UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR

POSGRADOS

MAESTRÍA EN

SOFTWARE CON MENCIÓN EN DISEÑO DE ARQUITECTURA DE SISTEMAS

RPC-SO-34-NO.778-2021

OPCIÓN DE TITULACIÓN:

PROYECTO DE TITULACIÓN CON COMPONENTES DE INVESTIGACIÓN APLICADA Y/O DE DESARROLLO

TEMA:

DESARROLLO E IMPLEMENTACIÓN DE UNA APLICACIÓN PARA EL CONTROL Y RECUENTO DE INVENTARIO INTEGRADA CON ERP SAP BUSINESS ONE, UTILIZANDO MICROSERVICIOS, DISPOSITIVOS IOT Y CÓDIGOS QR EN LA EMPRESA PLAPASA S.A.

AUTOR(ES)

CESAR HUGO GUERRA CAMPUZANO

DIRECTOR:

ERWIN JAIRO SACOTO CABRERA

GUAYAQUIL – ECUADOR

2026

Autor(es):

Cesar Hugo Guerra Campuzano
Ingeniero en Sistemas Computacionales
Candidato a Magíster en Software con Mención en Diseño de Arquitectura de Sistemas por la Universidad Politécnica Salesiana – Sede Guayaquil.
cguerrac1@est.ups.edu.ec

Dirigido por:

Erwin Jairo Sacoto Cabrera
Ingeniero Electrónico
Doctor en Telecomunicaciones
esacoto@ups.edu.ec

Cesar Hugo Guerra Campuzano
Desarrollo e implementación de una aplicación para el control y recuento de inventario integrada con ERP SAPBusiness One, utilizando microservicios, dispositivos IoT y códigos QR en la empresa Plapasa S.A.

DEDICATORIA

A mi familia, quien ha sido mi mayor apoyo a lo largo de este camino académico. Sus palabras de aliento en los momentos más difíciles y su inquebrantable fe en mí han sido el combustible que me impulsó a seguir adelante. Gracias por comprender las largas noches de estudio y por celebrar cada pequeño logro como si fuera el más grande. Este triunfo es tan suyo como mío.

A mis padres, Fátima y Hugo, quienes desde mi infancia sembraron en mí el amor por el conocimiento y la importancia de la educación. Ustedes son los arquitectos de mis sueños y los pilares de mi formación. Gracias por cada consejo, por cada esfuerzo realizado para brindarme las mejores oportunidades y por enseñarme que, con dedicación y perseverancia, todo es posible.

A mis hijos, mi mayor inspiración y la razón por la que busco ser mejor cada día. Espero que este logro les sirva de ejemplo y los motive a perseguir sus propios sueños con la misma valentía y determinación.

A todos ustedes, con amor y gratitud infinitos, dedico este trabajo.

AGRADECIMIENTO

A Dios, por concederme la fortaleza, la sabiduría y la perseverancia necesarias para culminar esta etapa tan importante de mi vida académica. Gracias por bendecirme con salud y con las oportunidades que hicieron posible este logro.

A la Universidad Politécnica Salesiana, por abrirme sus puertas y ofrecerme una formación académica de excelencia. A todos los docentes que compartieron sus conocimientos y experiencias durante la Maestría en Software, y que contribuyeron de manera significativa a mi crecimiento profesional.

A mi tutor de tesis, el Ing. Erwin Sacoto, por su invaluable guía, dedicación y apoyo constante durante el desarrollo de este trabajo. Sus conocimientos, experiencia y observaciones críticas fueron fundamentales para la culminación exitosa de este proyecto.

A la empresa en la que actualmente laboro, Plapasa S.A., especialmente a la gerencia de operaciones, Ing. Fernando Elizalde, y al equipo de sistemas, encabezado por el Ing. David Albarracín, por permitirme desarrollar este proyecto y brindarme las facilidades y recursos tecnológicos necesarios. Su colaboración y apertura fueron esenciales para la implementación práctica de esta solución.

A mis compañeros de maestría, por convertir este camino en una experiencia enriquecedora, llena de aprendizaje compartido, apoyo mutuo y vivencias que atesoraré siempre.

A todos aquellos que, de una u otra manera, contribuyeron a la realización de este proyecto, mi más sincero agradecimiento.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

Desarrollo e implementación de una aplicación para el control y recuento de inventario integrada con ERP SAP Business One, utilizando microservicios, dispositivos IoT y códigos QR en la empresa Plapasa S.A.

Autor(es):

CESAR HUGO GUERRA CAMPUZANO

Resumen

El presente trabajo de titulación desarrolla e implementa una solución tecnológica integral para el control y recuento de inventario en la empresa Plapasa S.A., integrada con el sistema ERP SAP Business One. La solución combina una arquitectura de microservicios desarrollada en .NET Core, una aplicación móvil multiplataforma en Flutter, tecnología de códigos QR para identificación de productos, y Kong como API Gateway para la gestión centralizada de comunicaciones.

La arquitectura implementada consta de cuatro microservicios independientes: AuthService para autenticación y autorización mediante JWT, UserService para gestión de usuarios y permisos basada en roles, OitmService para consulta de productos desde SAP HANA en tiempo real, y OincService para gestión de documentos de conteo integrados con SAP Service Layer. Cada microservicio opera con su propia base de datos PostgreSQL, siguiendo principios de Clean Architecture, SOLID y patrones de diseño como Repository, Service y Dependency Injection.

El desarrollo se ejecutó mediante metodología ágil Scrum en 12 sprints organizados en 4 fases: Fundamentos y Arquitectura, Integración con SAP, Aplicación Móvil, y Optimización y Despliegue. La integración con SAP Business One se realizó mediante dos estrategias complementarias: Service Layer para operaciones transaccionales de conteo, y conexión directa a SAP HANA mediante ODBC para consultas de productos y stock con alto rendimiento.

Los resultados obtenidos demuestran una mejora significativa en la eficiencia operativa del proceso de inventario. El tiempo promedio de conteo se redujo de 45 a 12 segundos por ítem (73.3% de reducción), la precisión del inventario mejoró del 92% al 99.5%, y la disponibilidad del sistema alcanzó 99.2% durante el período de prueba de 30 días. Las pruebas de rendimiento con Apache JMeter validaron la

capacidad del sistema para manejar 100 usuarios concurrentes con tiempo de respuesta promedio de 245ms y tasa de error menor al 0.1%.

La solución implementa seguridad mediante autenticación JWT con refresh tokens, autorización basada en políticas, encriptación HTTPS/TLS, y registro de auditoría de todas las operaciones. La arquitectura de microservicios proporciona escalabilidad horizontal, alta disponibilidad mediante health checks, y mantenibilidad a través de la separación de responsabilidades y desacoplamiento de servicios.

Palabras clave:

Microservicios, SAP Business One, Control de inventario, Flutter, .NET Core, API Gateway, Clean Architecture, Integración ERP.

Abstract

This thesis project develops and implements a comprehensive technological solution for inventory control and counting at Plapasa S.A., integrated with the SAP Business One ERP system. The solution combines a microservices architecture developed in .NET Core, a cross-platform mobile application in Flutter, QR code technology for product identification, and Kong as an API Gateway for centralized communications management.

The implemented architecture consists of four independent microservices: AuthService for authentication and authorization using JWT, UserService for user management and role-based permissions, OitmService for real-time product queries from SAP HANA, and OincService for counting document management integrated with SAP Service Layer. Each microservice operates with its own PostgreSQL database, following Clean Architecture principles, SOLID, and design patterns such as Repository, Service, and Dependency Injection.

Development was executed using Scrum agile methodology across 12 sprints organized into 4 phases: Fundamentals and Architecture, SAP Integration, Mobile Application, and Optimization and Deployment. Integration with SAP Business One was achieved through two complementary strategies: Service Layer for transactional counting operations, and direct connection to SAP HANA via ODBC for high-performance product and stock queries.

The results obtained demonstrate significant improvement in operational efficiency of the inventory process. Average counting time was reduced from 45 to 12 seconds per item (73.3% reduction), inventory accuracy improved from 92% to 99.5%, and system availability reached 99.2% during the 30-day testing period.

