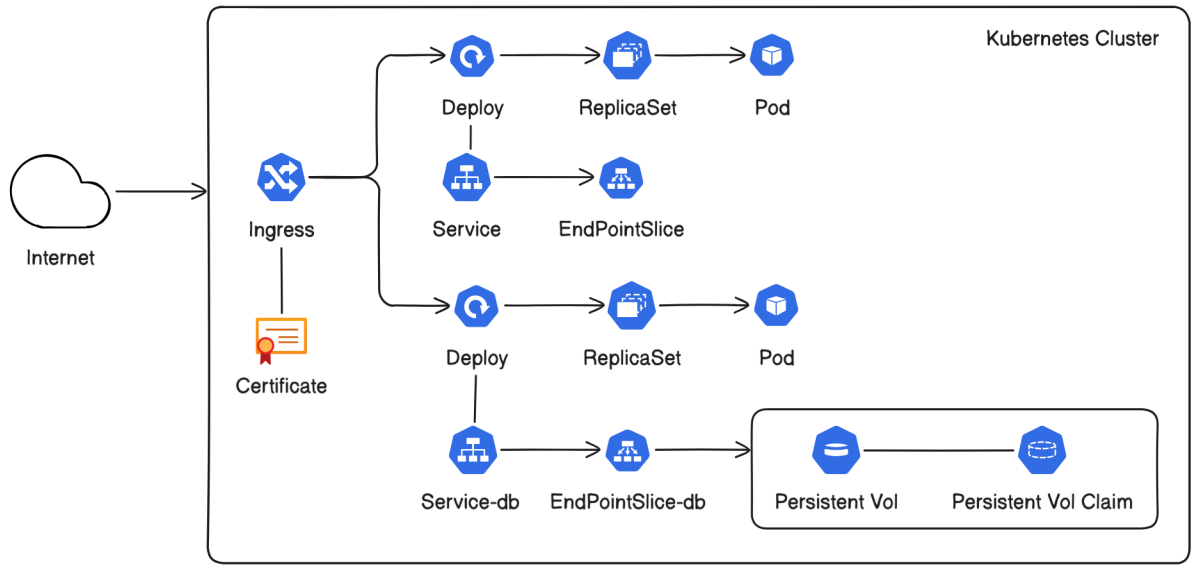


Figura 8. Arquitectura de cluster de kubernetes

6.2.6. Configuración del repositorio y el entorno local

6.2.6.1. Repositorio. El repositorio fue configurado en GitHub, una plataforma SaaS (Software as a Service) para el almacenamiento y gestión de código. Se organizó como una estructura de organización en GitHub para facilitar la administración, distribuyéndose en cuatro repositorios: **client**, **api**, **keycloak** y **monarka-project**.

- En el repositorio **client** se desarrolló la página web, correspondiente al frontend que interactúa directamente con el usuario.
- El repositorio **api** contiene los procesos internos y la lógica de negocio necesarios para resolver los requerimientos del proyecto.
- En **keycloak** se configuraron los contenedores relacionados, necesarios para el desarrollo local y la gestión de autenticación.
- Finalmente, **monarka-project** agrupa todo lo relacionado con el despliegue de la aplicación, incluyendo las configuraciones de Argo CD, certificados SSL y manifiestos de Kubernetes.

Toda la organización fue configurada como privada, garantizando que todos los repositorios estuvieran protegidos.

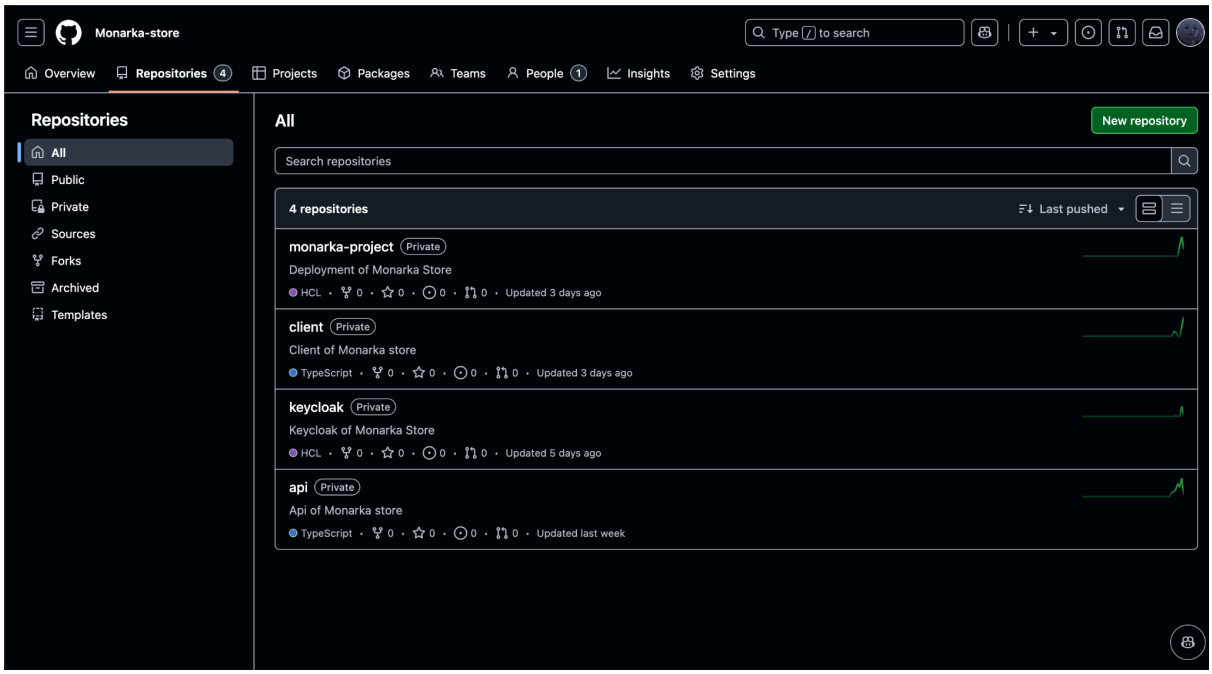
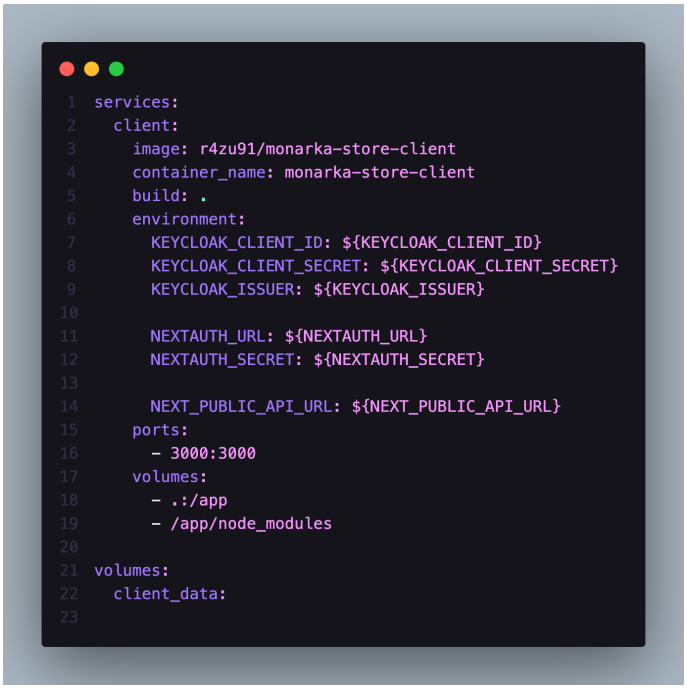


Figura 9. Organización de MONARKA con sus respectivos repositorios

6.2.6.2. Entorno Local. Para el entorno local, se utilizó Docker como herramienta principal, implementando Docker Compose para levantar los contenedores correspondientes a cada repositorio. Las configuraciones iniciales se establecieron por defecto, y la conexión a la base de datos se realizó utilizando PostgreSQL. Las imágenes empleadas fueron las proporcionadas por Bitnami para PostgreSQL y por Red Hat para Keycloak. Finalmente, se llevaron a cabo pruebas de conexión entre todos los componentes, confirmando su funcionamiento normal y esperado.

```yaml
services:
  api-db:
    image: bitnami/postgresql:14
    container_name: api-db
    environment:
      POSTGRESQL_DATABASE: ${POSTGRES_DB}
      POSTGRESQL_USERNAME: ${POSTGRES_USER}
      POSTGRESQL_PASSWORD: ${POSTGRES_PASSWORD}
    volumes:
      - db_data:/bitnami/postgresql
    ports:
      - 5432:5432

volumes:
  db_data:
```

Figura 10. Configuración docker-compose.yaml de la api



```yaml
services:
  client:
    image: r4zu91/monarka-store-client
    container_name: monarka-store-client
    build: .
    environment:
      KEYCLOAK_CLIENT_ID: ${KEYCLOAK_CLIENT_ID}
      KEYCLOAK_CLIENT_SECRET: ${KEYCLOAK_CLIENT_SECRET}
      KEYCLOAK_ISSUER: ${KEYCLOAK_ISSUER}

      NEXTAUTH_URL: ${NEXTAUTH_URL}
      NEXTAUTH_SECRET: ${NEXTAUTH_SECRET}

      NEXT_PUBLIC_API_URL: ${NEXT_PUBLIC_API_URL}
    ports:
      - 3000:3000
    volumes:
      - .:/app
      - /app/node_modules

volumes:
  client_data:
```

Figura 11. Configuración docker-compose.yaml del client

```yaml
services:
  keycloak-db:
    image: bitnami/postgresql:14
    container_name: keycloak-db
    environment:
      POSTGRESQL_DATABASE: ${KEYCLOAK_POSTGRES_DB}
      POSTGRESQL_USERNAME: ${KEYCLOAK_POSTGRES_USER}
      POSTGRESQL_PASSWORD: ${KEYCLOAK_POSTGRES_PASSWORD}
    volumes:
      - db_data:/bitnami/postgresql
    healthcheck:
      test: ['CMD', 'pg_isready', '-h', 'localhost', '-U', 'keycloak']
      interval: 10s
      timeout: 5s
      retries: 5

  keycloak:
    image: quay.io/keycloak/keycloak:26.0.6
    container_name: monarka-keycloak
    environment:
      KC_DB_URL_HOST: jdbc:postgresql://keycloak-db:5432/${KEYCLOAK_POSTGRES_DB}
      KEYCLOAK_DATABASE_PORT: 5432
      KEYCLOAK_DATABASE_NAME: ${KEYCLOAK_POSTGRES_DB}
      KEYCLOAK_DATABASE_USER: ${KEYCLOAK_POSTGRES_USER}
      KEYCLOAK_DATABASE_PASSWORD: ${KEYCLOAK_POSTGRES_PASSWORD}

      KC_BOOTSTRAP_ADMIN_USERNAME: ${KEYCLOAK_ADMIN}
      KC_BOOTSTRAP_ADMIN_PASSWORD: ${KEYCLOAK_ADMIN_PASSWORD}

      KC_HEALTH_ENABLED: 'true'
      KC_METRICS_ENABLED: 'true'
    command: start-dev
    ports:
      - 8080:8080
    depends_on:
      - keycloak-db

volumes:
  db_data:
  keycloak:
```

Figura 12. Configuración docker-compose.yaml del keycloak

6.2.7. Configuración del pipeline de CI/CD con Github Actions. Se utilizó a GitHub Actions como la herramienta principal para automatizar el pipeline de CI/CD. Se configuraron workflows específicos por separado tanto del backend como del frontend. Cada commit disparaba la construcción de la aplicación y ejecutaba pruebas unitarias en el backend.

En la rama donde se realizaban los cambios, el pipeline estaba configurado para desplegar automáticamente la aplicación en el entorno de destino. Este proceso comenzaba con la ejecución de pruebas, y una vez superadas, se construían las imágenes Docker correspondientes tanto para el backend como para el frontend. Estas imágenes se almacenaban en el registro de contenedores de Azure (Azure Container Registry). Posteriormente, se actualizaban los recursos en **Argo CD**, lo que permitía sincronizar los cambios y observar las actualizaciones en producción sin interrupciones ni inconvenientes.

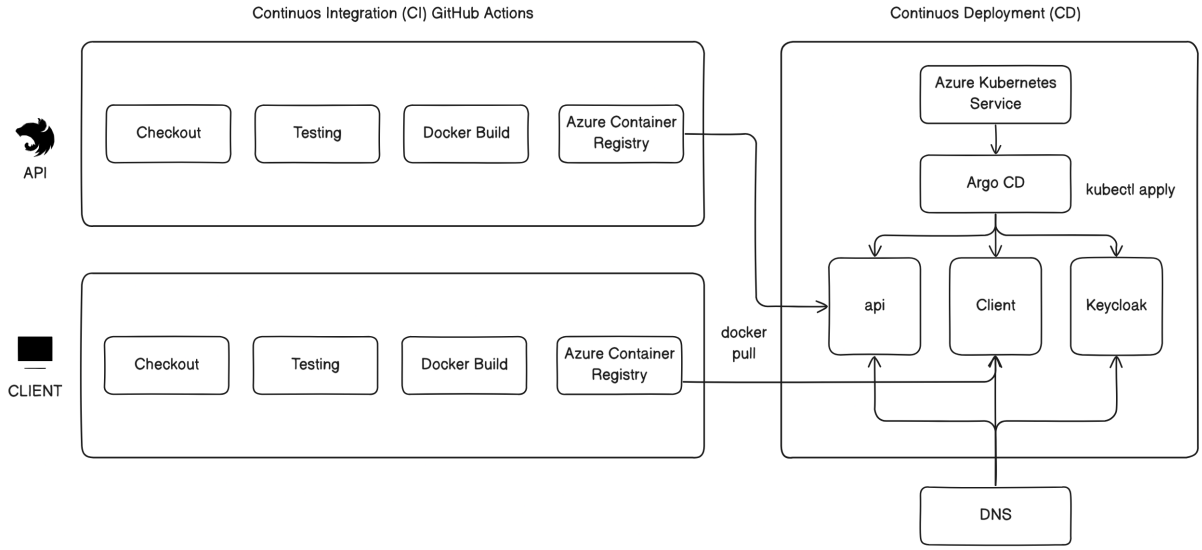
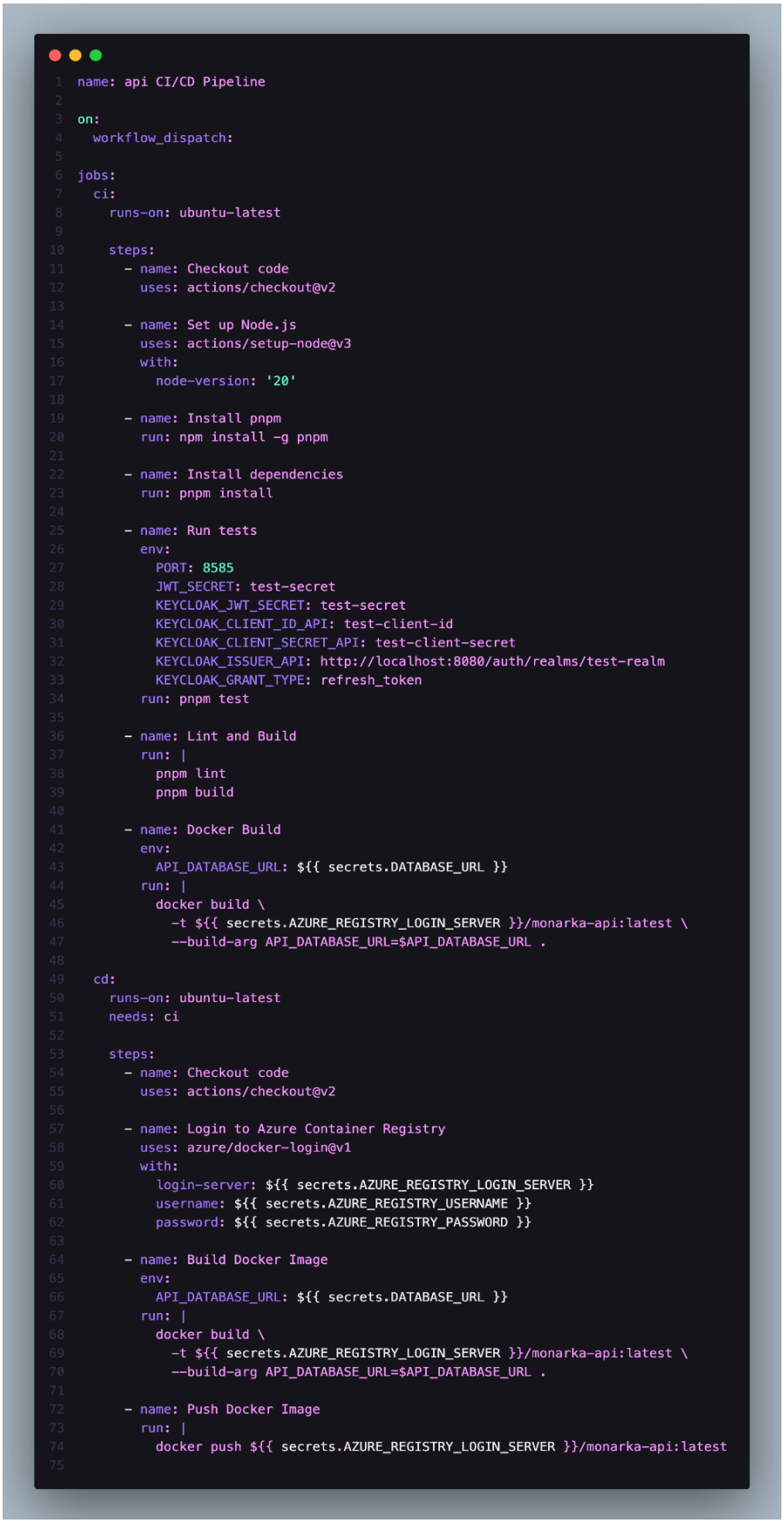


Figura 13. Diagrama de Integración y despliegue continuos



```yaml
name: api CI/CD Pipeline

on:
  workflow_dispatch:

jobs:
  ci:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Set up Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '20'

      - name: Install pnpm
        run: npm install -g pnpm

      - name: Install dependencies
        run: pnpm install

      - name: Run tests
        env:
          PORT: 8585
          JWT_SECRET: test-secret
          KEYCLOAK_JWT_SECRET: test-secret
          KEYCLOAK_CLIENT_ID_API: test-client-id
          KEYCLOAK_CLIENT_SECRET_API: test-client-secret
          KEYCLOAK_ISSUER_API: http://localhost:8080/auth/realms/test-realm
          KEYCLOAK_GRANT_TYPE: refresh_token
        run: pnpm test

      - name: Lint and Build
        run: |
          pnpm lint
          pnpm build

      - name: Docker Build
        env:
          API_DATABASE_URL: ${{ secrets.DATABASE_URL }}
        run: |
          docker build \
            -t ${{ secrets.AZURE_REGISTRY_LOGIN_SERVER }}/monarka-api:latest \
            --build-arg API_DATABASE_URL=$API_DATABASE_URL .

  cd:
    runs-on: ubuntu-latest
    needs: ci

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Login to Azure Container Registry
        uses: azure/docker-login@v1
        with:
          login-server: ${{ secrets.AZURE_REGISTRY_LOGIN_SERVER }}
          username: ${{ secrets.AZURE_REGISTRY_USERNAME }}
          password: ${{ secrets.AZURE_REGISTRY_PASSWORD }}

      - name: Build Docker Image
        env:
          API_DATABASE_URL: ${{ secrets.DATABASE_URL }}
        run: |
          docker build \
            -t ${{ secrets.AZURE_REGISTRY_LOGIN_SERVER }}/monarka-api:latest \
            --build-arg API_DATABASE_URL=$API_DATABASE_URL .

      - name: Push Docker Image
        run: |
          docker push ${{ secrets.AZURE_REGISTRY_LOGIN_SERVER }}/monarka-api:latest
```