Performance tests with Apache JMeter validated the system's capacity to handle 100 concurrent users with an average response time of 245ms and error rate below 0.1%.

The solution implements security through JWT authentication with refresh tokens, policy-based authorization, HTTPS/TLS encryption, and audit logging of all operations. The microservices architecture provides horizontal scalability, high availability through health checks, and maintainability through separation of concerns and service decoupling.

Palabras clave:

Microservices, SAP Business One, Inventory Control, Flutter, .NET Core, API Gateway, Clean Architecture, ERP Integration.

1. Introducción

En el contexto actual de transformación digital, las organizaciones se encuentran en un proceso de modernización profunda que implica la adopción de tecnologías emergentes para optimizar sus operaciones, mejorar la eficiencia y fortalecer la toma de decisiones basada en datos (Sacoto-Cabrera E. , Perez-Torres, Tello-Oquendo, & Cerrada, 2025) . La digitalización de procesos críticos, como la gestión y control de inventarios, se ha convertido en un componente esencial para aumentar la competitividad, garantizar la trazabilidad de la información y permitir la integración fluida entre sistemas empresariales, dispositivos móviles y plataformas inteligentes (Sacoto-Cabrera & Perez-Torres, 2023).

La gestión eficiente del inventario es un factor crítico para optimizar la cadena de suministro, reducir costos operativos y garantizar la disponibilidad de productos. Las organizaciones que integran sus procesos de control de inventario con sistemas de Planificación de Recursos Empresariales (ERP) alcanzan mayor precisión en la información y fortalecen la toma de decisiones (Monk, E., Wagner, B., 2013). En este contexto, la empresa Plapasa S.A. requiere una solución tecnológica que optimice sus procesos de control y recuento de inventario, asegurando una integración eficiente con su sistema ERP.

El presente trabajo aborda el desarrollo e implementación de una solución tecnológica integral para el control y recuento de inventario, diseñada para integrarse con SAP Business One, uno de los ERP más utilizados en Pequeñas y Medianas Empresas (PYME). La solución propuesta combina tecnologías emergentes como microservicios, dispositivos Internet de las Cosas (IoT) y código QR para crear un ecosistema tecnológico que automatiza y optimiza los procesos de inventario.

La arquitectura implementada se fundamenta en principios de desarrollo moderno de software, empleando microservicios en .NET Core con C# para el backend, lo que

garantiza escalabilidad, mantenibilidad y alta disponibilidad. El frontend móvil desarrollado en Flutter permite operaciones multiplataforma, mientras que Kong como API Gateway centraliza la gestión de comunicaciones entre componentes. La integración con SAP Business One se realiza mediante Service Layer y conexiones ODBC a bases de datos PostgreSQL y SAP HANA, asegurando la sincronización en tiempo real de la información. Esta solución no solo moderniza los procesos de inventario, sino que también establece un framework replicable para futuras integraciones empresariales, demostrando cómo la convergencia de tecnologías puede transformar procesos operativos en sistemas inteligentes y eficientes.

El presente proyecto tiene como objetivo general desarrollar e implementar una solución tecnológica integral que optimice el proceso de control y recuento de inventario mediante el uso de microservicios, dispositivos móviles y la integración directa con SAP Business One. Entre sus objetivos específicos se encuentran mejorar la precisión del inventario, reducir los tiempos operativos, facilitar la trazabilidad de datos y asegurar la disponibilidad de información en tiempo real. El alcance del proyecto abarca el análisis, diseño arquitectónico, desarrollo del backend y aplicación móvil, así como la integración con el ERP y la validación funcional en el entorno operativo de Plapasa S.A.

El documento se estructura en capítulos que desarrollan de forma progresiva cada componente del proyecto. El Capítulo 2 presenta el marco teórico y conceptual que fundamenta la solución, abarcando microservicios, integración con ERP, tecnologías móviles e IoT. El Capítulo 3 describe el desarrollo del proyecto, la metodología aplicada, el análisis de requisitos y el diseño arquitectónico de la solución. El Capítulo 4 expone los resultados obtenidos y el análisis de su impacto en los procesos de inventario de la empresa. Finalmente, el Capítulo 5 presenta las conclusiones alcanzadas y las recomendaciones para trabajos futuros.

2. Marco teórico referencial

Arquitectura de Microservicios

La arquitectura de microservicios constituye un paradigma de diseño de software donde las aplicaciones se estructuran como una colección de servicios débilmente acoplados y altamente cohesivos (Newman et. al, 2021). Cada microservicio opera de manera independiente, puede desplegarse de forma autónoma y se orienta a una funcionalidad específica del negocio. La arquitectura de microservicios permite mayor flexibilidad en el desarrollo, facilita la escalabilidad horizontal y reduce el impacto de fallos individuales en el sistema complete (Richardson, 2018).

En el contexto del presente proyecto, la implementación de microservicios en .NET Core proporciona ventajas significativas. Al respecto los microservicios permiten que equipos independientes trabajen en paralelo, utilizando diferentes tecnologías según las necesidades específicas de cada servicio (Fowler, M., & Lewis, J., 2014). La comunicación entre servicios se realiza mediante protocolos HTTP/REST, lo que facilita la interoperabilidad y el mantenimiento del sistema (Nadareishvili, I., Mitra, R., McLarty, M., & Amundsen, M., 2016)

API Gateway y Gestión de Comunicaciones

Kong, como API Gateway de código abierto, funciona como punto único de entrada para todas las solicitudes del cliente hacia los microservicio (Palladino, 2021). Esta capa de abstracción ofrece funcionalidades esenciales como autenticación, autorización, limitación de tasa, y transformación de solicitudes. En este sentido, el API Gateway es esencial en arquitecturas de microservicios para centralizar preocupaciones transversales y simplificar la experiencia del cliente (Indrasiri, K., & Siriwardena, P., 2018).

La implementación de Kong en el proyecto permite gestionar eficientemente el tráfico entre la aplicación móvil Flutter y los microservicios backend,

proporcionando seguridad mediante JSON Web Tokens (JWT) y facilitando el versionado de APIs. Esta aproximación alinea con las mejores prácticas descritas por (Bruce, M., & Pereira, P. A., 2018), quienes enfatizan la importancia de un gateway robusto para mantener la seguridad y rendimiento en sistemas distribuidos (Sacoto-Cabrera, Chuchuca, Yupanqui, Reyes, & Martinez-Ledesma, 2025)

Integración con Sistemas ERP

SAP Business One es uno de los sistemas ERP más utilizados en empresas medianas, y se caracteriza por ofrecer módulos integrados para la gestión de finanzas, ventas, compras e inventario (SAP SE., 2023). La integración con sistemas ERP externos requiere consideraciones especiales de arquitectura para mantener la consistencia de datos y garantizar transacciones confiables. La integración efectiva con SAP requiere comprensión profunda de su Service Layer y capacidades de extensión (Monk, E., Wagner, B., 2013).

El Service Layer de SAP Business One proporciona una interfaz RESTful que permite operaciones CRUD sobre las entidades del sistema (Niefert, 2020). La implementación mediante ODBC para conexiones directas a SAP HANA ofrece acceso de alto rendimiento a datos transaccionales, como señalan (Plattner et.al., 2011) en su análisis de arquitecturas in-memory. Esta combinación de Service Layer y acceso directo a base de datos optimiza tanto la flexibilidad como el rendimiento del sistema integrado.

Tecnología Móvil

Flutter, desarrollado por Google, se ha consolidado como uno de los mejores framework líderes para el desarrollo móvil multiplataforma (Windmill, 2020). Su arquitectura basada en widgets y compilación a código nativo permite crear aplicaciones con alto rendimiento y experiencia de usuario consistente en los sistemas operativos iOS y Android. En este sentido Flutter reduce significativamente el tiempo de desarrollo al mantener una única base de código para múltiples plataformas (Napoli, 2019).

La elección de Flutter para la capa de presentación móvil se fundamenta en su capacidad de hot reload para desarrollo ágil, su rico conjunto de widgets Material Design y Cupertino, y su excelente rendimiento en operaciones de renderizado (Zammetti, 2019). La integración con códigos QR mediante la librería *mobile_scanner* permite captura eficiente de datos de inventario, alineándose con las tendencias de automatización en gestión de almacenes descritas por (Richards, G., 2017).

IoT y Códigos QR

La incorporación de dispositivos IoT y códigos QR en la gestión de inventarios constituye una evolución natural en el marco de la Industria 4.0 (Schwab, 2016). Los códigos QR proporcionan identificación única y rápida de productos, mientras que los dispositivos IoT pueden monitorear condiciones ambientales y movimientos de inventario en tiempo real. Así mismo la combinación de estas tecnologías puede reducir errores de inventario hasta en un 80% (Macaulay, J., Buckalew, L., & Chung, G., 2015).

La arquitectura propuesta integra scanners móviles como dispositivos IoT edge que procesan información localmente antes de sincronizar con el sistema central. Este enfoque híbrido reduce significativamente la latencia de respuesta y garantiza la continuidad operativa en modo offline, aspectos críticos en entornos de almacén donde la conectividad de red puede presentar interrupciones intermitentes (Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L., 2016).

Patrones de Diseño y Clean Code

La aplicación de patrones de diseño y principios de clean code resulta esencial para asegurar la calidad y sostenibilidad del software a largo plazo (Martin et. al, 2008). El patrón Repository implementado en el proyecto abstrae el acceso a datos, facilitando pruebas unitarias y cambios futuros en la persistencia. El patrón Service Layer, descrito por Fowler et. al. (2002), encapsula la lógica de negocio y coordina operaciones entre múltiples repositorios.

La implementación sigue los principios SOLID establecidos en Martin et. al.(2000), un conjunto de cinco principios de diseño orientado a objetos que promueven código flexible y mantenible. Específicamente se aplica el Principio de Responsabilidad Única (SRP) donde cada microservicio tiene una única razón para cambiar, y el Principio de Inversión de Dependencias (DIP) mediante inyección de dependencias en .NET Core (Microsoft, 2023). Estos principios, combinados con prácticas de clean code como nombres descriptivos, funciones pequeñas y comentarios mínimos pero significativos, resultan en código mantenible y extensible (Martin et. al, 2008).

Containerización con Docker

Docker es una plataforma de containerización que permite empaquetar aplicaciones y sus dependencias en contenedores ligeros y portables (Merkel, 2014). Esta tecnología ha revolucionado el despliegue de aplicaciones mediante containerización, proporcionando ambientes consistentes desde desarrollo hasta producción (Matthias, K., & Kane, S. P., 2018). La dockerización de microservicios garantiza portabilidad, escalabilidad y aislamiento de dependencias. Al respecto la tecnología Docker simplifica significativamente la gestión de microservicios al encapsular cada servicio con sus dependencias exactas (Poulton, 2023).

La orquestación mediante Docker Compose permite definir y ejecutar aplicaciones multi-contenedor, facilitando el desarrollo local y pruebas de integración (Gallagher, 2022). Esta aproximación es especialmente valiosa en arquitecturas de microservicios donde múltiples servicios deben coordinarse, como señalan (Burns et.al., 2016) en sus patrones de diseño para sistemas distribuidos containerizados.

3. Desarrollo del proyecto

Este capítulo presenta el desarrollo completo del proyecto de control de inventario, identificación inicial de requerimientos hasta la implementación final de la solución.

Se detallan las metodologías utilizadas, decisiones técnicas, arquitectura implementada y los resultados obtenidos durante el desarrollo.

Metodología de Desarrollo

El desarrollo del sistema se ejecutó mediante una metodología ágil híbrida que combina elementos de Scrum y Extreme Programming (XP), adaptada específicamente a las necesidades del proyecto de integración con SAP Business One. Esta aproximación permitió mantener la flexibilidad necesaria para ajustes técnicos mientras se cumplían los plazos establecidos con el área de operaciones.

Estructura de Sprints

La estructura de sprints implementada en este proyecto sigue el marco de trabajo Scrum adaptado a las necesidades específicas de un proyecto de integración empresarial con sistemas legacy. Según Schwaber et. al (2020), Scrum proporciona un marco ligero que ayuda a las personas, equipos y organizaciones a generar valor a través de soluciones adaptativas para problemas complejos. En el contexto de este proyecto, la complejidad radica en la integración con SAP Business One, la coordinación de múltiples microservicios y la necesidad de entregas incrementales que no interrumpan las operaciones del almacén.

El proyecto se estructuró en 12 sprints de 2 semanas cada uno, totalizando 24 semanas de desarrollo activo (aproximadamente 6 meses). Esta duración se seleccionó basándose en estudios que sugieren que sprints de 2 semanas ofrecen el equilibrio óptimo entre flexibilidad y momentum del proyecto, permitiendo ajustes frecuentes sin generar overhead excesivo de ceremonias (Sutherland, J., 2014). Sprints más cortos (1 semana) habrían generado interrupciones constantes en el flujo de desarrollo, particularmente problemático al trabajar con integraciones SAP que requieren configuraciones y validaciones extensas. Sprints más largos (3-4 semanas) habrían reducido la

capacidad de adaptación a cambios en requisitos, crítica en un entorno empresarial dinámico.

Duración de sprints: 2 semanas (14 días)

Ceremonias implementadas:

- Sprint Planning
- Daily Stand-ups
- Sprint Review y Retrospectiva

Sprint Planning (Día 1)

- Duración: 4 horas
- Participantes: Equipo de desarrollo, Product Owner (Jefe de Bodega), Stakeholder SAP
- Entregables: Sprint Backlog definido, criterios de aceptación documentados

El Sprint Planning se estructuró en dos fases diferenciadas: la primera orientada al qué, centrada en la selección de los ítems del Product Backlog de acuerdo con su prioridad de negocio y las dependencias técnicas identificadas; y la segunda orientada al cómo, dedicada a la descomposición técnica de las historias de usuario en tareas específicas, acompañadas de sus respectivas estimaciones en horas (Sacoto-Cabrera, Sarumeño-Ávila, Cuji-Torres, & Salamea-Palacios, 2024). La participación del stakeholder responsable de SAP resultó especialmente crítica durante los sprints de integración (4–6), ya que permitió validar la viabilidad técnica de las soluciones propuestas e identificar de manera temprana posibles restricciones del ERP que pudieran afectar las decisiones de diseño.

Daily Stand-ups

- Duración: 15 minutos
- Formato: Virtual/Presencial híbrido

- Foco: Impedimentos técnicos, integración SAP, sincronización entre servicios

Los Daily Stand-ups siguieron el formato clásico de tres preguntas:

- ¿Qué hice ayer?
- ¿Qué haré hoy?
- ¿Tengo impedimentos?

Dado que el proyecto involucró integración con sistemas externos (SAP Business One), se agregó una cuarta pregunta implícita:

- ¿Existen dependencias bloqueantes con sistemas externos?

Esto permitió identificar tempranamente problemas de conectividad, disponibilidad de ambientes SAP de prueba o cambios no comunicados en configuraciones del ERP.

Sprint Review y Retrospectiva (Día 14)

- Demostración en ambiente de pruebas
- Pruebas con usuarios reales de bodega
- Apuntes de lecciones aprendidas

El Sprint Review se llevó a cabo en un ambiente de pruebas que replicaba fielmente la configuración de producción, incluyendo la conexión con la base de datos SAP HANA del entorno de Quality Assurance (QA). Este enfoque permitió que las demostraciones reflejaran con precisión el comportamiento real del sistema, evitando depender únicamente de mocks o datos sintéticos.

La participación de los usuarios finales —particularmente los operadores de bodega— fue fundamental, ya que proporcionó retroalimentación inmediata especto a la usabilidad y a la adecuación de las funcionalidades a sus flujos de trabajo cotidianos.

- ¿Qué funcionó bien?
- ¿Qué se puede mejorar?
- ¿Acciones concretas para el próximo sprint?

Gestión del Ciclo de Desarrollo

Fase 1: Foundation Sprint (Sprints 1-3)

Establecimiento de infraestructura base, configuración de ambientes de desarrollo, implementación de servicios core de autenticación y gestión de usuarios. Esta fase sentó las bases arquitectónicas del sistema, definiendo patrones de diseño, convenciones de código y configuraciones de infraestructura que se replicarían en fases posteriores.