Figura 14. Manifiesto del workflow del GitHub Actions de la API

```yaml
name: client CI/CD Pipeline

on:
  workflow_dispatch:

jobs:
  ci:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '20'

      - name: Install pnpm
        run: npm install -g pnpm

      - name: Install dependencies
        run: pnpm install

      - name: Lint and Build
        run: |
          pnpm lint
          pnpm build

      - name: Docker Build
        run: |
          docker build \
            -t ${{ secrets.AZURE_REGISTRY_LOGIN_SERVER }}/monarka-client:latest .

  cd:
    runs-on: ubuntu-latest
    needs: ci

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Log in to Azure Container Registry
        uses: azure/docker-login@v1
        with:
          login-server: ${{ secrets.AZURE_REGISTRY_LOGIN_SERVER }}
          username: ${{ secrets.AZURE_REGISTRY_USERNAME }}
          password: ${{ secrets.AZURE_REGISTRY_PASSWORD }}

      - name: Build Docker image
        run: docker build -t ${{ secrets.AZURE_REGISTRY_LOGIN_SERVER }}/monarka-client:latest .

      - name: Push Docker image
        run: docker push ${{ secrets.AZURE_REGISTRY_LOGIN_SERVER }}/monarka-client:latest

```

Figura 15. Manifiesto del workflow del GitHub Actions del Client

6.3. Sprint 2 - Desarrollo del Backend

6.3.1. Configuración de la base de datos. Una vez configurada la base de datos utilizando el contenedor bitnami/postgresql, se empleó el ORM (Object-Relational Mapper) Prisma para simplificar la definición y gestión de las tablas relacionales de la base de datos asociada a la API del proyecto. La configuración detallada puede observarse en la figura 16. Es importante mencionar que, fuera de esta configuración, no se requirió ninguna adicional, ya que Keycloak utiliza su propia configuración de base de datos. Para su funcionamiento, fue suficiente establecer la conexión con el contenedor correspondiente.

6.3.2. Keycloak. Una vez iniciados los contenedores de la base de datos de Keycloak y la imagen de Keycloak, se accede al panel de administración mediante la URL predeterminada http://localhost:8080. El inicio de sesión inicial utiliza las credenciales por defecto: usuario admin y contraseña admin. Al ingresar, es necesario actualizar estas credenciales con unas nuevas.

Posteriormente, se creó un nuevo **realm** llamado monarka para gestionar los usuarios del proyecto, y se llevaron a cabo las siguientes configuraciones:

Client Scope

- Crear un nuevo *Client Scope* llamado openid con tipo Default.
- Activar la opción **Include in token scope** y guardar.

Cliente para la API

- Crear un cliente llamado nestjs.
- Activar **Client authentication** y habilitar la opción **Service accounts roles**.
- Configurar la **Root URL** como http://localhost:8585 (correspondiente a la API local) y guardar.

Cliente para la frontend

- Crear un cliente llamado nextjs.
- Activar **Client authentication**, pero desactivar la opción **Direct access grants**.
- Configurar los siguientes valores:
    - **Valid redirect URIs:** http://localhost:3000/api/auth/callback/keycloak.
    - **Valid post logout redirect URIs:** http://localhost:3000.
- Guardar los cambios.

Configuración del Realm

- En **Realm Settings**:
    - Activar las opciones en el apartado **Login**:
        - **User registration**.
        - **Forgot password**.
        - **Remember me**.
        - **Email as username**.
    - En **Localization**, activar la **Internationalization**, configurar el idioma predeterminado como español y guardar.
    - En **Tokens**, modificar el parámetro **Access Token Lifespan** en la sección **Access tokens** a 30 minutos y guardar.

Políticas de Contraseña

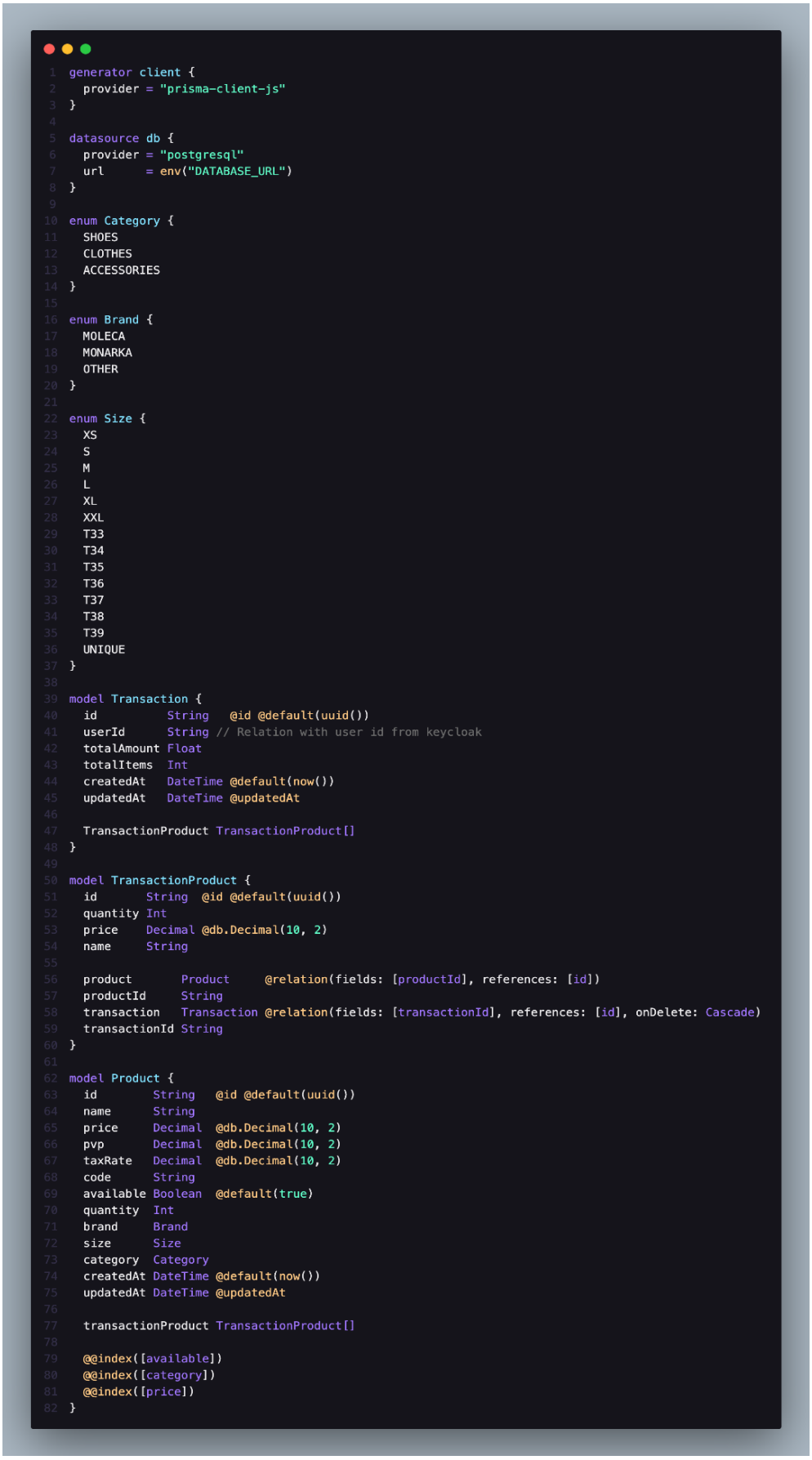
- En **Authentication > Policies**, configurar las políticas de contraseña con los siguientes requisitos:
    - Mínimo 8 caracteres.
    - No debe ser el nombre de usuario ni el correo electrónico.
    - No debe contener el usuario.
    - Debe incluir al menos:
        - Un carácter especial.
        - Una letra mayúscula.
        - Una letra minúscula.
        - Un dígito.

Creación de Roles

- En la sección **Realm roles**, se crea el rol de administrador de la API y guardas.
- Se asigna a **Associated roles** todos los roles del **realm roles**.

Creación de Usuario de Prueba

- En la sección **Usuarios**, crear un usuario de prueba con el rol de “admin” para verificar las configuraciones realizadas.



```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

enum Category {
  SHOES
  CLOTHES
  ACCESSORIES
}

enum Brand {
  MOLECA
  MONARKA
  OTHER
}

enum Size {
  XS
  S
  M
  L
  XL
  XXL
  T33
  T34
  T35
  T36
  T37
  T38
  T39
  UNIQUE
}

model Transaction {
  id          String   @id @default(uuid())
  userId      String // Relation with user id from keycloak
  totalAmount Float
  totalItems  Int
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  TransactionProduct TransactionProduct[]
}

model TransactionProduct {
  id       String  @id @default(uuid())
  quantity Int
  price    Decimal @db.Decimal(10, 2)
  name     String

  product       Product     @relation(fields: [productId], references: [id])
  productId     String
  transaction   Transaction @relation(fields: [transactionId], references: [id], onDelete: Cascade)
  transactionId String
}

model Product {
  id        String   @id @default(uuid())
  name      String
  price     Decimal  @db.Decimal(10, 2)
  pvp       Decimal  @db.Decimal(10, 2)
  taxRate   Decimal  @db.Decimal(10, 2)
  code      String
  available Boolean  @default(true)
  quantity  Int
  brand     Brand
  size      Size
  category  Category
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  transactionProduct TransactionProduct[]

  @@index([available])
  @@index([category])
  @@index([price])
}
```

Figura 16. Configuración de la base de datos de la API

6.3.3. Desarrollo del backend con NestJS.

6.3.3.1. Configuración del Módulo de Autenticación. La configuración del módulo de autenticación fue el primer paso para garantizar la seguridad y el control de acceso en el sistema. Este módulo está conectado directamente con la API de Keycloak, lo que permite gestionar la recuperación y validación de tokens de autorización de forma eficiente.

6.3.3.1.1. Integración con Keycloak.

Recuperación del Token. El módulo de autenticación utiliza los endpoints de Keycloak para autenticar usuarios y obtener tokens de acceso para la aplicación, y estos tokens contienen información necesaria para validar las credenciales del usuario y roles asociados. Podemos observar en la Figura 17.

```
async login(loginDto: LoginDto) {
    try {
      const { data } = await axios.post(
        `${envs.keycloakIssuerApi}/protocol/openid-connect/token`,
        new URLSearchParams({
          client_id: envs.keycloakClientIdApi,
          client_secret: envs.keycloakClientSecretApi,
          grant_type: envs.keycloakGrantType,
          username: loginDto.email,
          password: loginDto.password,
        }),
        {
          headers: {
            'Content-Type': 'application/x-www-form-urlencoded',
          },
        },
      );
      if (!data.access_token) {
        throw new UnauthorizedException('Invalid credentials');
      }
      return { access_token: data.access_token };
    } catch (error) {
      if (axios.isAxiosError(error)) {
        console.error('Error Response Data:', error.response?.data);
        console.error('Error Response Status:', error.response?.status);
        throw new UnauthorizedException(
          `Invalid credentials: ${
            error.response?.data?.error_description || error.message
          }`,
        );
      }
      throw new UnauthorizedException('An unexpected error occurred');
    }
  }
```

Figura 17. Método con el endpoint de recuperación del access token

Uso del JWT (JSON Web Token). Para implementar la autorización basada en roles, se empleó la estrategia JWT, este enfoque permite extraer y verificar los datos del token, asegurando que solo usuarios con roles específicos puedan acceder a determinadas funcionalidades de la aplicación.

6.3.3.1.2. Implementación del Patrón Decorator. El patrón de diseño **Decorator** se utiliza para manejar la autorización por los roles de la aplicación. Este patrón facilita la anotación de controladores y métodos específicos con los roles requeridos, simplificando la validación y mejorando la legibilidad del código. La aplicación al recibir la solicitud, la estrategia JWT verifica el token, se compara con los roles autorizados y valida si el usuario tiene los permisos para acceder al recurso solicitado. La Figura 18 describe el uso del decorador para verificar el token.

Figura 18. Decorator Auth para la validación de roles

6.3.3.1.3. Pruebas de Módulo. Se realizaron pruebas unitarias para garantizar que las funcionalidades de autenticación y autorización funcionen correctamente en diferentes escenarios. Las pruebas incluyeron la validación de tokens válidos e inválidos, la autorización basada en roles y el manejo de errores comunes.

6.3.3.2. Desarrollo del Módulo de Productos. El módulo de productos fue desarrollado para gestionar las operaciones CRUD (Crear, Leer, Actualizar y Eliminar) relacionadas con los productos del sistema. Este módulo es una parte esencial para el proyecto, ya que permite administrar y consultar la información de los productos de forma eficiente y segura.

6.3.3.2.1. Protección de Rutas. Todas las rutas del módulo de producto fueron protegidas mediante la integración con Keycloak, esta autenticación asegura que solamente usuarios autenticados puedan acceder a las rutas protegidas, además, la autorización basada en roles garantiza que algunas operaciones, como eliminar productos, solo esté disponible para usuarios con permisos de administrador.

6.3.3.2.2. Pruebas de Módulo. Se utilizaron herramientas como Postman para verificar el correcto funcionamiento de los endpoints. Estas pruebas incluyeron la generación y validación de tokens, así como el acceso a las rutas protegidas. En la figura 19 se muestra un

ejemplo de la respuesta del token generado y en la Figura 20 se observa los productos de la base de datos. Todas estas pruebas unitarias se desarrollaron con Jest y Supertest.

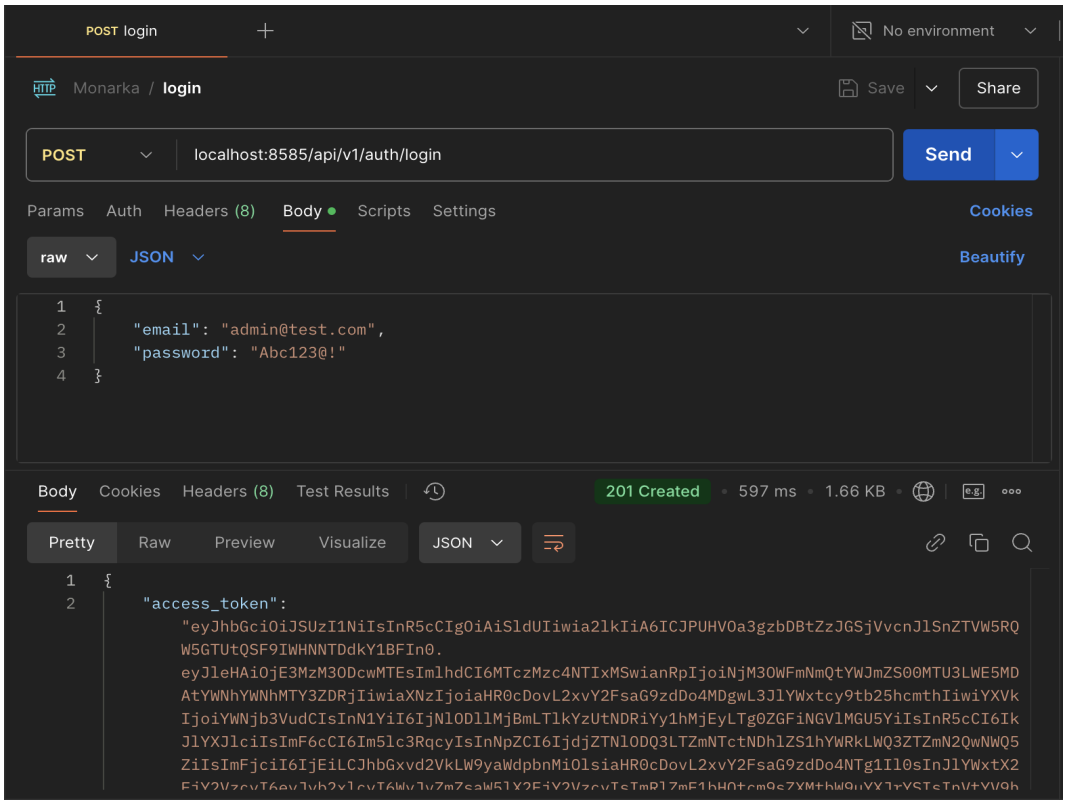


Figura 19. Endpoint de la API con el retorno del token

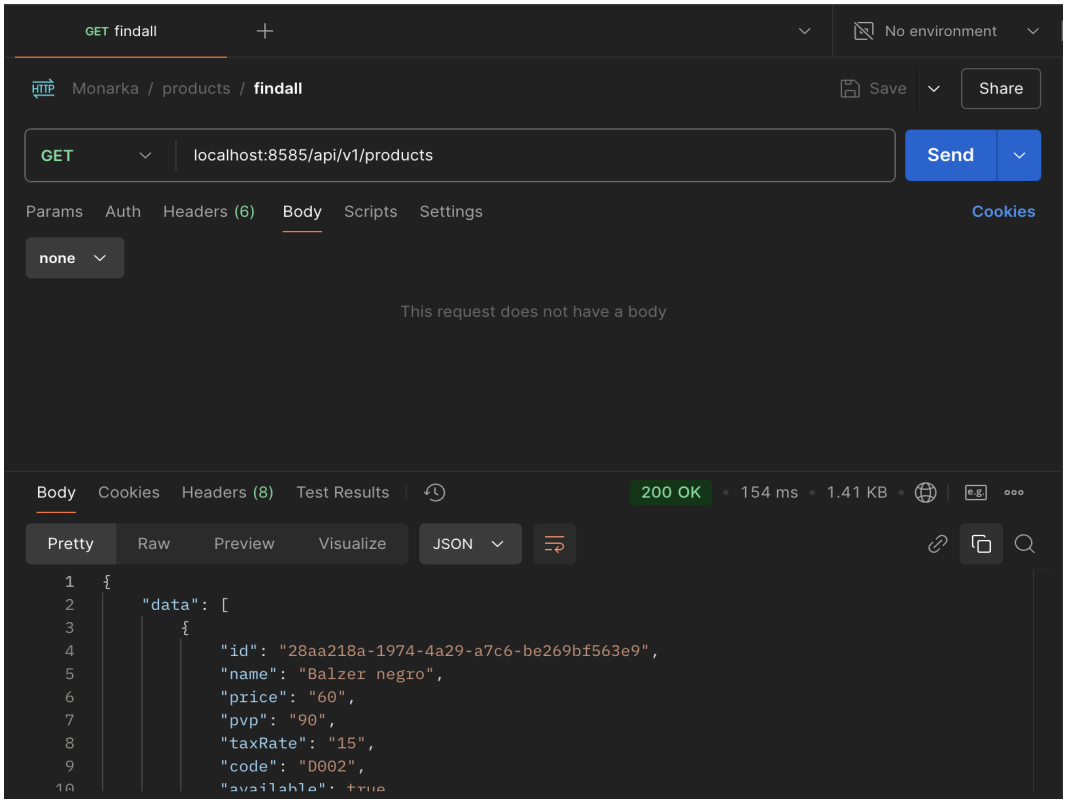


Figura 20. Endpoint de la API con el retorno de datos del producto

6.3.3.3. Validación de Parámetros. Una parte crítica del desarrollo fue la implementación de validaciones en las solicitudes al backend, estas validaciones aseguran que los parámetros enviados cumplan con los formatos esperados, minimizando errores y protegiendo la integridad de la base de datos. Se utilizaron decoradores de validación que permiten especificar reglas como validación UUID, tipo de dato y valores permitidos.