Sprint 1 - Configuración de ambiente y arquitectura base

En la Tabla 1 se presenta el Sprint 1 enfocado en la configuración del ambiente de desarrollo y la definición de la arquitectura base del sistema, estableciendo los fundamentos tecnológicos para los microservicios.

Tarea	Descripción	Horas estimadas
Setup Docker	Instalación y configuración de Docker y Docker Compose en servidor Ubuntu Linux	8h
Configuración PostgreSQL	Configuración de PostgreSQL y conexión inicial con SAP HANA	6h
Estructura microservicios	Estructura base de microservicios con .NET Core 8.0	10h

Tabla 1 Sprint 1: Configuración ambiente y arquitectura

Criterios de aceptación:

- Docker y Docker Compose funcionando correctamente
- Conexión establecida con PostgreSQL y SAP HANA
- Plantilla base de microservicios creada

Sprint 2 - Servicio de Autenticación

En la Tabla 2 se presenta el Sprint 2 dedicado al desarrollo del Servicio de Autenticación, implementando JWT y las funcionalidades de login/logout para la seguridad del sistema.

Tarea	Descripción	Horas estimadas
Implementación JWT	Implementación de JWT con refresh tokens	12h
Middleware autorización	Middleware de autorización compartido	6h
Base de datos usuarios	Base de datos de usuarios y roles en PostgreSQL	8h
Endpoints autenticación	Endpoints de login, logout y refresh	10h

Tabla 2 Sprint 2: Servicio de Autenticación

Criterios de aceptación:

- JWT implementado con refresh tokens funcional
- Middleware de autorización operativo
- Base de datos de usuarios creada
- Endpoints de autenticación probados

Sprint 3 - Gestión de Usuarios y Permisos

En la Tabla 3 se presenta el Sprint 3 centrado en la Gestión de Usuarios y Permisos, desarrollando el CRUD completo de usuarios, roles y la matriz de permisos del sistema.

Tarea	Descripción	Horas estimadas
CRUD usuarios	CRUD completo de usuarios en el backend	10h
Sistema de roles	Sistema de roles personalizado	8h
Permisos granulares	Permisos granulares por endpoint	6h
Auditoría login	Auditoría de login	4h

Tabla 3 Sprint 3: Gestión de Usuarios y Permisos

Criterios de aceptación:

- CRUD de usuarios funcionando
- Roles asignables a usuarios
- Permisos validándose en endpoints
- Logs de auditoría registrándose

Fase 2: Core Services (Sprints 4-6)

Integración con SAP Business One mediante conexiones directas a SAP HANA y Service Layer. Esta fase representó el mayor riesgo técnico del proyecto, requiriendo validaciones exhaustivas de conectividad, permisos y consistencia de datos entre sistemas.

Sprint 4 - Integración SAP HANA - Parte 1

En la Tabla 4 se presenta el Sprint 4 correspondiente a la primera fase de Integración SAP HANA, estableciendo la conexión directa con la base de datos y consultando tablas OITM (artículos).

Tarea	Descripción	Horas estimadas
Driver ODBC	Configuración de driver ODBC en contenedor Docker	8h
Repository pattern	Repository pattern para acceso a datos	10h
Consultas productos	Consultas básicas de productos	8h
Manejo errores	Manejo de errores	6h

Tabla 4 Sprint 4: Integración SAP HANA - Parte 1

Sprint 5 - Integración SAP HANA - Parte 2

En la Tabla 5 se presenta el Sprint 5 que completa la Integración SAP HANA con consultas a tablas OITW (stock por almacén) y optimización de queries para mejorar el rendimiento.

Tarea	Descripción	Horas estimadas
Endpoints inventario	Endpoints de inventario por bodega	10h
Sincronización movimientos	Sincronización de movimientos	8h
Optimización queries	Optimización de queries	6h

Tabla 5 Sprint 5: Integración SAP HANA - Parte 2

Sprint 6 - Service Layer SAP Business One

En la Tabla 6 se presenta el Sprint 6 enfocado en la integración con Service Layer SAP Business One, implementando autenticación y operaciones CRUD de documentos de inventario.

Tarea	Descripción	Horas estimadas
Integración API REST SAP	Integración con API REST de SAP B1	12h
Mapeo entidades	Mapeo de entidades SAP a DTOs	8h
Actualización inventarios	Actualización de inventarios	10h
Manejo transacciones	Manejo de transacciones	6h

Tabla 6 Sprint 6: Service Layer SAP Business One

Fase 3: Frontend & User Experience (Sprints 7-9)

Desarrollo de aplicación móvil Flutter con funcionalidades de escaneo QR, consultas de inventario y registro de conteos. El enfoque user-centric de esta fase incluyó sesiones de pruebas con usuarios reales del almacén al final de cada sprint.

Sprint 7 - Aplicación Flutter – Core

En la Tabla 7 se presenta el Sprint 7 dedicado al desarrollo del Core de la Aplicación Flutter, construyendo la estructura base, navegación, autenticación y gestión de estado.

Tarea	Descripción	Horas estimadas
Arquitectura clean	Arquitectura clean con Provider	12h
Pantallas login	Pantallas de login y navegación	8h
Gestión estado	Gestión de estado y sesión	6h
Almacenamiento local	Almacenamiento local con SharedPreferences	4h

Tabla 7 Sprint 7: Aplicación Flutter – Core

Sprint 8 - Integración con backend

En la Tabla 8 se presenta el Sprint 8 centrado en la Integración con Backend, conectando la aplicación Flutter con los microservicios y desarrollando las pantallas de consulta de stock.

Tarea	Descripción	Horas estimadas
Consumo AuthService	Consumo de microservicio de auth y userservice	8h
Consumo OitmService	Consumo de microservicio oitmservice para consulta de productos	10h
Consumo OincService	Consumo de microservicio oincservice para creación de recuento	8h

Tabla 8 Sprint 8: Integración con backend

Sprint 9 - Scanner QR y Funcionalidades

En la Tabla 9 se presenta el Sprint 9 enfocado en el Scanner QR y Funcionalidades, implementando la lectura de códigos QR, búsqueda de productos y actualización de conteos.

Tarea	Descripción	Horas estimadas
Integración scanner	Integración de mobile_scanner	8h
Búsqueda manual	Búsqueda manual de productos	6h
Visualización inventario	Visualización de inventario	8h
Sincronización backend	Sincronización con backend	6h
Manejo conflictos	Manejo de conflictos	4h

Tabla 9 Sprint 9: Scanner QR y Funcionalidades

Fase 4: Security & Deployment (Sprints 10-12)

Implementación de API Gateway Kong, hardening de seguridad, pruebas de carga, documentación y despliegue en producción. Esta fase garantizó que el sistema cumpliera con estándares enterprise de seguridad y rendimiento antes del lanzamiento.

Sprint 10 - Kong API Gateway

En la Tabla 10se presenta el Sprint 10 dedicado a la implementación de Kong API Gateway, configurando rutas, plugins de seguridad y rate limiting para proteger los microservicios.

Tarea	Descripción	Horas estimadas
Instalación Kong	Instalación y configuración de Kong	10h
Registro microservicios	Registro de microservicios	6h
Configuración rutas	Configuración de rutas y plugins	8h
Rate limiting y CORS	Rate limiting y CORS	4h

Tabla 10 Sprint 10: Kong API Gateway

Sprint 11 - Seguridad y Optimización

En la Tabla 11 se presenta el Sprint 11 centrado en Seguridad y Optimización, implementando encriptación, políticas de autorización, caché y monitoreo del sistema.

Tarea	Descripción	Horas estimadas
HTTPS	Implementación de HTTPS	8h
Hardening contenedores	Hardening de contenedores	6h
Optimización Docker	Optimización de imágenes Docker	6h
Health checks	Health checks y monitoring	6h

Tabla 11 Sprint 11: Seguridad y Optimización

Sprint 12 - Testing y Deployment

En la Tabla 12 se presenta el Sprint 12 final dedicado a Testing y Deployment, ejecutando pruebas unitarias, de integración, configurando Docker Compose y documentación de APIs.