6.3.3.4. Contenedorización de la API. La contenedorización de la API se realizó mediante un **Dockerfile** para asegurar la portabilidad y la facilidad de despliegue en entornos locales y de producción.

6.3.3.4.1. Configuración del Dockerfile. El Dockerfile define los pasos necesarios para construir la imagen de la API, incluyendo la instalación de dependencias, la compilación del código y la exposición de los puertos requeridos. Además, se configuraron variables de entorno para garantizar la conexión segura con la base de datos y otros servicios como Keycloak.

6.3.3.4.2. Despliegue en Kubernetes. La imagen que se generó, se utilizó para implementar la API en Kubernetes, aprovechando herramientas como Argo CD para la gestión del despliegue continuo. Este enfoque garantiza que las actualizaciones de la API se implementen de manera rápida y confiable, reduciendo el tiempo de inactividad.

6.4. Sprint 3 - Desarrollo del Frontend e Integración

El desarrollo del frontend para el proyecto MONARKA se llevó a cabo siguiendo prácticas modernas de desarrollo web y una metodología centrada en el usuario. Este proceso abarcó desde la implementación de componentes reutilizables y la integración con el backend hasta la validación y mejora continua basada en pruebas con usuarios.

6.4.1. Componentes Genéricos. Se priorizó la creación de componentes genéricos para garantizar la consistencia visual y funcional en toda la aplicación, así como para facilitar su mantenimiento y expansión.

6.4.1.1. Botones. Fueron diseñados en variantes específicas como lo son primarios, secundarios y deshabilitados, estos botones incorporan retroalimentación visual al usuario mediante animaciones suaves para indicar estados como hover, clic y deshabilitación. También se configuraron para soportar diferentes tamaños y adaptarse a los estilos definidos en el sistema de diseño.

6.4.1.2. Formularios. Se desarrollaron formularios flexibles con validaciones dinámicas en tiempo real, como la verificación de campos requeridos, formatos válidos (por ejemplo, correos electrónicos y números) y longitudes mínimas o máximas, todo esto gracias a Zod y a React Form Hook. Estos también incluyen mensajes contextuales específicos debajo de cada campo, informando de manera clara los errores detectados.

6.4.1.3. Tarjetas. Fueron diseñadas para presentar datos clave en un formato atractivo y organizado, además, incorporan opciones de personalización como iconos, imágenes y botones de acción que permiten al usuario interactuar directamente con los datos presentados.

Este enfoque de componentes genéricos redujo significativamente la duplicación del código, aceleró los tiempos de desarrollo y permitió una implementación más coherente del diseño en todas las vistas de la aplicación.

6.4.2. Sistema de Diseño con Tailwind CSS. El sistema de diseño se implementó utilizando **Tailwind CSS**, un framework de clases utilitarias que permitió mantener una estética uniforme y personalizable en toda la interfaz.

6.4.2.1. Temas Personalizados. Se definieron colores corporativos alineados con la identidad de MONARKA, y la tipografía y los tamaños de fuentes fueron seleccionados para maximizar la legibilidad y obtener un aspecto profesional.

6.4.2.2. Clases Utilitarias. Las clases utilitarias facilitaron la creación rápida de elementos estilizados sin necesidad de escribir CSS desde cero, además, se aplicaron estilos responsive para optimizar la experiencia en dispositivos móviles y escritorios.

6.4.3. Integración con Backend y Autenticación mediante Axios. Se utilizó Axios como cliente HTTP para gestionar las comunicaciones entre el frontend y el backend. Pues se creó una instancia de Axios personalizada con encabezados predeterminados, con el token JWT para autenticar las solicitudes. En sí, las respuestas exitosas fueron procesadas para extraer datos útiles, y los errores, tanto de red como de servidor, se manejan centralmente, enviando mensajes adecuados al usuario final.

En cuanto al flujo de autenticación permite al usuario iniciar sesión mediante Keycloak, guarda el token JWT en un almacenamiento seguro y realiza solicitudes autenticadas. Por otra parte, se implementaron verificaciones para detectar tokens expirados y redirigir automáticamente a la pantalla de inicio de sesión en caso necesario.

6.4.4. Gestión de Estados de Carga y Manejo de Errores.

6.4.4.1. Indicadores de progreso. Durante la interacción con la API, se implementaron indicadores visuales para mantener informado al usuario, se implementaron **Spinners** que son utilizados durante operaciones como la carga de listas o la creación de nuevos registros. Y mediante **Toasts** se proporciona información sobre el progreso de las acciones solicitadas, como “Producto creado” o “Producto no creado”.

6.4.4.2. Errores Contextualizados. En cuanto a errores de red tendremos mensajes como “Error de conexión. Verifique su red e inténtalo de nuevo”. Cuando exista errores de backend la tarjeta de información dirá “No se pudo guardar el producto. Por favor, verifique los datos ingresados”, y por errores del usuario tendrá alertas claras sobre campos mal completados o faltantes.

6.4.4. Mensajes de Error Claros y Registros Técnicos. Se implementó logs técnicos que capturan detalles de las solicitudes de Endpoints accedidos, Parámetros enviados y Respuestas recibidas, y esto facilitó el diagnóstico y la resolución rápida de problemas en el entorno de producción.

6.4.5. Pruebas de Usuario. Se invitó a usuarios de la empresa MONARKA a interactuar con el sistema, enfocándose en tareas críticas como la creación y consulta de productos, y los problemas encontrados durante estas pruebas fueron documentados detalladamente y fueron clasificados en base a su impacto en la experiencia de usuario. Finalmente los problemas críticos fueron resueltos de manera prioritaria, y las mejoras implementadas fueron validadas por usuarios de la empresa MONARKA, garantizando su efectividad y alineación con sus expectativas.

6.4.6. Producto Obtenido. En las siguiente figura podemos observar el producto obtenido del frontend, con algunas de sus funcionalidades.

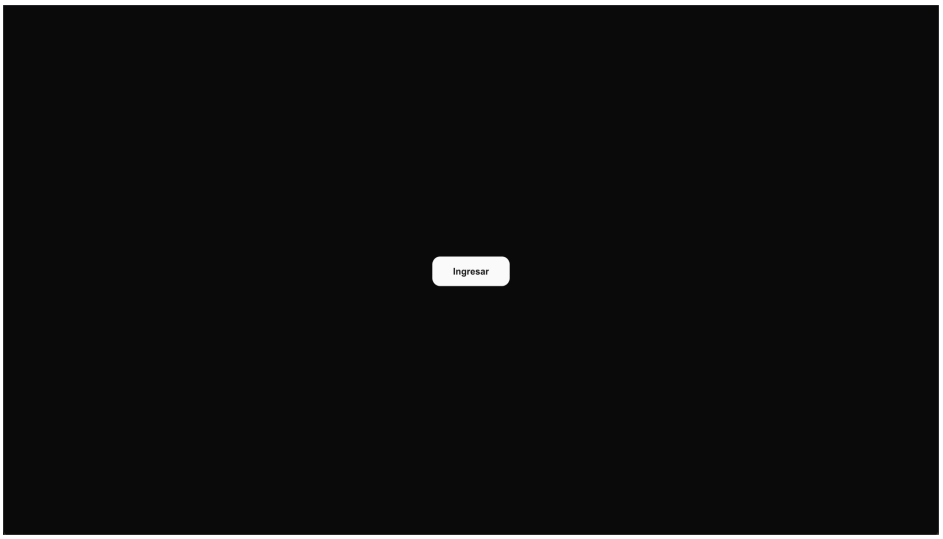


Figura 21. Pantalla de inicio de la aplicación

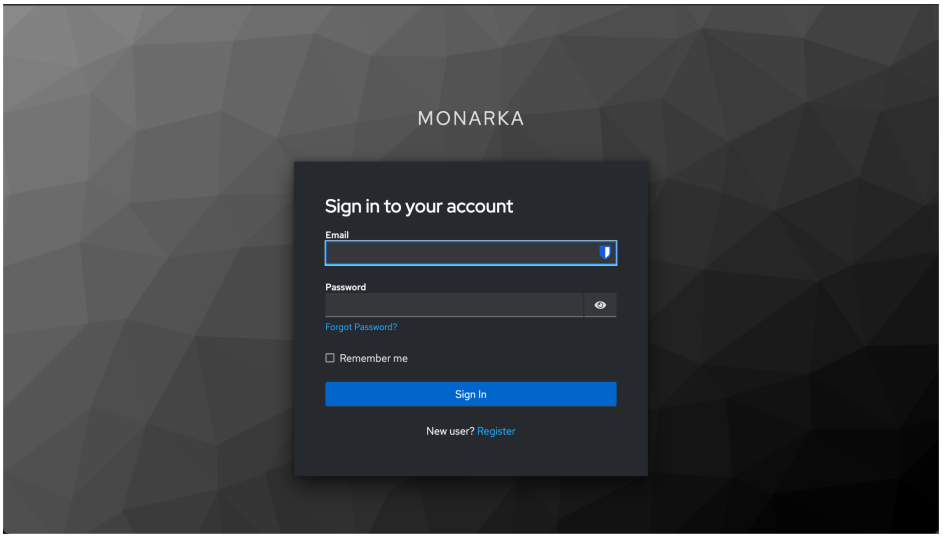


Figura 22. Pantalla de ingreso con Keycloak

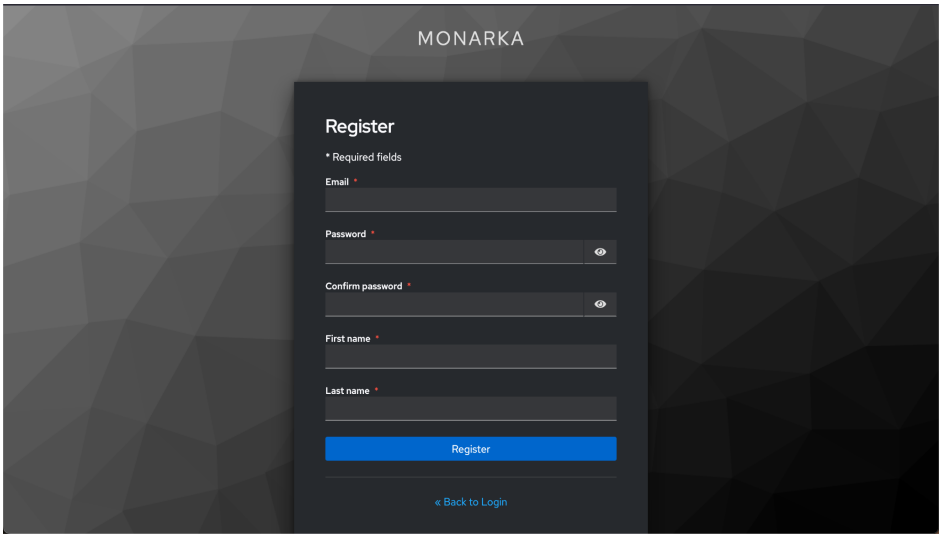


Figura 23. Pantalla de registro con Keycloak

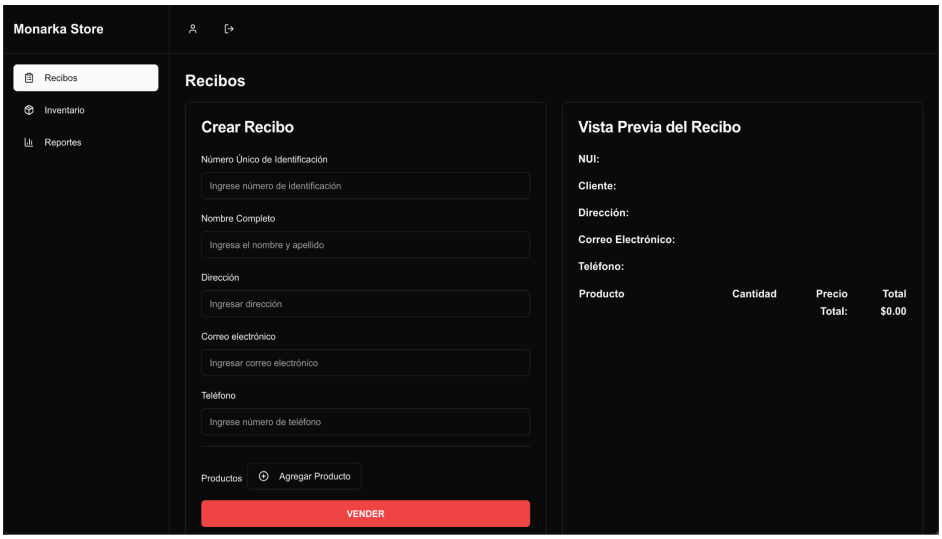


Figura 24. Pantalla principal de la aplicación (hacer recibos)

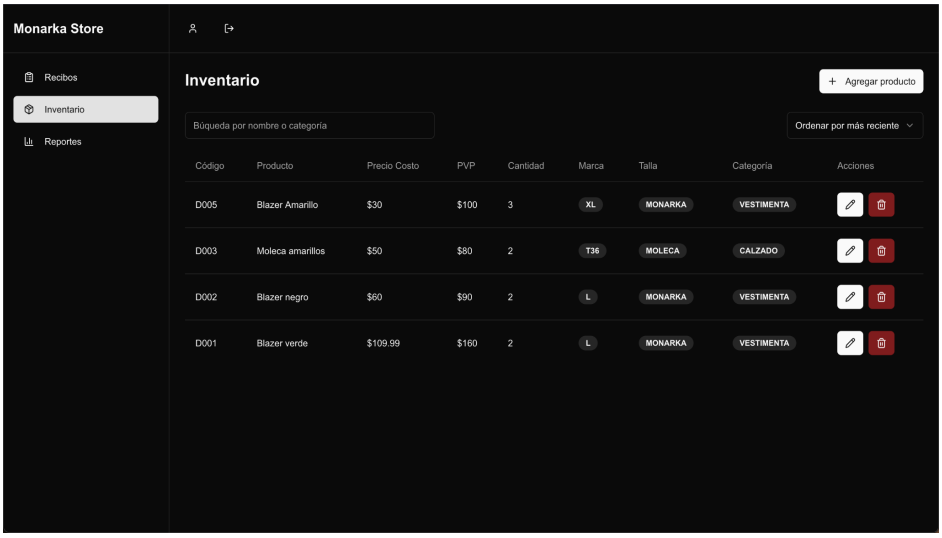


Figura 25. Pantalla de inventario de la aplicación (lista de productos)

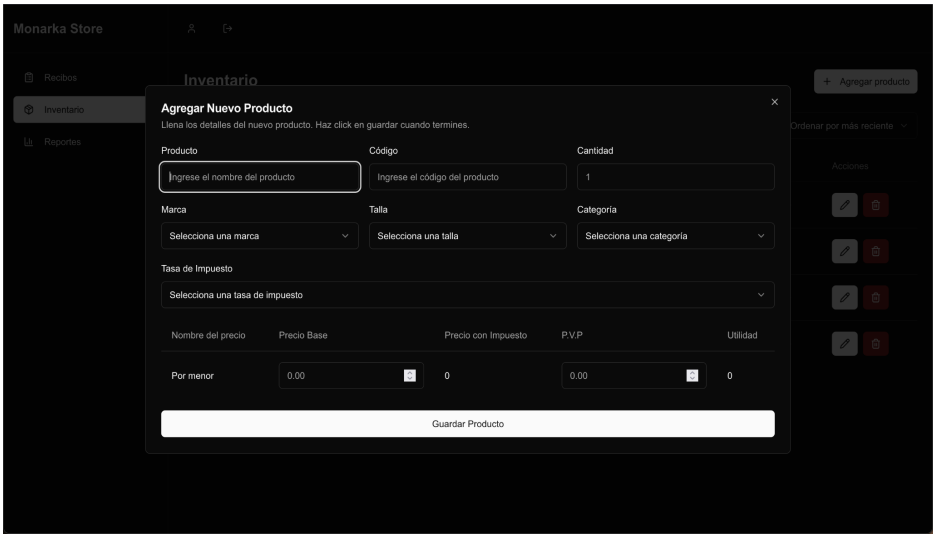


Figura 26. Pantalla de inventario de la aplicación (agregar y editar productos)

6.5. Sprint 4 - Pruebas Finales, Despliegue e Implementación

Este último sprint representó la culminación de los esfuerzos realizados durante el proyecto. El enfoque principal se centró en garantizar que el sistema estuviera completamente funcional, estable, optimizado y listo para su puesta en producción. Asimismo, se implementaron herramientas clave para monitoreo continuo y se validó la satisfacción del **Product Owner** con las funcionalidades desarrolladas. Cada actividad realizada en este sprint fue crucial para asegurar el éxito del despliegue final.

6.5.1. Configuración de JMeter para Pruebas de Rendimiento. Se realizaron pruebas de rendimiento utilizando **JMeter**, una herramienta confiable para evaluar el comportamiento del sistema bajo cargas específicas y garantizar su capacidad para manejar usuarios concurrentes.

Se configuró un escenario de prueba enfocado en la gestión de productos, simulando la interacción de 250 usuarios en un periodo de 10 segundos con un incremento progresivo de un usuario por segundo. Durante estas pruebas, se ejecutaron cuatro tipos de solicitudes a la API: listar productos, crear nuevos productos, actualizar un producto existente y buscar un producto específico. En la Figura 27 se presenta un resumen detallado de la configuración del JMeter. Estas pruebas fueron fundamentales para evaluar el desempeño de la API bajo condiciones de alta carga, replicando escenarios de demanda intensa, como los que podrían ocurrir en días de promociones o eventos especiales.