Tarea	Descripción	Horas estimadas
Pruebas integración	Pruebas de integración end-to-end	12h
Pruebas carga	Pruebas de carga con JMeter	8h
Documentación	Documentación técnica y de usuario	10h
Despliegue producción	Despliegue en producción	8h

Tabla 12 Sprint 12: Testing y Deployment

Cada sprint mantuvo una estructura consistente de ceremonias Scrum, promoviendo transparencia, inspección y adaptación continua del proceso de desarrollo. La disciplina en las ceremonias fue fundamental para mantener sincronización entre desarrollo, operaciones de almacén y equipo técnico de SAP.

El levantamiento de requerimientos se llevó a cabo mediante un proceso estructurado de cuatro semanas que incorporó diversas técnicas de recolección de información, con el fin de asegurar una comprensión integral del problema y de las necesidades del negocio. Para dicho proceso se empleó la metodología Design Thinking, complementada con técnicas etnográficas de observación participante. Entre las herramientas utilizadas se destacan:

- Plantillas de entrevistas estructuradas basadas en el estándar IEEE 830
- Herramientas de mapeo de procesos para documentar
- Workshops colaborativos con técnicas de prototipado en papel

El producto resultante del levantamiento de requisitos fue:

- Documento de Especificación de Requisitos de Software (ERS) según IEEE 830
- 15 requisitos funcionales priorizados
- 8 requisitos no funcionales con criterios de aceptación medibles
- Prototipos de interfaz validados por stakeholders
- Matriz de trazabilidad de requisitos a casos de uso

Semana 1: Inmersión en el Proceso Actual

Observación Directa (Shadowing)

- 5 días completos en almacén observando el proceso de inventario

- Identificación de cuellos de botella y puntos de dolor

Hallazgos principales:

- Tiempo promedio de conteo: 15 minutos por ubicación
- Errores de transcripción: 3-5% de los registros
- Retrabajo por errores: 1 hora diaria promedio
- Desactualización de datos: 4-6 horas entre conteo y sistema

Semana 2: Entrevistas y Workshops

Entrevistas Estructuradas

La metodología empleada para las entrevistas estructuradas siguió el enfoque propuesto por Sommerville (2011) para ingeniería de requisitos, adaptando las técnicas al contexto empresarial de Plapasa S.A.

El proceso de entrevistas se diseñó con los siguientes elementos:

Preparación:

- Diseño de cuestionarios diferenciados por rol (gerencia, supervisores, bodegueros)
- Revisión previa de documentación ISO 9001 de la empresa
- Coordinación de sesiones con disponibilidad de todos los stakeholders

Herramientas utilizadas:

- Plantillas de entrevista semi-estructuradas
- Matriz de análisis de respuestas en Excel

Producto obtenido:

- Documento consolidado de "Necesidades del Usuario" con 23 puntos de dolor identificados
- Matriz de stakeholders con nivel de influencia y poder de decisión

Gerencia (2 sesiones de 1 hora):

- Objetivos estratégicos de control de inventario
- Integración con procesos empresariales

Jefe de Bodega (4 sesiones de 45 minutos):

- Problemas operativos diarios
- Necesidades de reportería
- Gestión de personal
- Puntos de integración con SAP

Bodegueros (6 sesiones de 30 minutos):

- Dificultades en el proceso actual
- Sugerencias de mejora
- Experiencia con tecnología móvil
- Preferencias de interfaz

Workshop de Diseño Participativo

- Duración: 4 horas
- Stakeholder

- Técnica: Design Thinking con prototipado en papel

Semana 3: Análisis de Sistemas Existentes

Auditoría SAP Business One

- Revisión de configuración actual de módulo de inventarios
- Análisis de tablas y campos personalizados
- Documentación de Service Layer endpoints disponibles
- Identificación de limitaciones y restricciones

Análisis de Infraestructura

- Evaluación de red WiFi en almacenes
- Disponibilidad de servidores
- Políticas de seguridad informática
- Capacidad de base de datos SAP HANA

Revisión Documental

- Procedimientos ISO 9001 de inventario
- Reportes históricos de auditorías
- Logs de errores del último año
- Documentación técnica de SAP

Semana 4: Consolidación y Validación

Matriz de Requerimientos Funcionales

A continuación, en la Tabla 13 se presenta la matriz consolidada de requerimientos funcionales identificados durante el proceso de levantamiento. Cada requerimiento ha sido priorizado según su impacto en los objetivos del negocio y clasificado por el tipo de usuario que lo solicita.

ID	Requerimiento	Prioridad	Usuario	Criterio de Aceptación
RF01	Escaneo de código QR de productos	Alta	Operador	Lectura en < 1 segundos, confirmación visual
RF02	Búsqueda manual por código/nombre	Alta	Operador	Resultados en < 2 segundo
RF03	Visualización de stock por bodega	Alta	Todos	Información actualizada en tiempo real
RF04	Actualización de conteo físico	Alta	Operador	Sincronización inmediata con SAP
RF05	Gestión de usuarios y roles	Alta	Administrador	CRUD completo, asignación de permisos
RF06	Auditoría de login	Alta	Administrador	Log no editable

Tabla 13 Matriz de Requerimientos Funcionales

Matriz de Requerimientos No Funcionales

La Tabla 14 detalla los requerimientos no funcionales del sistema, los cuales establecen las restricciones y criterios de calidad que debe cumplir la solución tecnológica. Estos requisitos son fundamentales para garantizar la operatividad, seguridad y rendimiento del sistema en el entorno productivo de Plapasa S.A.

ID	Requerimiento	Categoría	Métrica Objetivo
RNF01	Tiempo de respuesta	Rendimiento	< 3 segundos para 95% de transacciones
RNF02	Disponibilidad	Confiabilidad	99.5% uptime mensual
RNF03	Seguridad de autenticación	Seguridad	JWT con expiración configurable
RNF04	Compatibilidad móvil	Portabilidad	Android 7+, iOS 12+
RNF05	Backup y recuperación	Confiabilidad	RPO: 1 hora, RTO: 2 horas
RNF06	Trazabilidad	Auditoría	100% de transacciones registradas

Tabla 14 Matriz de Requerimientos No Funcionales

Matriz de Trazabilidad

La matriz de trazabilidad de la Tabla 15 permite establecer una relación directa entre los requisitos funcionales y no funcionales del sistema, los casos de uso definidos y los componentes arquitectónicos encargados de implementarlos.

Esta relación asegura que cada requisito identificado durante el levantamiento ha sido considerado en el diseño, desarrollo y validación del sistema. Asimismo, facilita la verificación de cumplimiento durante las pruebas y garantiza una cobertura completa del proyecto en todas sus etapas. La matriz constituye un elemento clave para asegurar la consistencia entre la especificación, el diseño y los resultados obtenidos.

Matriz de Trazabilidad del Sistema

ID	Descripción	Caso(s) de Uso Relacionado(s)	Componentes que lo implementan	Verificación / Evidencia
RF01	Escaneo de código QR de productos	CU01	Flutter App (scanner), OitmService	Pruebas funcionales de escaneo
RF02	Búsqueda manual por código/nombre	CU01, CU02	Flutter App, OitmService	Prueba de búsqueda y retorno de producto
RF03	Visualización de stock por bodega	CU02	OitmService, SAP HANA	Consulta de stock y validación en SAP
RF04	Actualización de conteo físico	CU01	OincService, OitmService, Service Layer SAP	Creación de documento OINC en SAP
RF05	Gestión de usuarios y roles	CU03	UserService, AuthService, PostgreSQL	CRUD de usuarios y roles funcionando
RF06	Auditoría de login	CU03 (indirecto)	AuthService, PostgreSQL	Registro de login/logout
RNF01	Tiempo de respuesta < 3 s	CU01, CU02, CU03	Todos los microservicios, Kong	Pruebas de rendimiento
RNF02	Disponibilidad 99.5%	Todos	Infraestructura Docker + Gateway	Health checks y monitoreo
RNF03	Seguridad mediante JWT	Todos	AuthService, Kong	Validación de token
RNF04	Compatibilidad móvil Android/iOS	CU01, CU02	Flutter App	Pruebas en dispositivos
RNF05	Backup y recuperación	No aplica a CU directo	PostgreSQL, SAP HANA	Procedimientos de backup
RNF06	Trazabilidad completa	CU01, CU03	AuthService, OincService	Registros de auditoría

Tabla 15 - Matriz de Trazabilidad del Sistema

3.1.1 Casos de Uso Principales

Los casos de uso representan las interacciones principales entre los usuarios y el sistema, describiendo el flujo de eventos desde la perspectiva del actor. A continuación, se presentan los tres casos de uso fundamentales que cubren las operaciones críticas del sistema de control de inventario.

Cada caso de uso se estructura siguiendo la plantilla estándar de UML, incluyendo:

- Actor principal y actores secundarios
- Precondiciones que deben cumplirse antes de la ejecución
- Flujo principal de eventos (camino feliz)
- Flujos alternativos para manejar excepciones y variaciones
- Postcondiciones que describen el estado del sistema tras la ejecución exitosa

Los casos de uso del sistema de control de inventario fueron diseñados a partir del análisis de los procesos operacionales de Plapasa S.A. y las necesidades identificadas durante el levantamiento de requerimientos. Cada caso de uso representa una funcionalidad crítica del sistema y ha sido estructurado siguiendo la notación estándar UML (Unified Modeling Language), especificando actores principales, precondiciones, flujo principal, flujos alternativos y postcondiciones.

A continuación, se presentan los tres casos de uso fundamentales que soportan las operaciones diarias de control y gestión de inventario:

- Figura 1 - CU01: Realizar Conteo de Inventario permite a los operadores realizar conteos físicos mediante escaneo de códigos QR con sincronización automática a SAP Business One.

- Figura 2 - CU02: Consultar Stock por Bodega habilita la consulta en tiempo real del stock disponible por cada bodega de la empresa accediendo directamente a SAP HANA.
- Figura 3 - CU03: Gestionar Usuarios del Sistema proporciona a los administradores las capacidades necesarias para gestionar usuarios, roles y permisos del sistema.

Estos casos de uso fueron validados mediante sesiones de revisión con usuarios finales y aseguran la trazabilidad completa entre los requerimientos funcionales (RF01-RF06) y la implementación técnica de los microservicios AuthService, UserService, OitmService y OincService.

CU01: Realizar Conteo de Inventario

Actor Principal: Operador de Almacén

Precondición: Usuario autenticado con rol Operador

El proceso completo de conteo de inventario involucra múltiples pasos coordinados entre el operador y el sistema. Como se observa en la Figura 1, el flujo inicia con la autenticación del operador y culmina con la sincronización de datos en SAP Business One, garantizando la trazabilidad de cada operación realizada.

Flujo Principal:

1. Operador selecciona opción "Conteo de Inventario"
2. Sistema muestra pantalla de scanner
3. Operador escanea código QR del producto
4. Sistema muestra información actual del producto
5. Operador ingresa cantidad física contada
6. Sistema valida y confirma actualización
7. Sistema sincroniza con SAP Business One
8. Sistema muestra confirmación de éxito

Flujos Alternativos:

3a. Código QR ilegible:

- Sistema activa búsqueda manual
- Operador ingresa código/nombre
- Continúa en paso 4

7a. Sin conexión a red:

- Sincroniza cuando recupera conexión

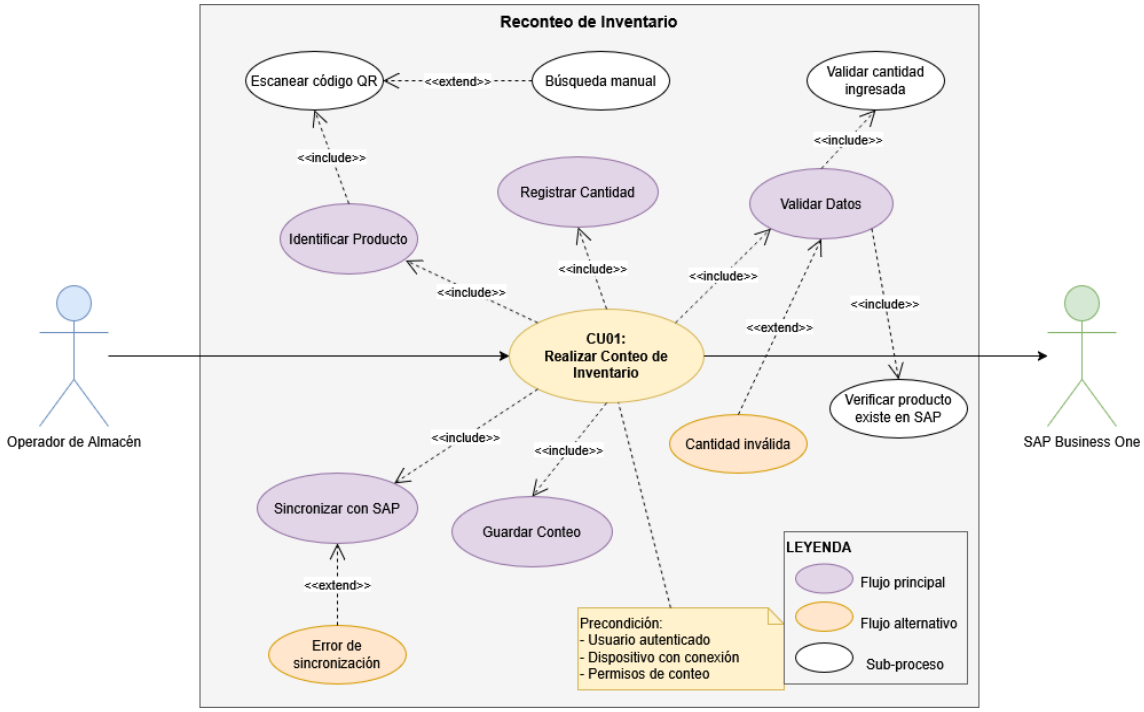


Figura 1 - CU01: Realizar Conteo de Inventario

CU02: Consultar Stock por Bodega

Actor Principal: Supervisor/Operador

Precondición: Usuario autenticado

La consulta de stock permite a supervisores y operadores obtener información actualizada en tiempo real sobre la disponibilidad de productos. En la Figura 2 se presenta el flujo de interacción que conecta

al usuario con la base de datos SAP HANA, mostrando las existencias disponibles y comprometidas por cada bodega.

Flujo Principal:

1. Usuario selecciona "Consulta de Stock"
2. Usuario ingresa/escanea código de producto
3. Sistema consulta SAP HANA
4. Sistema muestra stock por cada bodega
5. Usuario puede expandir detalles por bodega
6. Sistema muestra stock comprometido y disponible