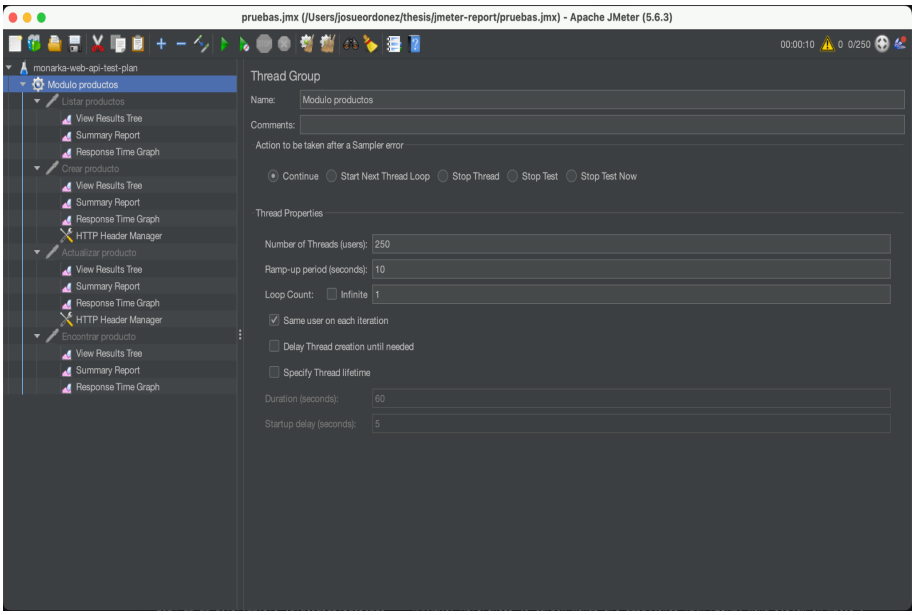


Figura 27. Configuración general del JMeter para las cargas de la API

Se evaluaron métricas clave como los tiempos de respuesta promedio, la tasa de errores y el **throughput**, que representa el número de solicitudes procesadas por segundo. Este análisis

permitió identificar posibles cuellos de botella y áreas de mejora en el sistema. Los resultados mostraron que las operaciones de **listar productos** y **buscar productos** fueron las más rápidas, con tiempos de respuesta promedio de 3.21 KB/s y 4.12 KB/s respectivamente. Por otro lado, las operaciones de **crear productos** y **actualizar productos** tuvieron tiempos de respuesta promedio de 10.78 KB/s y 6.17 KB/s respectivamente. La tasa de errores fue baja, lo que indica un comportamiento estable del sistema bajo carga, y se alcanzó un throughput promedio de 25.1 solicitudes por segundo. En las Figuras 28, 29, 30 y 31 se presentan los resultados detallados de estas pruebas realizadas con JMeter.

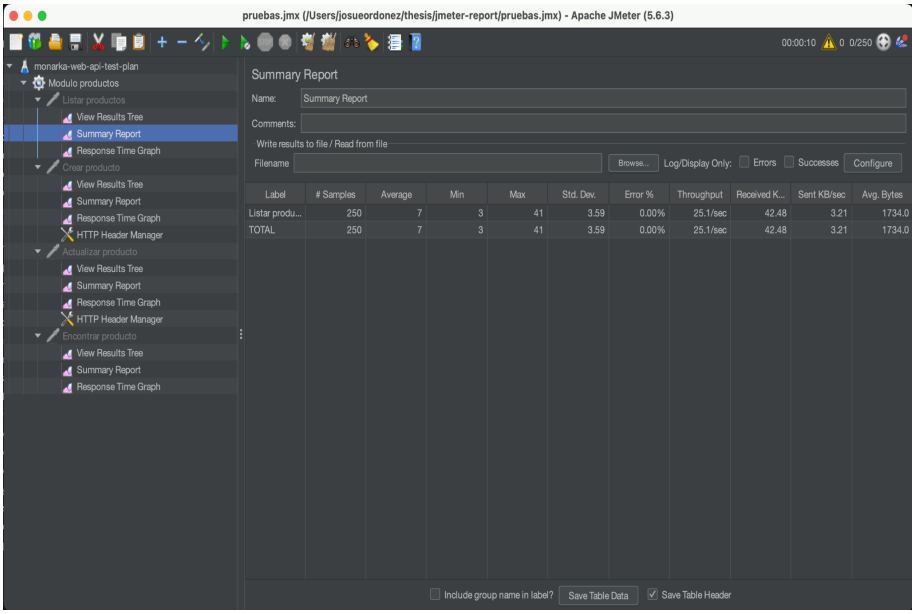


Figura 28. Reporte resumen de listar productos de la API en JMeter

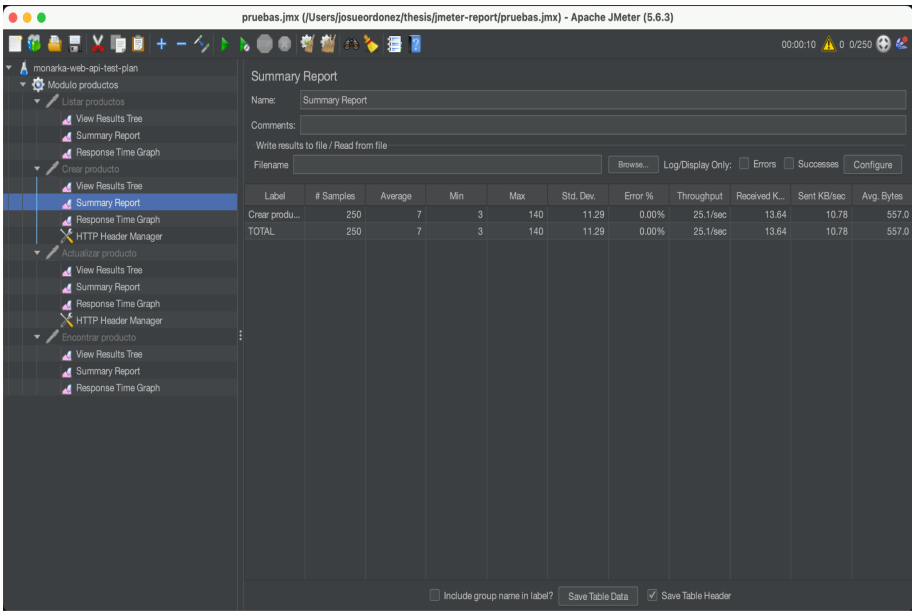


Figura 29. Reporte resumen de crear productos de la API en JMeter

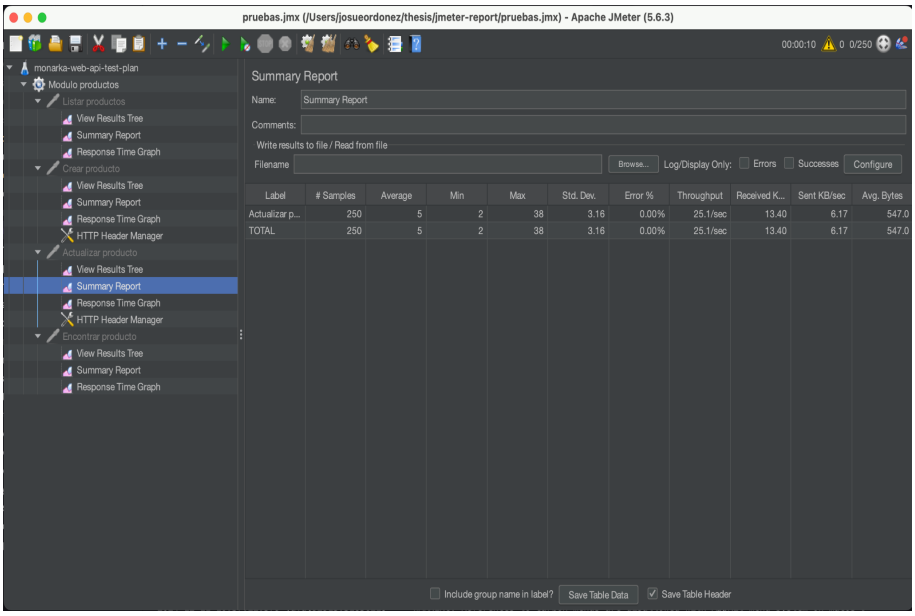


Figura 30. Reporte resumen de actualizar productos de la API en JMeter

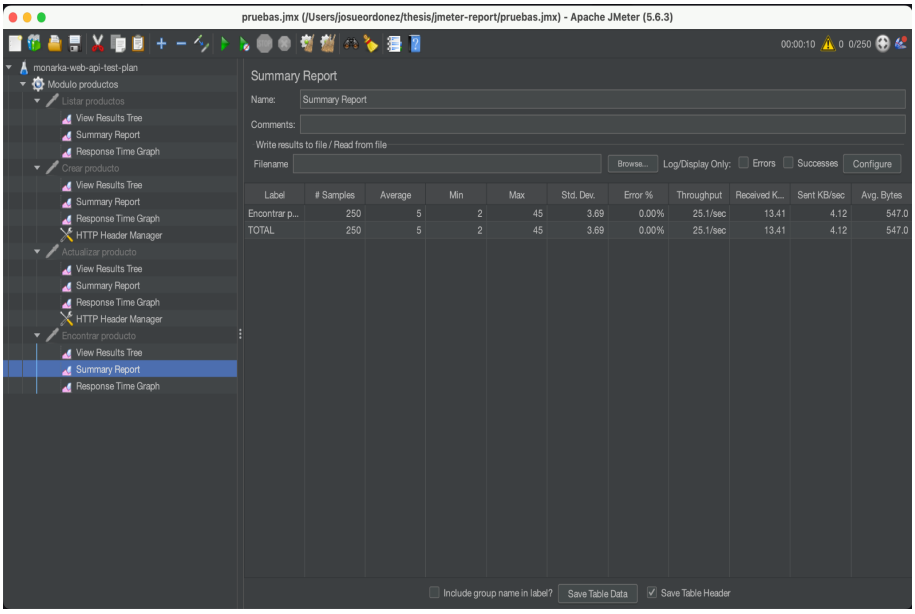
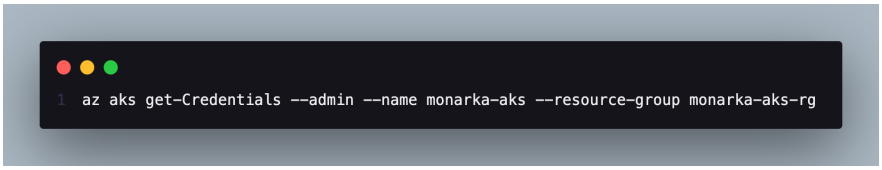


Figura 31. Reporte resumen de encontrar producto de la API en JMeter

6.5.2. Configuración de Argo CD para despliegues Automatizados. Como se especificó en el Sprint 1, **Argo CD** fue configurado como el sistema principal de gestión de despliegues. Esta herramienta permite una implementación continua eficiente y asegura la sincronización constante entre los repositorios Git y el clúster de producción desplegado en **AKS (Azure Kubernetes Service)**. Para exponer el AKS en la computadora, se utilizó el comando mostrado en la Figura 32, que permitió establecer la conexión necesaria para interactuar con el clúster desde el entorno local.

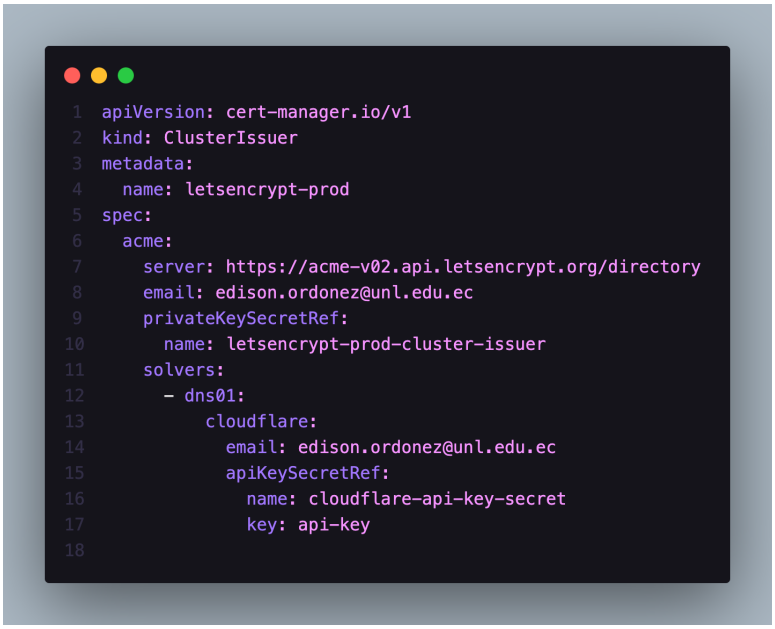
Cada vez que se hace un commit en el repositorio ArgoCD sincroniza automáticamente el cluster de Kubernetes, aplicando las actualizaciones necesarias sin interrupciones.



```
az aks get-Credentials --admin --name monarka-aks --resource-group monarka-aks-rg
```

Figura 32. Comando para conectar el AKS con la computadora

Una vez creado el namespace y configurado en **Argo CD**, el servicio se expuso en la URL http://localhost:8080 para su acceso. Para garantizar un enrutamiento eficiente y la gestión de certificados SSL, fue necesario instalar y configurar los servicios de **Ingress-NGINX** y **Cert-Manager**. La configuración utilizada para esta implementación se detalla en la Figura 33, asegurando que el sistema estuviera preparado para manejar tráfico con certificados válidos y rutas correctamente definidas.

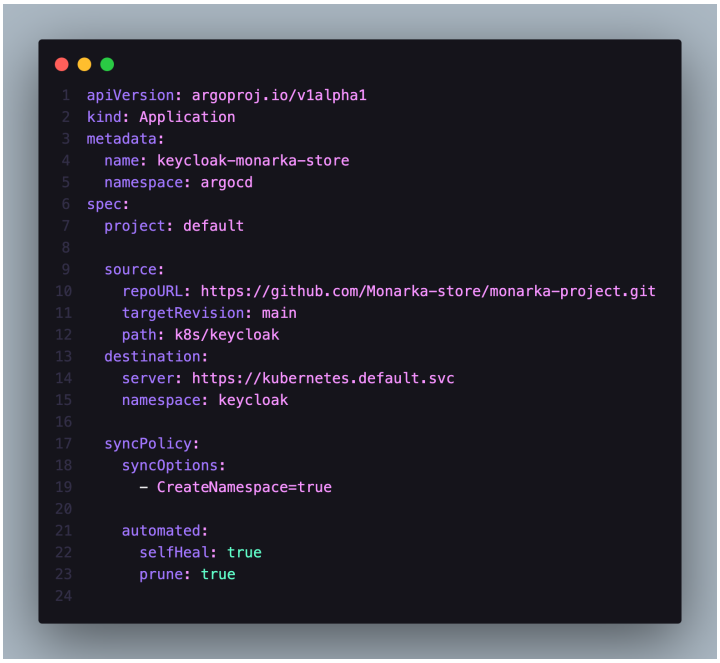


```yaml
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: edison.ordonez@unl.edu.ec
    privateKeySecretRef:
      name: letsencrypt-prod-cluster-issuer
    solvers:
      - dns01:
          cloudflare:
            email: edison.ordonez@unl.edu.ec
            apiKeySecretRef:
              name: cloudflare-api-key-secret
              key: api-key

```

Figura 33. Manifiesto de la Certificación como dns de Cloudflare

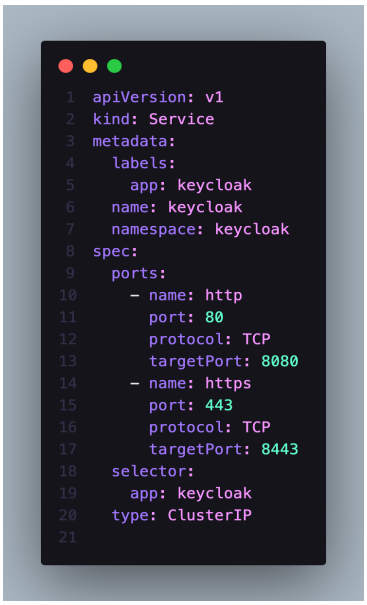
6.5.3. Manifiestos de Kubernetes. Se configuraron namespaces y secretos específicos para cada servicio del sistema, garantizando un aislamiento adecuado y una gestión segura de la información. Los servicios desplegados en Kubernetes incluyen **Keycloak**, la **API** y el **Cliente**, cada uno con su configuración individualizada. Además, se estableció la configuración de exposición en **Argo CD** para gestionar los despliegues de manera eficiente. Dado que el repositorio Git utilizado es privado, se configuraron las credenciales necesarias en Argo CD para asegurar una integración fluida y segura con el repositorio.

6.5.3.1. Manifiesto de Keycloak. Este servicio fue configurado con su propio namespace, junto con los recursos fundamentales como **Ingress**, **Deployment**, **Service**, y su respectiva integración en **Argo CD** para gestionar los despliegues de manera eficiente. Las configuraciones detalladas y los manifiestos correspondientes a **Keycloak** se presentan en las Figuras 34, 35, 36 y 37, ilustrando cómo se estructuró cada componente para garantizar su correcto funcionamiento dentro del clúster de Kubernetes.