Postcondición: Información mostrada sin modificaciones

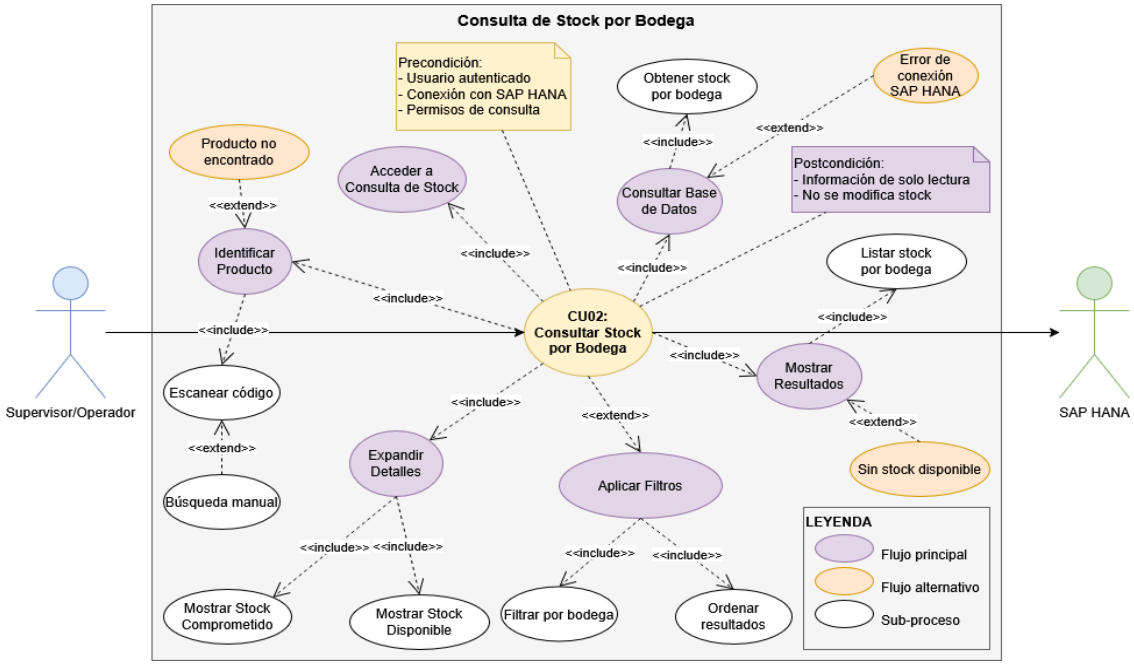


Figura 2 - CU02: Consultar Stock por Bodega

CU03: Gestionar Usuarios del Sistema

Actor Principal: Administrador

Precondición: Usuario con rol Administrador

La administración de usuarios es fundamental para mantener la seguridad y control de acceso del sistema. En la Figura 3 se observa el

proceso de gestión que permite al administrador crear, modificar y asignar roles a los usuarios, garantizando que cada operador tenga únicamente los permisos necesarios para sus funciones específicas.

Flujo Principal:

1. Administrador accede a "Gestión de Usuarios"
2. Sistema muestra lista de usuarios actuales
3. Administrador selecciona "Crear Usuario"
4. Administrador completa formulario (nombre, email, rol)
5. Sistema valida datos únicos
6. Sistema crea usuario y envía credenciales
7. Sistema registra operación en auditoría

Flujos Alternativos:

5a. Email duplicado:

- Sistema rechaza y muestra error
- Administrador corrige información

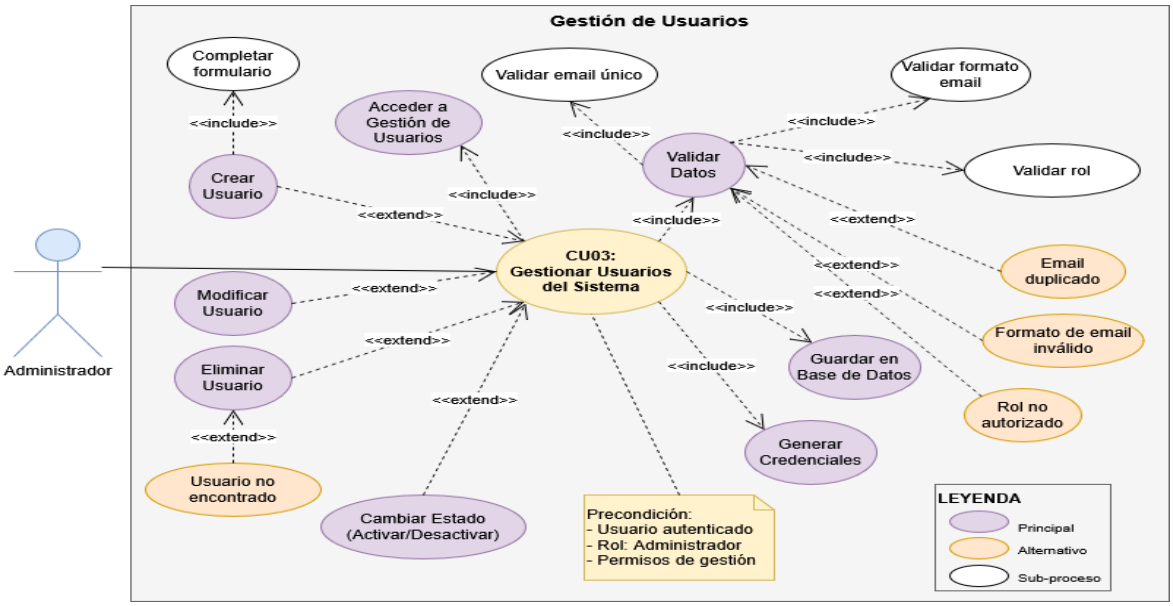


Figura 3 - CU03: Gestionar Usuarios del Sistema

Arquitectura del sistema

En la Figura 4 - Arquitectura del sistema implementa el patrón de microservicios, definido por Newman et. al (2021) como "un enfoque para desarrollar una aplicación como un conjunto de pequeños servicios, cada uno ejecutándose en su propio proceso y comunicándose con mecanismos ligeros". Esta arquitectura permite que cada componente evolucione independientemente, facilitando el mantenimiento y la escalabilidad del sistema completoFigura 4 - Arquitectura del sistema. A continuación, se describe cada componente de la arquitectura mostrado en la Figura 4:

1. **Captura**: Usuario escanea QR o ingresa código en app Flutter
2. **Transmisión**: Request HTTP hacia Kong Gateway
3. **Autenticación**: Kong valida JWT token con Auth Service
4. **Enrutamiento**: Kong direcciona al microservicio correspondiente
5. **Procesamiento**: Microservicio ejecuta lógica de negocio
6. **Persistencia**: Datos se almacenan en PostgreSQL/HANA según corresponda
7. **Sincronización**: Service Layer actualiza SAP Business One
8. **Respuesta**: Resultado retorna por la misma ruta al cliente
9. **Actualización UI**: Flutter actualiza la interfaz con los nuevos datos

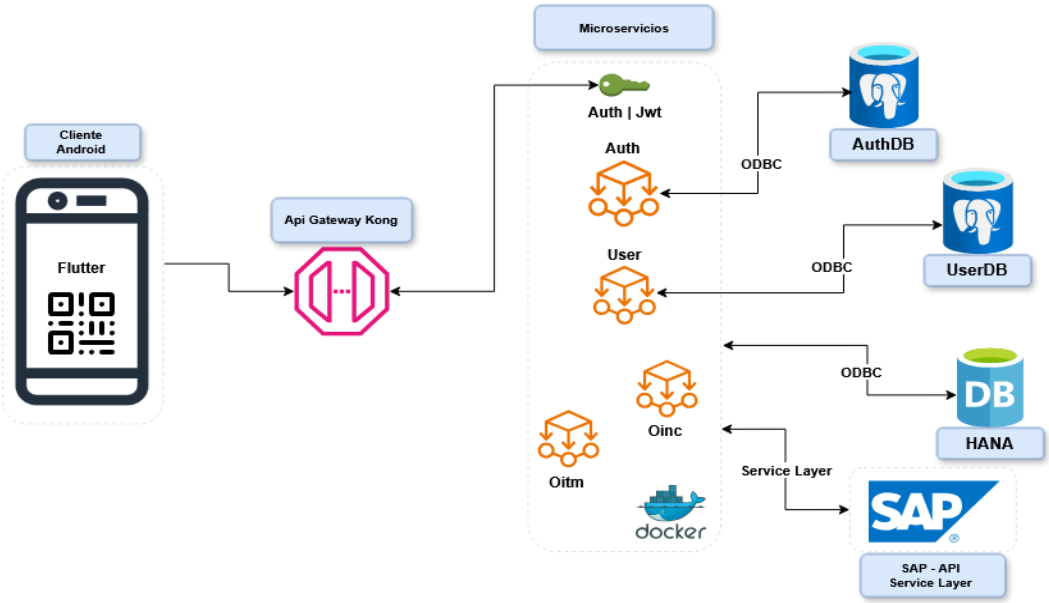


Figura 4 - Arquitectura del sistema

Cliente Móvil - Flutter:

El cliente móvil desarrollado en Flutter constituye la interfaz principal de interacción con los usuarios (véase Figura 4 - Arquitectura del sistema). De acuerdo con Windmill, (2020), "Flutter permite crear aplicaciones nativas compiladas para móvil, web y desktop desde una única base de código", lo que reduce significativamente el tiempo y costo de desarrollo multiplataforma.