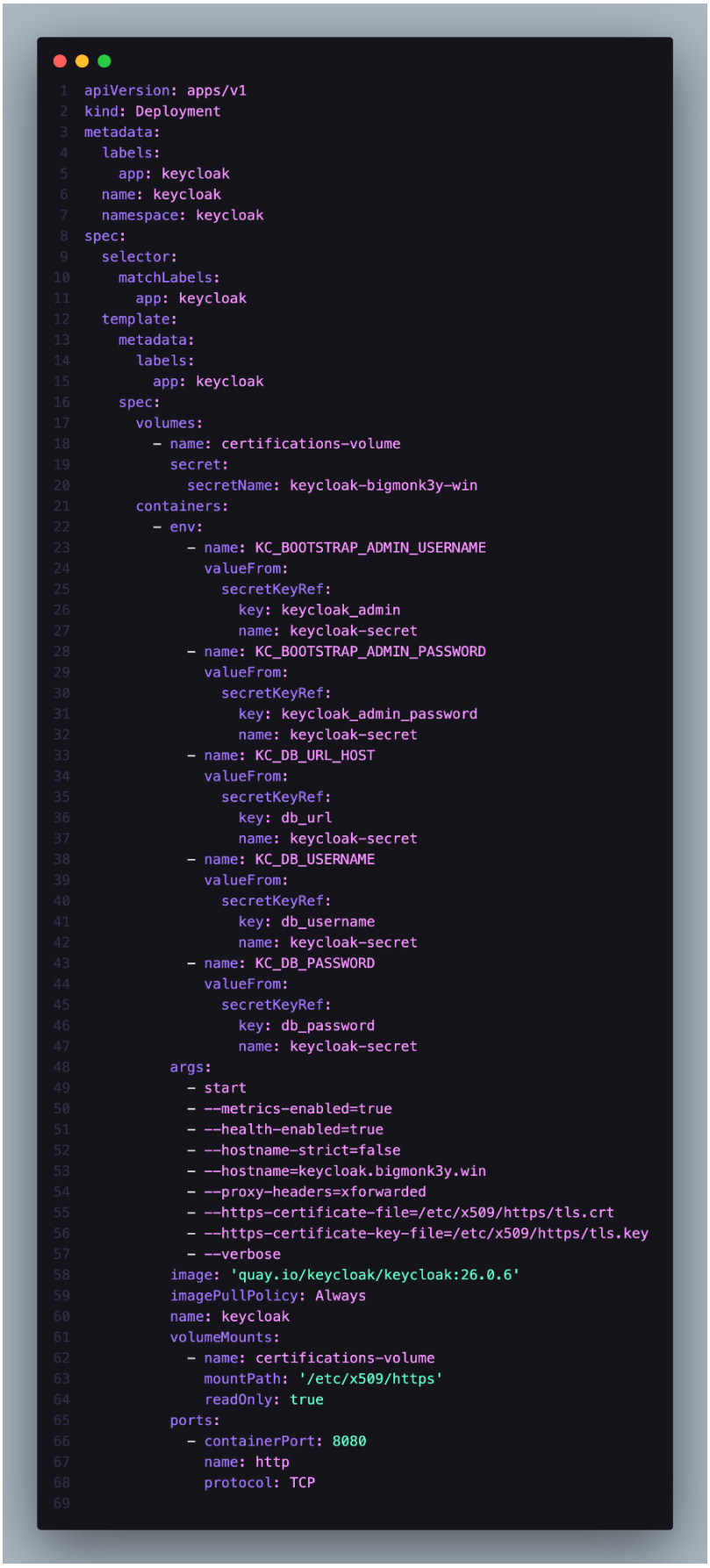
```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: keycloak-monarka-store
  namespace: argocd
spec:
  project: default

  source:
    repoURL: https://github.com/Monarka-store/monarka-project.git
    targetRevision: main
    path: k8s/keycloak
  destination:
    server: https://kubernetes.default.svc
    namespace: keycloak

  syncPolicy:
    syncOptions:
      - CreateNamespace=true

    automated:
      selfHeal: true
      prune: true
```

Figura 34. Manifiesto de la configuración del Argo CD del servicio de Keycloak



```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: keycloak
  name: keycloak
  namespace: keycloak
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 8080
    - name: https
      port: 443
      protocol: TCP
      targetPort: 8443
  selector:
    app: keycloak
  type: ClusterIP
```

Figura 35. Manifiesto del servicio de Keycloak



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: keycloak
  name: keycloak
  namespace: keycloak
spec:
  selector:
    matchLabels:
      app: keycloak
  template:
    metadata:
      labels:
        app: keycloak
    spec:
      volumes:
        - name: certifications-volume
          secret:
            secretName: keycloak-bigmonk3y-win
      containers:
        - env:
            - name: KC_BOOTSTRAP_ADMIN_USERNAME
              valueFrom:
                secretKeyRef:
                  key: keycloak_admin
                  name: keycloak-secret
            - name: KC_BOOTSTRAP_ADMIN_PASSWORD
              valueFrom:
                secretKeyRef:
                  key: keycloak_admin_password
                  name: keycloak-secret
            - name: KC_DB_URL_HOST
              valueFrom:
                secretKeyRef:
                  key: db_url
                  name: keycloak-secret
            - name: KC_DB_USERNAME
              valueFrom:
                secretKeyRef:
                  key: db_username
                  name: keycloak-secret
            - name: KC_DB_PASSWORD
              valueFrom:
                secretKeyRef:
                  key: db_password
                  name: keycloak-secret
          args:
            - start
            - --metrics-enabled=true
            - --health-enabled=true
            - --hostname-strict=false
            - --hostname=keycloak.bigmonk3y.win
            - --proxy-headers=xforwarded
            - --https-certificate-file=/etc/x509/https/tls.crt
            - --https-certificate-key-file=/etc/x509/https/tls.key
            - --verbose
          image: 'quay.io/keycloak/keycloak:26.0.6'
          imagePullPolicy: Always
          name: keycloak
          volumeMounts:
            - name: certifications-volume
              mountPath: '/etc/x509/https'
              readOnly: true
          ports:
            - containerPort: 8080
              name: http
              protocol: TCP

```

Figura 36. Manifiesto del despliegue de Keycloak

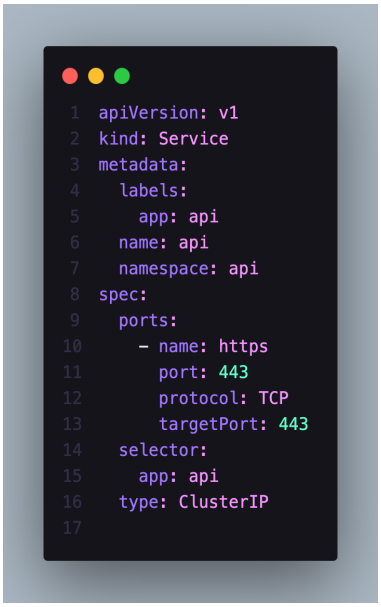
```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  labels:
    ingress-controller: nginx
  name: keycloak-ingress
  namespace: keycloak
  annotations:
    cert-manager.io/cluster-issuer: 'letsencrypt-prod'
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/ssl-redirect: 'true'
    nginx.ingress.kubernetes.io/ssl-passthrough: 'true'
    nginx.ingress.kubernetes.io/backend-protocol: 'HTTPS'
spec:
  ingressClassName: nginx
  rules:
    - host: keycloak.bigmonk3y.win
      http:
        paths:
          - backend:
              service:
                name: keycloak
                port:
                  number: 443
            path: /
            pathType: ImplementationSpecific
  tls:
    - hosts:
        - keycloak.bigmonk3y.win
      secretName: keycloak-bigmonk3y-win
```

Figura 37. Manifiesto del ingreso de Keycloak

6.5.3.2. Manifiesto de la API. De manera similar al servicio anterior, este también cuenta con su propio namespace y recursos esenciales como **Ingress**, **Deployment**, **Service**, además de la configuración necesaria en **Argo CD** para su gestión y despliegue. En las Figuras 38, 39, 40 y 41 se presentan los manifiestos correspondientes a la **API**, detallando cómo se configuraron sus componentes para integrarse de forma eficiente en el clúster de Kubernetes.

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: api-monarka-store
  namespace: argocd
spec:
  project: default

  source:
    repoURL: https://github.com/Monarka-store/monarka-project.git
    targetRevision: main
    path: k8s/api
  destination:
    server: https://kubernetes.default.svc
    namespace: api

  syncPolicy:
    syncOptions:
      - CreateNamespace=true

    automated:
      selfHeal: true
      prune: true
```

Figura 38. Manifiesto de la configuración del Argo CD del servicio del API



```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: api
  name: api
  namespace: api
spec:
  ports:
    - name: https
      port: 443
      protocol: TCP
      targetPort: 443
  selector:
    app: api
  type: ClusterIP
```

Figura 39. Manifiesto del servicio de la API

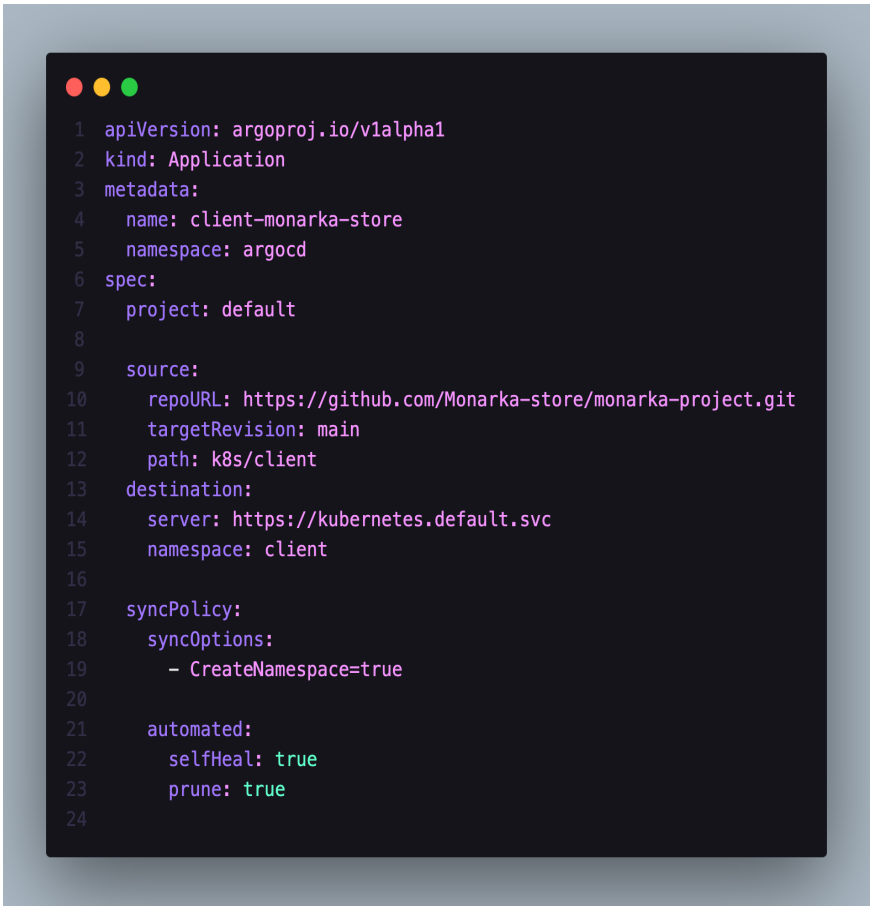
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: api
  name: api
  namespace: api
spec:
  selector:
    matchLabels:
      app: api
  template:
    metadata:
      labels:
        app: api
    spec:
      volumes:
        - name: certifications-volume
          secret:
            secretName: api-bigmonk3y-win
      imagePullSecrets:
        - name: acr-secret
      containers:
        - name: api
          imagePullPolicy: Always
          image: monarkastoreapirn.azurecr.io/monarka-api:latest
          env:
            - name: PORT
              value: '443'
            - name: DATABASE_URL
              valueFrom:
                secretKeyRef:
                  key: db_url
                  name: api-secret
            - name: JWT_SECRET
              valueFrom:
                secretKeyRef:
                  key: jwt_secret
                  name: api-secret
            - name: KEYCLOAK_JWT_SECRET
              valueFrom:
                secretKeyRef:
                  key: keycloak_jwt_secret
                  name: api-secret
            - name: KEYCLOAK_CLIENT_ID_API
              valueFrom:
                secretKeyRef:
                  key: keycloak_client_id_api
                  name: api-secret
            - name: KEYCLOAK_CLIENT_SECRET_API
              valueFrom:
                secretKeyRef:
                  key: keycloak_client_secret_api
                  name: api-secret
            - name: KEYCLOAK_ISSUER_API
              valueFrom:
                secretKeyRef:
                  key: keycloak_issuer_api
                  name: api-secret
            - name: KEYCLOAK_GRANT_TYPE
              valueFrom:
                secretKeyRef:
                  key: keycloak_grant_type
                  name: api-secret
          volumeMounts:
            - name: certifications-volume
              mountPath: '/etc/ssl/certs'
              readOnly: true
          ports:
            - name: https
              containerPort: 443
              protocol: TCP

```

Figura 40. Manifiesto del despliegue de la API

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  labels:
    ingress-controller: nginx
  name: api-ingress
  namespace: api
  annotations:
    cert-manager.io/cluster-issuer: 'letsencrypt-prod'
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/ssl-passthrough: 'true'
    nginx.ingress.kubernetes.io/ssl-redirect: 'true'
    nginx.ingress.kubernetes.io/backend-protocol: 'HTTPS'
spec:
  ingressClassName: nginx
  rules:
    - host: api.bigmonk3y.win
      http:
        paths:
          - backend:
              service:
                name: api
                port:
                  number: 443
            path: /
            pathType: ImplementationSpecific
  tls:
    - hosts:
        - api.bigmonk3y.win
      secretName: api-bigmonk3y-win
```

Figura 41. Manifiesto del ingreso de la API

6.5.3.3. Manifiesto del Client. Finalmente, este servicio, al igual que los dos anteriores, está configurado con su propio **namespace**, así como con los recursos fundamentales: **Ingress**, **Deployment**, y **Service**, integrados con la configuración esencial de **Argo CD** para garantizar un despliegue eficiente y controlado. En las Figuras 42, 43, 44 y 45 se presentan los manifiestos correspondientes al **Client**, mostrando en detalle cómo se estructuraron sus componentes dentro del clúster de Kubernetes.



```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: client-monarka-store
  namespace: argocd
spec:
  project: default

  source:
    repoURL: https://github.com/Monarka-store/monarka-project.git
    targetRevision: main
    path: k8s/client
  destination:
    server: https://kubernetes.default.svc
    namespace: client

  syncPolicy:
    syncOptions:
      - CreateNamespace=true

    automated:
      selfHeal: true
      prune: true

```

Figura 42. Manifiesto de la configuración del Argo CD del servicio del Client

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: client
  name: client
  namespace: client
spec:
  ports:
    - name: https
      port: 443
      protocol: TCP
      targetPort: 3000
  selector:
    app: client
  type: ClusterIP

```

Figura 43. Manifiesto del servicio del Client

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: client
  name: client
  namespace: client
spec:
  selector:
    matchLabels:
      app: client
  template:
    metadata:
      labels:
        app: client
    spec:
      volumes:
        - name: certifications-volume
          secret:
            secretName: www-bigmonk3y-win
      imagePullSecrets:
        - name: acr-secret
      containers:
        - name: client
          imagePullPolicy: Always
          image: monarkastoreapirn.azurecr.io/monarka-client:latest
          command: ['pnpm']
          args:
            [
              'dev',
              '--experimental-https',
              '--experimental-https-cert',
              '/etc/ssl/certs/tls.crt',
              '--experimental-https-key',
              '/etc/ssl/certs/tls.key',
            ]
          env:
            - name: KEYCLOAK_CLIENT_ID
              valueFrom:
                secretKeyRef:
                  key: keycloak_client_id
                  name: client-secret
            - name: KEYCLOAK_CLIENT_SECRET
              valueFrom:
                secretKeyRef:
                  key: keycloak_client_secret
                  name: client-secret
            - name: KEYCLOAK_ISSUER
              valueFrom:
                secretKeyRef:
                  key: keycloak_issuer
                  name: client-secret
            - name: NEXTAUTH_URL
              valueFrom:
                secretKeyRef:
                  key: nextauth_url
                  name: client-secret
            - name: NEXTAUTH_SECRET
              valueFrom:
                secretKeyRef:
                  key: nextauth_secret
                  name: client-secret
            - name: NEXT_PUBLIC_API_URL
              valueFrom:
                secretKeyRef:
                  key: next_public_api_url
                  name: client-secret
          volumeMounts:
            - name: certifications-volume
              mountPath: '/etc/ssl/certs'
              readOnly: true
          ports:
            - name: https
              containerPort: 3000
              protocol: TCP

```

Figura 44. Manifiesto del despliegue del Client

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  labels:
    ingress-controller: nginx
  name: client-ingress
  namespace: client
  annotations:
    cert-manager.io/cluster-issuer: 'letsencrypt-prod'
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/ssl-passthrough: 'true'
    nginx.ingress.kubernetes.io/ssl-redirect: 'true'
    nginx.ingress.kubernetes.io/backend-protocol: 'HTTPS'
spec:
  ingressClassName: nginx
  rules:
    - host: www.bigmonk3y.win
      http:
        paths:
          - backend:
              service:
                name: client
                port:
                  number: 443
            path: /
            pathType: ImplementationSpecific
  tls:
    - hosts:
        - www.bigmonk3y.win
      secretName: www-bigmonk3y-win
```

Figura 45. Manifiesto del ingreso del Client

6.5.3. Implementación de Prometheus y Grafana para Monitoreo. Para poder revisar el desempeño del sistema en tiempo real, se implementó **Prometheus** y **Grafana** como herramientas principales de monitoreo en el clúster de Kubernetes.

Estas herramientas fueron configuradas directamente en el AKS (Azure Kubernetes Service), estas se configuraron para capturar métricas de todos los contenedores en el clúster, como el uso de CPU, memoria, tráfico de red y tiempos de respuesta a la aplicación.

Se configuraron alertas para eventos críticos en el monitoreo como son el consumo excesivo de recursos, caídas de servicios y largos tiempos de respuestas, estas serán enviadas mediante correo electrónico.

6.5.3.1. Visualización en Grafana. Se diseñaron dashboards personalizados en **Grafana** para facilitar el monitoreo, estos dashboards permitieron observar métricas clave de manera clara e intuitiva, ayudando a identificar y solucionar problemas rápidamente. Podemos observar algunas gráficas del clúster en la Figuras 46, 47, 48, 49 y 50.

Figura 46. Gráficas del clúster llamado monarka-aks de Azure Kubernetes Service

Nota:Se observa algunos parámetros como Uso de la CPU, Cuota de la CPU, Memoria, Peticiones de la Memoria.

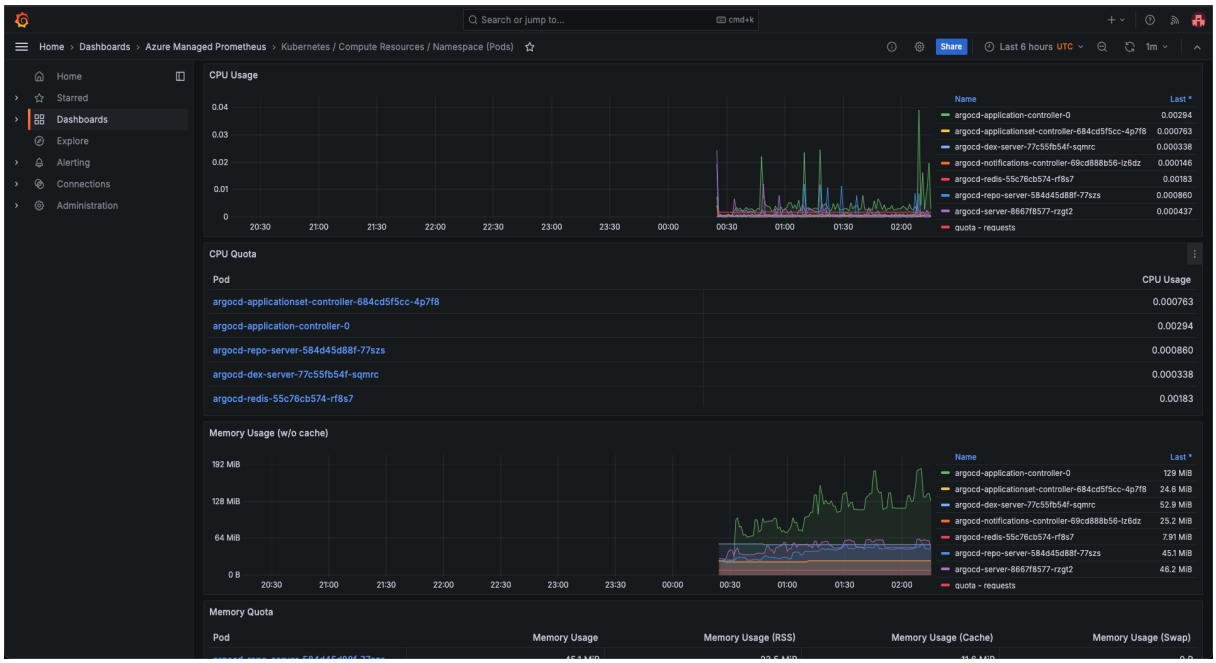


Figura 47. Gráfica en Grafana por el servicio de Argo CD

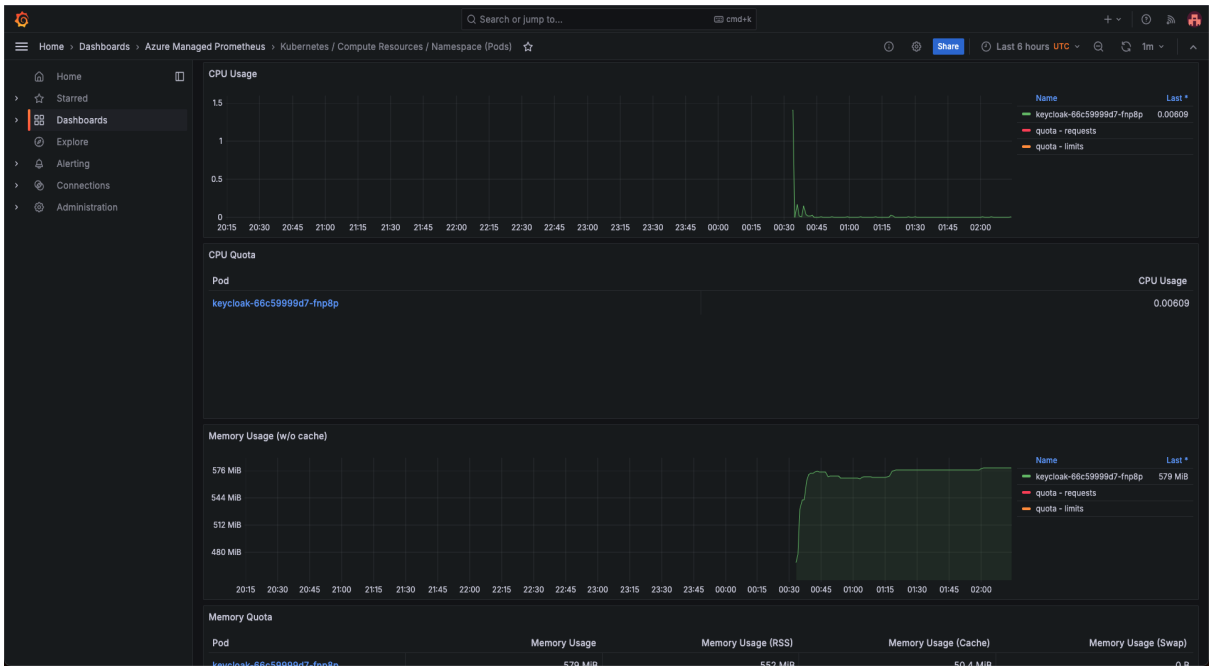


Figura 48. Gráfica en Grafana por el servicio de Keycloak

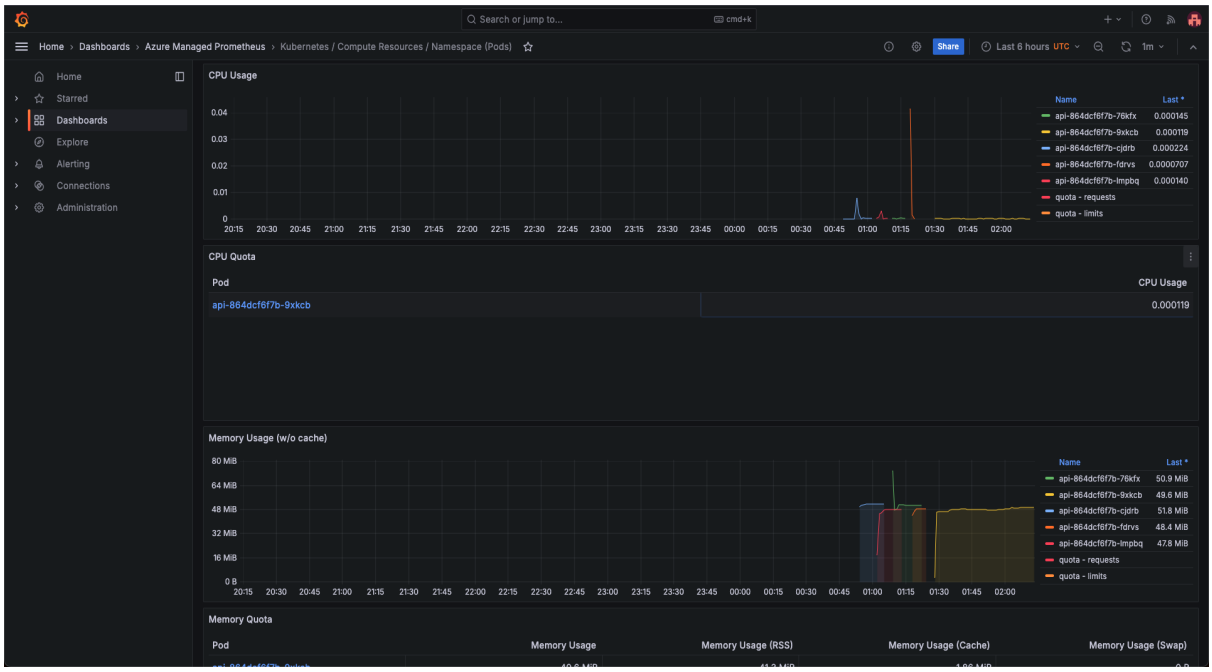


Figura 49. Gráfica en Grafana por el servicio de la API

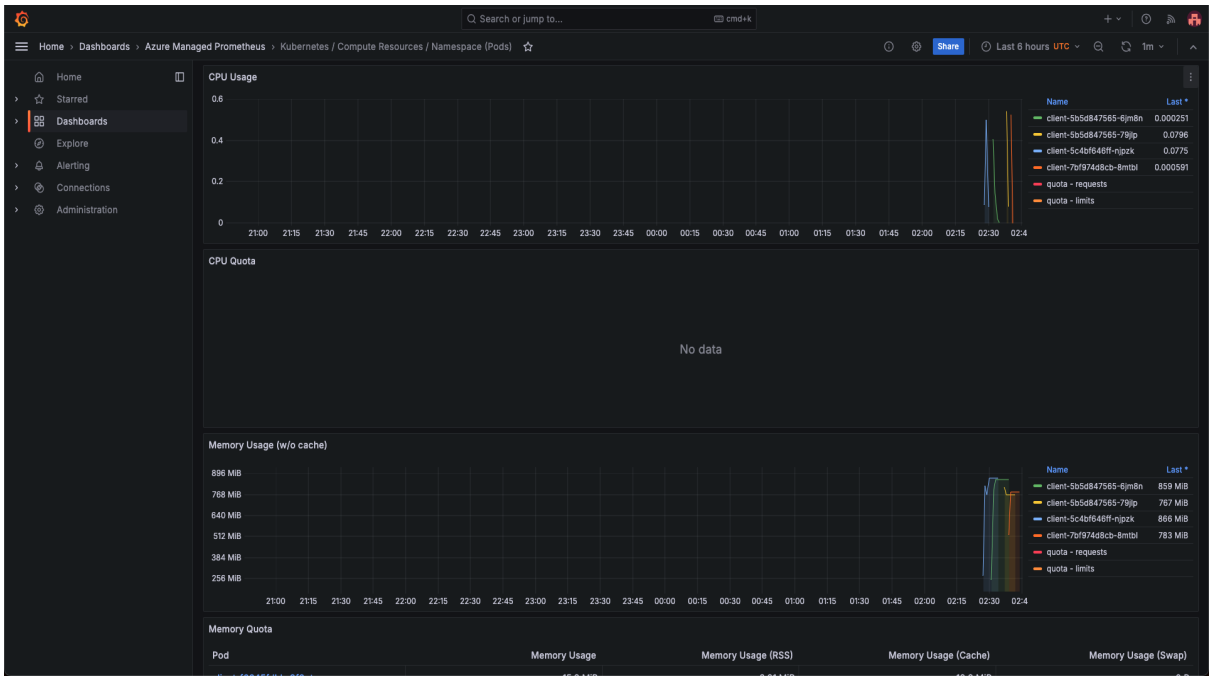


Figura 50. Gráfica en Grafana por el servicio de Client

7. Discusión

La migración de la aplicación de escritorio a un sistema web para la empresa MONARKA, junto con la implementación de un pipeline de CI/CD, representó un avance significativo en la optimización de procesos y la entrega de valor tecnológico.

La integración de GitHub Actions permitió automatizar tareas de construcción, prueba y despliegue, reduciendo el tiempo de entrega y los errores humanos. Los resultados demostraron que el pipeline de CI/CD logró cumplir con el objetivo de estandarizar y acelerar las operaciones, facilitando despliegues en Azure Kubernetes Service (AKS) y eliminando interrupciones. Esto fue clave para garantizar la estabilidad del sistema en producción.

La combinación de tecnologías como NestJS para la API, Next.js para el cliente y Keycloak para la gestión de identidad proporcionó un sistema flexible y seguro. Esta arquitectura, implementada en Kubernetes con Argo CD, permitió un control centralizado de despliegues y configuraciones. La decisión de utilizar estas herramientas resultó acertada, ya que se logró una infraestructura capaz de soportar futuras expansiones, como la incorporación de más módulos.

El nuevo sistema ofrece mejoras significativas en los procesos internos de MONARKA, como la gestión del inventario de calzado, accesorios y ropa, y en la facturación. Las pruebas de funcionalidad mostraron una mayor eficiencia en la actualización de stock y la generación de reportes. Esto optimiza el flujo de trabajo, facilitando a los usuarios tareas que antes eran manuales y propensas a errores.

Aunque Prometheus y Grafana se integraron parcialmente en el entorno, la configuración inicial permitió identificar cuellos de botella en las operaciones del sistema. Estos indicadores proporcionan información valiosa para ajustar el rendimiento y garantizar la disponibilidad del servicio.

8. Conclusiones

- La implementación de un pipeline CI/CD basado en GitHub Actions y desplegado en AKS permitió automatizar procesos clave en el ciclo de vida del desarrollo, logrando una entrega continua eficiente y confiable.
- La arquitectura basada en servicios independientes, soportada por tecnologías como NestJS, Next.js y Keycloak, cumplió con los requerimientos funcionales y no funcionales, brindando un sistema escalable, seguro y modular para MONARKA.
- La integración de Kubernetes y Argo CD optimizó la gestión de despliegues, reduciendo tiempos y errores durante las actualizaciones del sistema.
- La migración al sistema web modernizó y mejoró los procesos de gestión de inventarios y facturación, alineándose con los objetivos del proyecto y aportando valor directo al negocio.
- Las herramientas de monitoreo implementadas permitieron una mayor visibilidad sobre el rendimiento del sistema, sentando las bases para un mantenimiento proactivo.

9. Recomendaciones

- Completar la integración de Prometheus y Grafana para monitorear métricas clave del sistema en tiempo real, mejorando la capacidad de respuesta ante incidentes y optimizando recursos.
- Proveer al equipo de MONARKA capacitaciones periódicas sobre el uso del pipeline CI/CD y las herramientas asociadas, asegurando un manejo eficiente y autónomo del sistema.
- Incorporar un conjunto más robusto de pruebas automatizadas en el pipeline CI/CD, incluyendo pruebas de integración y end-to-end, para garantizar la calidad del sistema en cada despliegue.
- Evaluar la posibilidad de extender el sistema a otras áreas del negocio, como la gestión de clientes y análisis de ventas, aprovechando la arquitectura flexible y modular implementada.
- Realizar evaluaciones regulares del rendimiento del sistema y del pipeline CI/CD para identificar oportunidades de mejora y actualizar herramientas según las necesidades del negocio.
- Explorar el uso de inteligencia artificial para análisis predictivo en inventarios o personalización de ofertas, maximizando el valor comercial de la plataforma.

10. Bibliografia

[1] M. Pérez and S. Pico, “El principio de simplicidad administrativa aplicado a través del sistema de facturación,” *Revista de Investigación Aplicada en Ciencias Empresariales*, vol. 9, no. 1, pp. 118–131, 2020.

[2] M. Guzmán Espinoza, “Propuesta de mejora del proceso de facturación de una empresa corporativa del sector retail,” 2023.

[3] I. Carballo, P. Garnero, A. Penedo, and J. Monje, *Expansión de herramientas financieras digitales para impulsar el comercio electrónico de las MiPyMEs de América Latina*. Banco Interamericano de Desarrollo, 2021.

[4] A. Zalazar, S. Gonnet, and H. Leone, “Migración de sistemas heredados a cloud computing,” in *XLIII Jornadas Argentinas de Informática e Investigación Operativa (43JAIIO)-XV Simposio Argentino de Ingeniería de Software (Buenos Aires, 2014)*, 2014.

[5] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, “DevOps,” *IEEE Software*, vol. 33, no. 3, pp. 94-100, May-June 2016, doi: 10.1109/MS.2016.68.

[6] Sabo, M. (2020). *NestJS* (Doctoral dissertation, Josip Juraj Strossmayer University of Osijek. Department of Mathematics. Chair of Applied Mathematics. Computer Science Research Group).

[7] G. Bierman, M. Abadi, and M. Torgersen, “Understanding TypeScript,” *ECOOP 2014 – Object-Oriented Programming*, pp. 257–281, 2014, doi: https://doi.org/10.1007/978-3-662-44202-9_11.

[8] M. Riva, *Real-World Next.js: Build Scalable, High-Performance, and Modern Web Applications Using Next.js, the React Framework for Production*. Packt Publishing Ltd., 2022.

[9] A. Fedosejev, *React.js Essentials*. Packt Publishing Ltd., 2015.

[10] H. Jartarghar, G. Salanke, A. AR, G. Sharvani, and S. Dalali, “React apps with server-side rendering: Next.js,” *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 14, no. 4, pp. 25-29, 2022.

[11] S. Thorgersen and P. Silva, *Keycloak - Identity and Access Management for Modern Applications: Harness the Power of Keycloak, OpenID Connect, and OAuth 2.0 Protocols to Secure Applications*. Packt Publishing Ltd., 2021.

[12] T. Kinsman, M. Wessel, M. Gerosa, and C. Treude, “How do software developers use GitHub Actions to automate their workflows?” in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, May 2021, pp. 420-431.

[13] C. Boettiger, “An introduction to Docker for reproducible research,” *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71-79, 2015.

[14] J. Park, S. An, D. Youn, G. Kim, y S. Ryu, “Jest: N+1-version differential testing of both JavaScript engines and specification,” en *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, mayo 2021, pp. 13–24.

[15] D. Nevedrov, “Using JMeter to performance test web services,” published on *dev2dev*, pp. 1-11, 2006.

[16] M. Deploy, S. Containerized, and S. Ifrah, *Getting Started with Containers in Azure*., 2024

[17] B. Yuen, A. Matyushentsev, J. Suen, and T. Ekenstam, *GitOps and Kubernetes: Continuous Deployment with Argo CD, Jenkins X, and Flux*. Simon and Schuster, 2021.

[18] M. Luksa, *Kubernetes in Action*. Simon and Schuster, 2017.

[19] H. Chawla, H. Kathuria, H. Chawla, and H. Kathuria, “Azure Kubernetes Service,” in *Building Microservices Applications on Microsoft Azure: Designing, Developing, Deploying, and Monitoring*, pp. 151-177, 2019.

[20] J. Turnbull, *Monitoring with Prometheus*. Turnbull Press, 2018.

[21] M. Chakraborty and A. Kundan, “Grafana,” in *Monitoring Cloud-Native Applications: Lead Agile Operations Confidently Using Open Source Software*, pp. 187-240, Berkeley, CA: Apress, 2021.

[22] K. Schwaber and J. Sutherland, *La guía de Scrum*. Scrumguides.org, 2013, pp. 1-21.

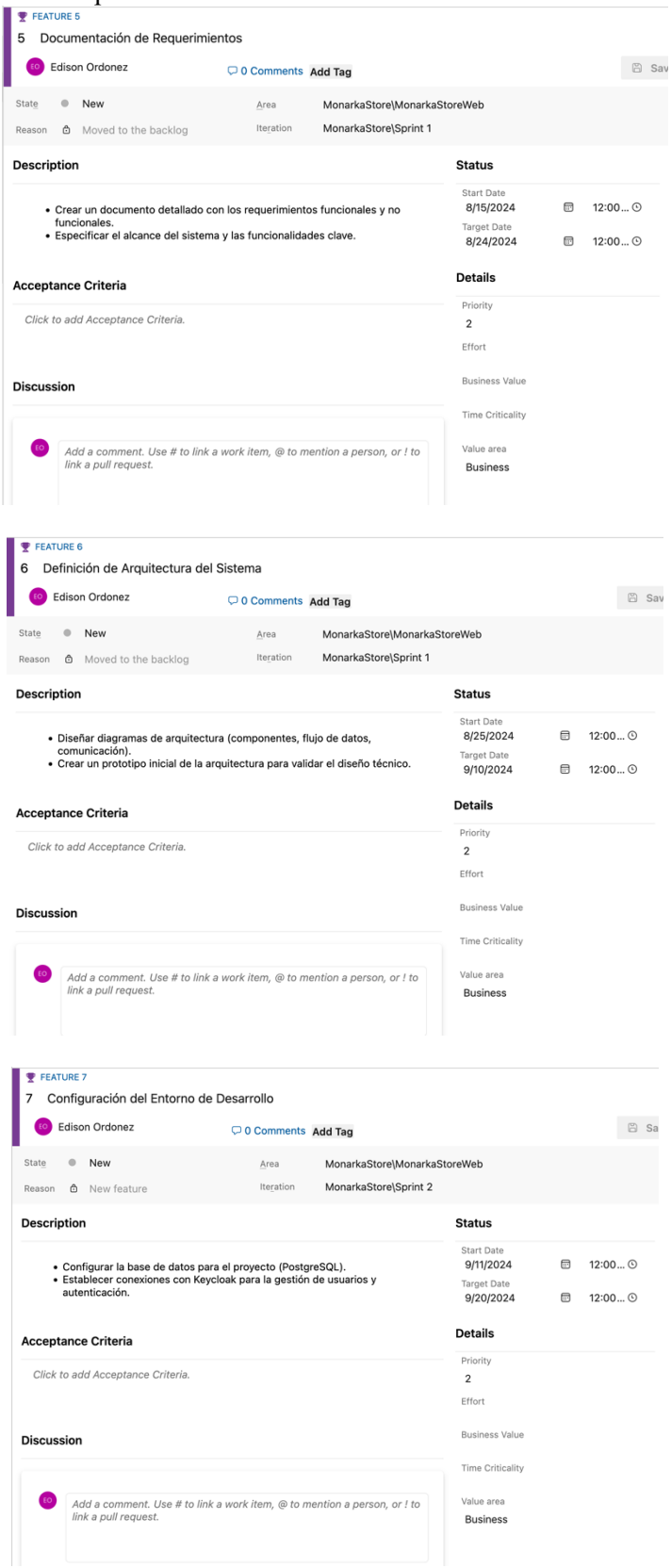
[23] Japan Management Association, *KANBAN: Y Just-in-time en Toyota*. Routledge, 2018.

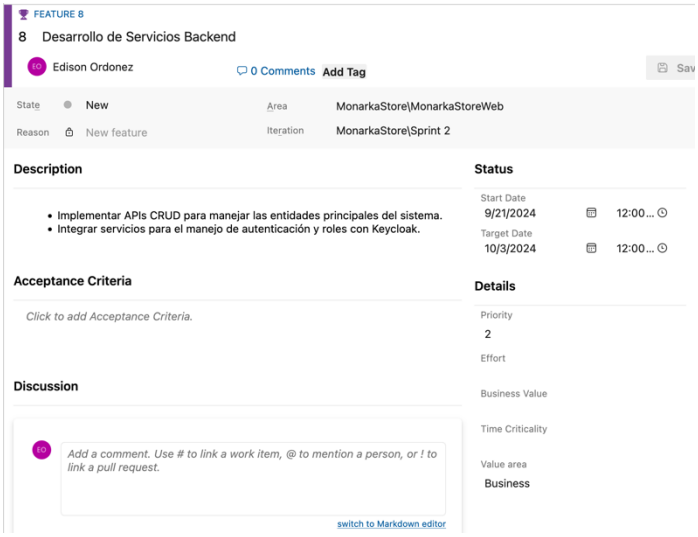
[24] K. Gómez Suárez, R. Anaya Hernández, and A. Cano, “Un acercamiento a los microservicios,” 2017.

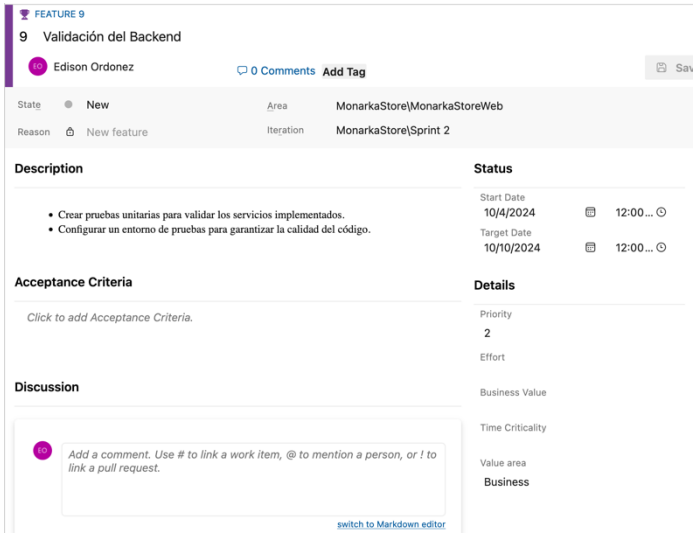
[25] F. Vera-Rivera, H. Astudillo, and C. Gaona, “Desarrollo de aplicaciones basadas en microservicios: tendencias y desafíos de investigación,” *RISTI-Revista Ibérica de Sistemas e Tecnologias de Informação*, no. E23 (2019), pp. 107–120, 2019.

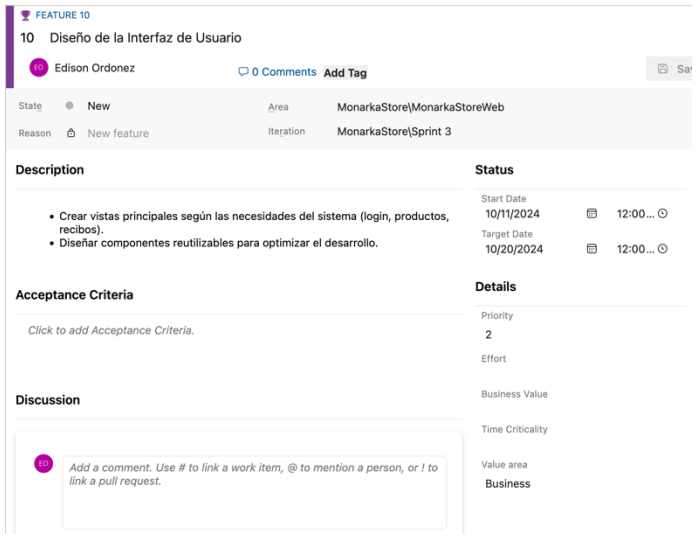
11. Anexos

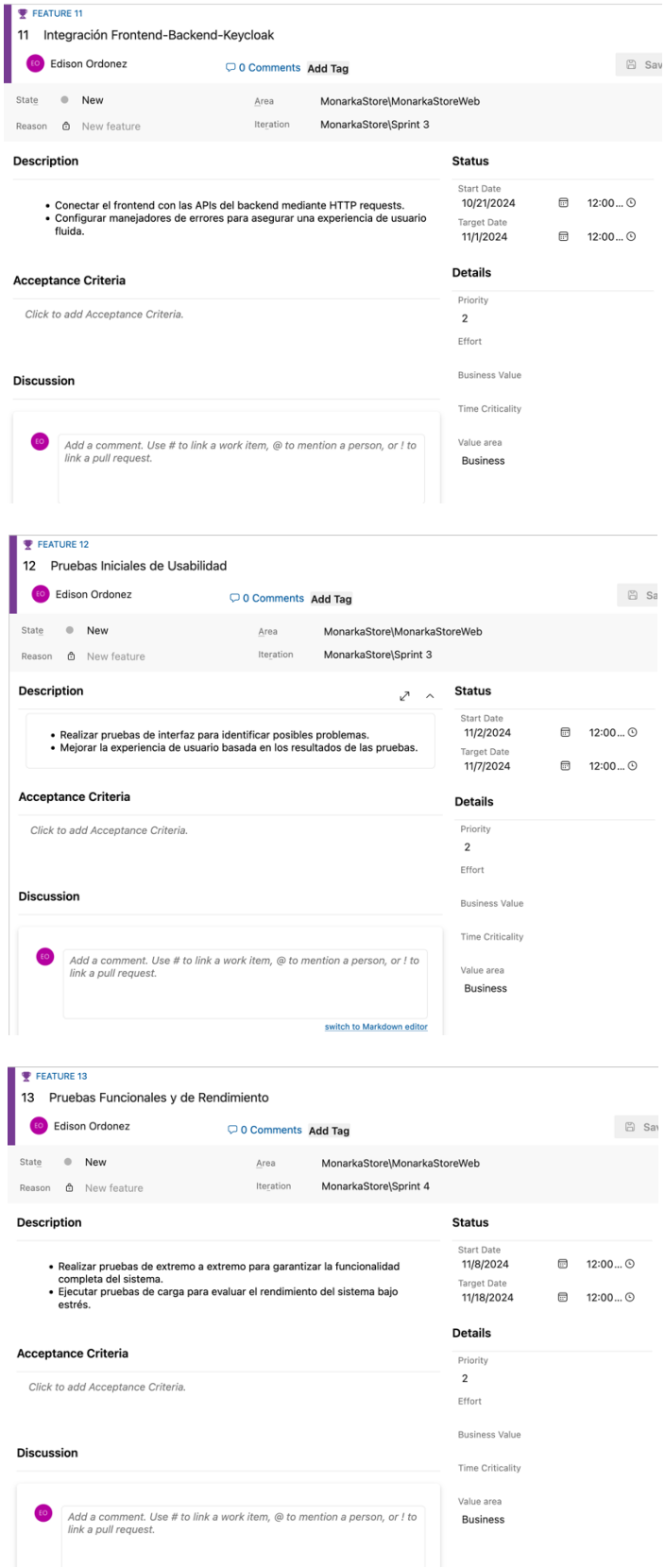
Anexo 1. Features detallados por funcionalidad

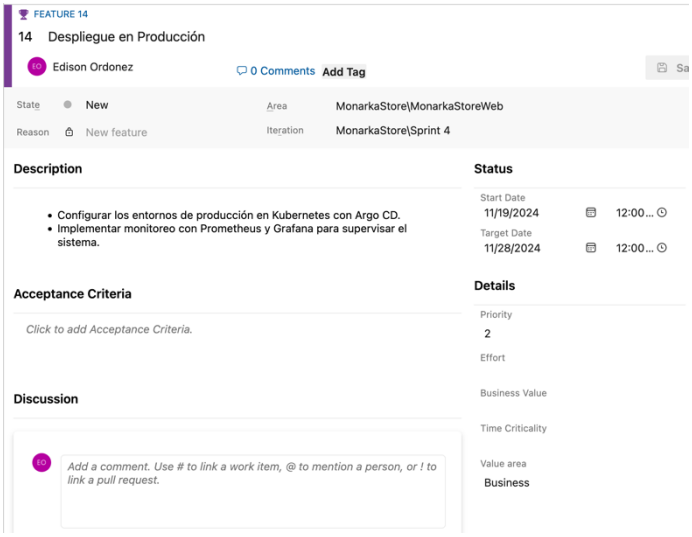



FEATURE 8
8 Desarrollo de Servicios Backend
Edison Ordonez
0 Comments
Add Tag
State
New
Reason
New feature
Area
MonarkaStore\MonarkaStoreWeb
Iteration
MonarkaStore\Sprint 2
Description
• Implementar APIs CRUD para manejar las entidades principales del sistema.
• Integrar servicios para el manejo de autenticación y roles con Keycloak.
Acceptance Criteria
Click to add Acceptance Criteria.
Discussion
Add a comment. Use # to link a work item, @ to mention a person, or ! to link a pull request.
switch to Markdown editor
Status
Start Date
9/21/2024
12:00...
Target Date
10/3/2024
12:00...
Details
Priority
2
Effort
Business Value
Time Criticality
Value area
Business


FEATURE 9
9 Validación del Backend
Edison Ordonez
0 Comments
Add Tag
State
New
Reason
New feature
Area
MonarkaStore\MonarkaStoreWeb
Iteration
MonarkaStore\Sprint 2
Description
• Crear pruebas unitarias para validar los servicios implementados.
• Configurar un entorno de pruebas para garantizar la calidad del código.
Acceptance Criteria
Click to add Acceptance Criteria.
Discussion
Add a comment. Use # to link a work item, @ to mention a person, or ! to link a pull request.
switch to Markdown editor
Status
Start Date
10/4/2024
12:00...
Target Date
10/10/2024
12:00...
Details
Priority
2
Effort
Business Value
Time Criticality
Value area
Business

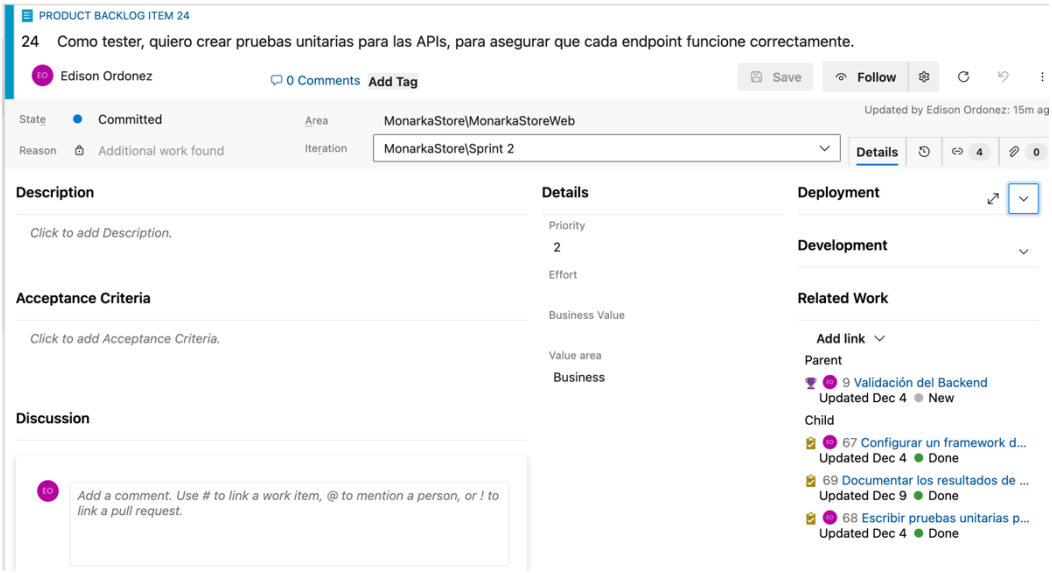

FEATURE 10
10 Diseño de la Interfaz de Usuario
Edison Ordonez
0 Comments
Add Tag
State
New
Reason
New feature
Area
MonarkaStore\MonarkaStoreWeb
Iteration
MonarkaStore\Sprint 3
Description
• Crear vistas principales según las necesidades del sistema (login, productos, recibos).
• Diseñar componentes reutilizables para optimizar el desarrollo.
Acceptance Criteria
Click to add Acceptance Criteria.
Discussion
Add a comment. Use # to link a work item, @ to mention a person, or ! to link a pull request.
Status
Start Date
10/11/2024
12:00...
Target Date
10/20/2024
12:00...
Details
Priority
2
Effort
Business Value
Time Criticality
Value area
Business

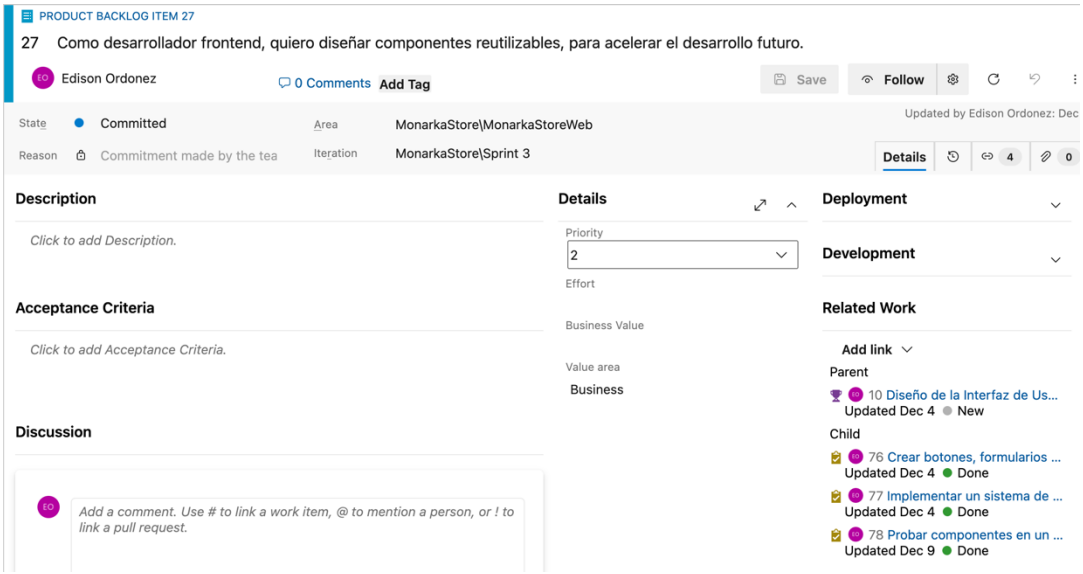

FEATURE 11
11 Integración Frontend-Backend-Keycloak
Edison Ordonez
0 Comments
Add Tag
State
New
Reason
New feature
Area
MonarkaStore\MonarkaStoreWeb
Iteration
MonarkaStore\Sprint 3
Description
• Conectar el frontend con las APIs del backend mediante HTTP requests.
• Configurar manejadores de errores para asegurar una experiencia de usuario fluida.
Acceptance Criteria
Click to add Acceptance Criteria.
Discussion
Add a comment. Use # to link a work item, @ to mention a person, or ! to link a pull request.
Status
Start Date
10/21/2024
12:00...
Target Date
11/1/2024
12:00...
Details
Priority
2
Effort
Business Value
Time Criticality
Value area
Business
FEATURE 12
12 Pruebas Iniciales de Usabilidad
Edison Ordonez
0 Comments
Add Tag
State
New
Reason
New feature
Area
MonarkaStore\MonarkaStoreWeb
Iteration
MonarkaStore\Sprint 3
Description
• Realizar pruebas de interfaz para identificar posibles problemas.
• Mejorar la experiencia de usuario basada en los resultados de las pruebas.
Acceptance Criteria
Click to add Acceptance Criteria.
Discussion
Add a comment. Use # to link a work item, @ to mention a person, or ! to link a pull request.
switch to Markdown editor
Status
Start Date
11/2/2024
12:00...
Target Date
11/7/2024
12:00...
Details
Priority
2
Effort
Business Value
Time Criticality
Value area
Business
FEATURE 13
13 Pruebas Funcionales y de Rendimiento
Edison Ordonez
0 Comments
Add Tag
State
New
Reason
New feature
Area
MonarkaStore\MonarkaStoreWeb
Iteration
MonarkaStore\Sprint 4
Description
• Realizar pruebas de extremo a extremo para garantizar la funcionalidad completa del sistema.
• Ejecutar pruebas de carga para evaluar el rendimiento del sistema bajo estrés.
Acceptance Criteria
Click to add Acceptance Criteria.
Discussion
Add a comment. Use # to link a work item, @ to mention a person, or ! to link a pull request.
Status
Start Date
11/8/2024
12:00...
Target Date
11/18/2024
12:00...
Details
Priority
2
Effort
Business Value
Time Criticality
Value area
Business

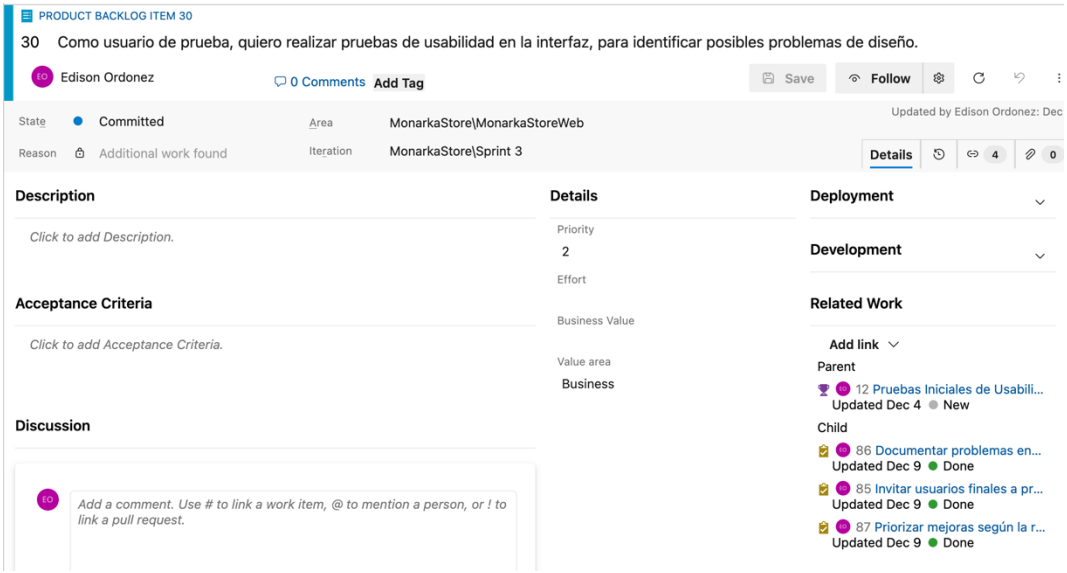

FEATURE 14
14 Despliegue en Producción
Edison Ordonez
0 Comments
Add Tag
State
New
Reason
New feature
Area
MonarkaStore\MonarkaStoreWeb
Iteration
MonarkaStore\Sprint 4
Description
Configurar los entornos de producción en Kubernetes con Argo CD.
Implementar monitoreo con Prometheus y Grafana para supervisar el sistema.
Acceptance Criteria
Click to add Acceptance Criteria.
Discussion
Add a comment. Use # to link a work item, @ to mention a person, or ! to link a pull request.
Status
Start Date
11/19/2024
12:00...
Target Date
11/28/2024
12:00...
Details
Priority
2
Effort
Business Value
Time Criticality
Value area
Business

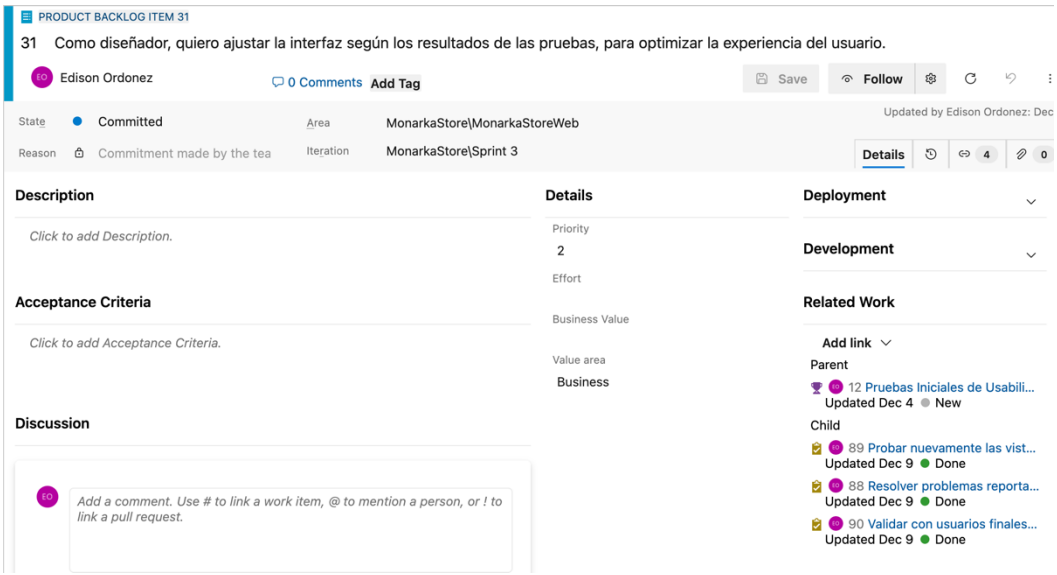
Anexo 2. Historias de usuarios

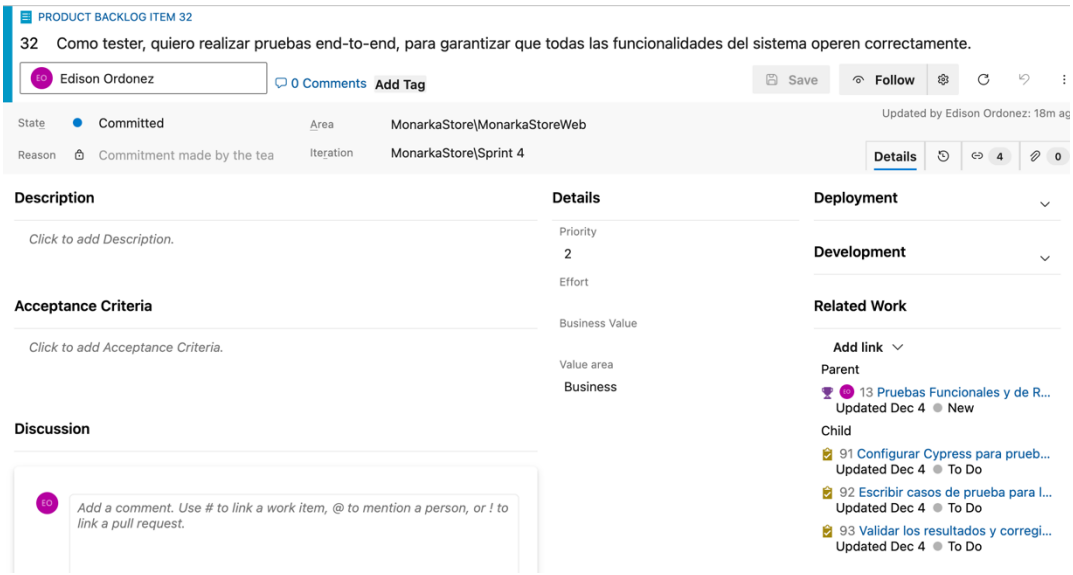
Historias de usuario detalladas por funcionalidades a resolver en la migración de la aplicación MONARKA, igualmente estas historias tienen vinculación entre funcionalidades entre API, Client y Keycloak.

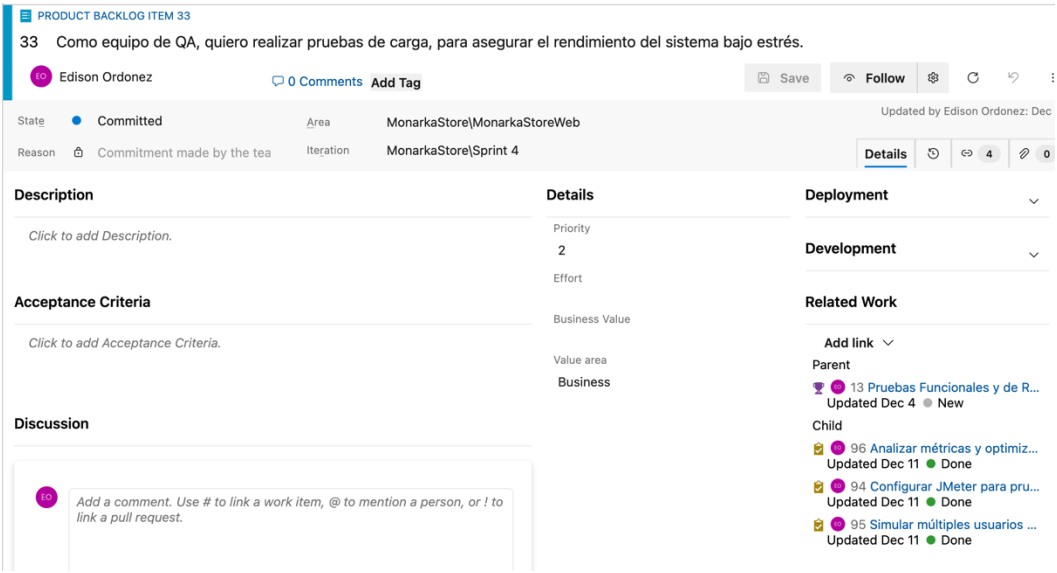


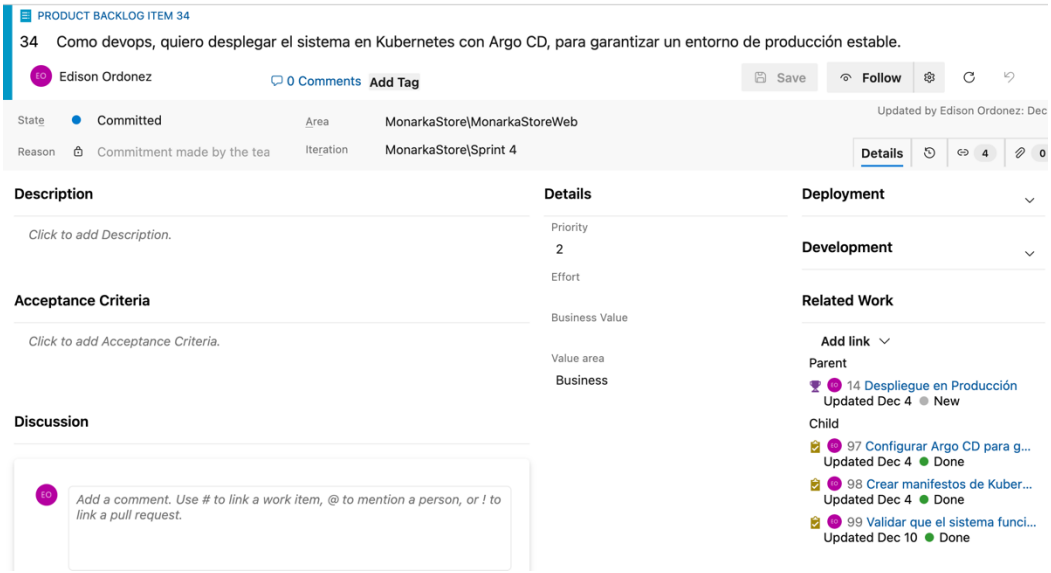


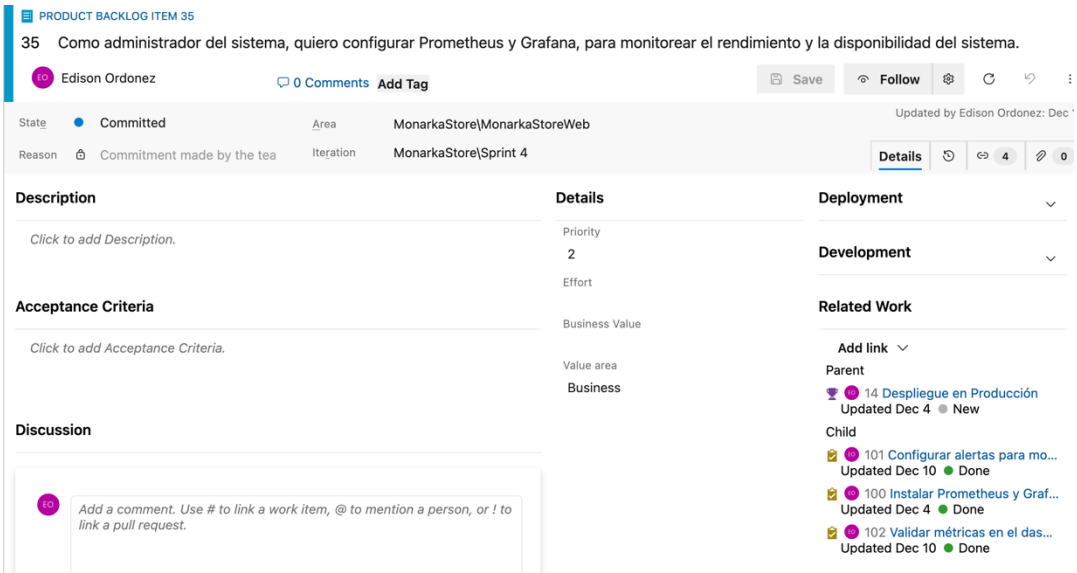

PRODUCT BACKLOG ITEM 30
30 Como usuario de prueba, quiero realizar pruebas de usabilidad en la interfaz, para identificar posibles problemas de diseño.
Edison Ordonez
0 Comments
Add Tag
Save
Follow
State
Committed
Area
MonarkaStore\MonarkaStoreWeb
Reason
Additional work found
Iteration
MonarkaStore\Sprint 3
Updated by Edison Ordonez: Dec
Details
Description
Click to add Description.
Acceptance Criteria
Click to add Acceptance Criteria.
Discussion
Add a comment. Use # to link a work item, @ to mention a person, or ! to link a pull request.
Details
Priority
2
Effort
Business Value
Value area
Business
Deployment
Development
Related Work
Add link
Parent
12 Pruebas Iniciales de Usabili...
Updated Dec 4 New
Child
86 Documentar problemas en...
Updated Dec 9 Done
85 Invitar usuarios finales a pr...
Updated Dec 9 Done
87 Priorizar mejoras según la r...
Updated Dec 9 Done


PRODUCT BACKLOG ITEM 31
31 Como diseñador, quiero ajustar la interfaz según los resultados de las pruebas, para optimizar la experiencia del usuario.
Edison Ordonez
0 Comments
Add Tag
Save
Follow
State
Committed
Area
MonarkaStore\MonarkaStoreWeb
Reason
Commitment made by the tea
Iteration
MonarkaStore\Sprint 3
Updated by Edison Ordonez: Dec
Details
Description
Click to add Description.
Acceptance Criteria
Click to add Acceptance Criteria.
Discussion
Add a comment. Use # to link a work item, @ to mention a person, or ! to link a pull request.
Details
Priority
2
Effort
Business Value
Value area
Business
Deployment
Development
Related Work
Add link
Parent
12 Pruebas Iniciales de Usabili...
Updated Dec 4 New
Child
89 Probar nuevamente las vist...
Updated Dec 9 Done
88 Resolver problemas reporta...
Updated Dec 9 Done
90 Validar con usuarios finales...
Updated Dec 9 Done


PRODUCT BACKLOG ITEM 32
32 Como tester, quiero realizar pruebas end-to-end, para garantizar que todas las funcionalidades del sistema operen correctamente.
Edison Ordonez
0 Comments
Add Tag
Save
Follow
State
Committed
Area
MonarkaStore\MonarkaStoreWeb
Reason
Commitment made by the tea
Iteration
MonarkaStore\Sprint 4
Updated by Edison Ordonez: 18m ag
Details
Description
Click to add Description.
Acceptance Criteria
Click to add Acceptance Criteria.
Discussion
Add a comment. Use # to link a work item, @ to mention a person, or ! to link a pull request.
Details
Priority
2
Effort
Business Value
Value area
Business
Deployment
Development
Related Work
Add link
Parent
13 Pruebas Funcionales y de R...
Updated Dec 4 New
Child
91 Configurar Cypress para prueb...
Updated Dec 4 To Do
92 Escribir casos de prueba para l...
Updated Dec 4 To Do
93 Validar los resultados y corregi...
Updated Dec 4 To Do


PRODUCT BACKLOG ITEM 33
33 Como equipo de QA, quiero realizar pruebas de carga, para asegurar el rendimiento del sistema bajo estrés.
Edison Ordonez
0 Comments
Add Tag
Save
Follow
State
Committed
Area
MonarkaStore\MonarkaStoreWeb
Reason
Commitment made by the tea
Iteration
MonarkaStore\Sprint 4
Updated by Edison Ordonez: Dec
Details
Description
Click to add Description.
Details
Priority
2
Effort
Business Value
Value area
Business
Deployment
Development
Related Work
Add link
Parent
13 Pruebas Funcionales y de R...
Updated Dec 4 New
Child
96 Analizar métricas y optimiz...
Updated Dec 11 Done
94 Configurar JMeter para pru...
Updated Dec 11 Done
95 Simular múltiples usuarios ...
Updated Dec 11 Done
Acceptance Criteria
Click to add Acceptance Criteria.
Discussion
Add a comment. Use # to link a work item, @ to mention a person, or ! to link a pull request.


PRODUCT BACKLOG ITEM 34
34 Como devops, quiero desplegar el sistema en Kubernetes con Argo CD, para garantizar un entorno de producción estable.
Edison Ordonez
0 Comments
Add Tag
Save
Follow
State
Committed
Area
MonarkaStore\MonarkaStoreWeb
Reason
Commitment made by the tea
Iteration
MonarkaStore\Sprint 4
Updated by Edison Ordonez: Dec
Details
Description
Click to add Description.
Details
Priority
2
Effort
Business Value
Value area
Business
Deployment
Development
Related Work
Add link
Parent
14 Despliegue en Producción
Updated Dec 4 New
Child
97 Configurar Argo CD para g...
Updated Dec 4 Done
98 Crear manifestos de Kuber...
Updated Dec 4 Done
99 Validar que el sistema funci...
Updated Dec 10 Done
Acceptance Criteria
Click to add Acceptance Criteria.
Discussion
Add a comment. Use # to link a work item, @ to mention a person, or ! to link a pull request.


PRODUCT BACKLOG ITEM 35
35 Como administrador del sistema, quiero configurar Prometheus y Grafana, para monitorear el rendimiento y la disponibilidad del sistema.
Edison Ordonez
0 Comments
Add Tag
Save
Follow
State
Committed
Area
MonarkaStore\MonarkaStoreWeb
Reason
Commitment made by the tea
Iteration
MonarkaStore\Sprint 4
Updated by Edison Ordonez: Dec
Details
Description
Click to add Description.
Details
Priority
2
Effort
Business Value
Value area
Business
Deployment
Development
Related Work
Add link
Parent
14 Despliegue en Producción
Updated Dec 4 New
Child
101 Configurar alertas para mo...
Updated Dec 10 Done
100 Instalar Prometheus y Graf...
Updated Dec 4 Done
102 Validar métricas en el das...
Updated Dec 10 Done
Acceptance Criteria
Click to add Acceptance Criteria.
Discussion
Add a comment. Use # to link a work item, @ to mention a person, or ! to link a pull request.

Anexo 3. Certificación de traducción del resumen

CERTIFICACIÓN DE TRADUCCIÓN

Loja, 25 de diciembre de 2024

Lic. Viviana Valdivieso Loyola Mg. Sc.
DOCENTE DE INGLÉS

A petición verbal de la parte interesada:

CERTIFICA:

Que, desde mi legal saber y entender, como profesional en el área del idioma inglés, he procedido a realizar la traducción del resumen, correspondiente al Trabajo de Integración Curricular titulado **Migración de una aplicación de escritorio de cobros a un sistema web mediante procesos DevOps. Caso de Aplicación: Empresa MONARKA,** de la autoría de: **Edison Josué Ordóñez Monge,** portador de la cédula de identidad número **1104122922**

Para efectos de traducción se han considerado los lineamientos que corresponden a un nivel de inglés técnico, como amerita el caso.

Es todo cuanto puedo certificar en honor a la verdad, facultando al portador del presente documento, hacer uso del mismo, en lo que a bien tenga.

Atentamente. -

VIVIANA DEL CISNE
VALDIVIESO LOYOLA

Lic. Viviana Valdivieso Loyola Mg. Sc.
1103682991

N° Registro Senescyt 4to nivel **1031-2021-2296049**
N° Registro Senescyt 3er nivel **1008-16-1454771**