Características implementadas:

- **Interfaz Material Design**: Consistencia visual y familiaridad para usuarios Android/iOS
- **Interfaz de usuario** responsiva adaptada a diferentes tamaños de pantalla
- **Capacidad de escaneo de códigos QR** mediante la cámara del dispositivo
- **Gestión local de estado con Provider**: Patrón recomendado para aplicaciones Flutter de mediana complejidad
- **Almacenamiento local de** sesión para mejorar la experiencia del usuario

API Gateway (Kong):

Kong opera como el punto central de entrada para todas las solicitudes del cliente dirigidas a los microservicios (ver Figura 4 - Arquitectura del sistema). Según Palladino (2021) “un API Gateway actúa como un proxy reverso que acepta todas las llamadas API, agrega los diversos servicios requeridos y devuelve el resultado apropiado”. Esta función permite un control unificado del enrutamiento, la seguridad y la gestión del tráfico dentro de la arquitectura distribuida del sistema.

Flujo de peticiones:

1. Cliente envía request a Kong
2. Kong valida JWT token
3. Kong aplica rate limiting y transformaciones
4. Kong enruta a microservicio correspondiente
5. Kong agrega headers de respuesta y retorna al cliente

Características implementadas:

- Autenticación y validación de tokens JWT
- Enrutamiento inteligente de solicitudes hacia microservicios específicos
- Rate limiting para prevenir sobrecarga del sistema
- Transformación de headers y gestión de CORS
- Load balancing entre instancias de microservicios

Microservicios Backend:

La arquitectura de microservicios se compone de servicios especializados que implementan el principio de responsabilidad única (Single Responsibility Principle del SOLID), garantizando alta cohesión dentro de cada servicio y bajo acoplamiento entre ellos.

Microservicio AuthService:

El servicio de autenticación implementa JSON Web Tokens (JWT) para manejo seguro de sesiones. (Jones, M., Bradley, J., & Sakimura, N., 2015) definen JWT como "un estándar abierto (RFC 7519) que define una forma compacta y autónoma de transmitir información de forma segura entre partes como un objeto JSON" (ver Figura 4 - Arquitectura del sistema).

Este microservicio constituye el punto de entrada obligatorio al sistema mediante un gateway centralizado. Sus funcionalidades principales incluyen: autenticación de usuarios con validación de credenciales, generación y renovación de tokens JWT (access y refresh tokens), revocación de tokens para cierre de sesión seguro, cambio de contraseña con validación de credencial

actual, gestión del ciclo de vida de credenciales (creación y eliminación), registro de actividad de sesiones (dirección IP y user agent), integración con el microservicio de usuarios para consulta de permisos y roles, y validación de tokens con extracción de claims de usuario. Toda la información de autenticación se almacena en una base de datos PostgreSQL dedicada exclusivamente a este microservicio, garantizando el aislamiento de datos sensibles.

El patrón de doble token (access token + refresh token) implementado permite sesiones seguras sin comprometer la experiencia del usuario. El access token tiene corta duración (1 hora) minimizando el riesgo de uso indebido, mientras el refresh token de larga duración (7 días) permite renovación automática sin solicitar credenciales repetidamente.

Funciones Principales

Autenticación de Usuarios:

- Login con validación de credenciales contra base de datos PostgreSQL
- Generación de tokens JWT (RS256) con claims de usuario, roles y permisos
- Consulta en tiempo real al microservicio UserService para obtener información actualizada del usuario
- Generación de access token (corta duración, 1 hora) y refresh token (larga duración, 7 días)
- Registro detallado de cada intento de login con IP address y user agent para auditoría
- Retorno de información completa de usuario incluyendo menú personalizado según permisos

Renovación de Tokens (Refresh Token Flow):

- Validación de refresh token contra base de datos
- Verificación de que el token no haya sido revocado manualmente
- Comprobación de fecha de expiración del refresh token

- Consulta de estado actualizado del usuario (activo/inactivo)
- Generación de nuevo par de tokens (access + refresh)
- Revocación automática del refresh token anterior para prevenir reutilización
- Rotación de tokens implementando el patrón "refresh token rotation" para mayor seguridad

Gestión de Sesiones:

- Revocación de tokens individuales para cierre de sesión en dispositivo específico
- Revocación de todos los tokens de un usuario para cierre de sesión global (logout from all devices)
- Cierre automático de sesión al cambiar contraseña
- Seguimiento de sesiones activas por usuario
- Identificación de dispositivos mediante user agent para detección de accesos sospechosos

Control de Seguridad:

- Contador de intentos fallidos de login por usuario
- Bloqueo temporal de cuenta tras 5 intentos fallidos (configurable, default: 5 intentos)
- Duración de bloqueo configurable (default: 30 minutos)
- Desbloqueo automático al expirar el tiempo de bloqueo
- Reseteo de contador tras login exitoso
- Prevención de ataques de fuerza bruta mediante rate limiting

Gestión de Contraseñas:

- Cambio de contraseña con validación de contraseña actual
- Hashing de contraseñas con BCrypt y salt único por usuario

- Factor de costo BCrypt configurable para adaptarse a capacidad de hardware
- Registro de fecha de último cambio de contraseña
- Revocación automática de todos los tokens al cambiar contraseña
- Almacenamiento seguro de hashes (nunca se almacenan contraseñas en texto plano)

Gestión del Ciclo de Vida de Credenciales:

- Creación de credenciales asociadas a usuarios existentes en UserService
- Validación de unicidad de username en el sistema
- Eliminación de credenciales y revocación automática de tokens asociados
- Sincronización con UserService para mantener consistencia de datos

Auditoría y Trazabilidad:

- Registro completo de historial de login (exitosos y fallidos)
- Captura de IP address y user agent de cada intento
- Registro de razón de fallo en intentos fallidos
- Timestamp UTC de cada evento para análisis temporal
- Logs estructurados en formato JSON para análisis con herramientas de observabilidad

Estructura del Servicio

El microservicio AuthService se estructura en las siguientes capas, conforme a los principios de la arquitectura hexagonal (Ports and Adapters), lo que permite una separación clara de responsabilidades y una alta independencia entre el dominio y las interfaces externas:

Capa de Presentación (Controllers):

- AuthController: Expone endpoints REST para operaciones de autenticación

- POST /auth/login: Autenticación de usuario
- POST /auth/revoke: Revocación de token específico
- POST /auth/logout: Cierre de sesión (revocación de todos los tokens)
- POST /auth/change-password: Cambio de contraseña
- POST /auth/credentials: Creación de credenciales (interno)
- DELETE /auth/credentials/{id}: Eliminación de credenciales (interno)

- Validación de parámetros de entrada mediante FluentValidation
- Transformación de excepciones a códigos HTTP semánticamente correctos
- Captura de IP address y user agent del request HTTP
- Documentación automática mediante Swagger/OpenAPI con ejemplos

Capa de Aplicación (Services):

- AuthService: Implementa toda la lógica de negocio de autenticación
  - Orquestación del flujo completo de login
  - Gestión del ciclo de vida de tokens
  - Implementación de políticas de seguridad
  - Coordinación con otros microservicios
- JwtService: Especializado en operaciones con JWT
  - Generación de access tokens con claims personalizados
  - Generación de refresh tokens seguros (GUID + hash)
  - Validación de firma de tokens
  - Extracción de claims de tokens
- PasswordService: Gestión segura de contraseñas
  - Hashing con BCrypt y salt único
  - Verificación de contraseñas contra hash
  - Generación de salt criptográficamente seguro
- UserServiceClient: Cliente HTTP para comunicación con UserService
  - Obtención de información completa de usuario

- Consulta de menú personalizado por permisos
- Implementación de circuit breaker con Polly
- Manejo de timeouts y reintentos

Capa de Dominio (Models y DTOs):

- Modelos de Dominio:
  - UserCredential: Entidad principal con username, hash, salt, contadores
  - RefreshToken: Entidad para tokens de larga duración
  - LoginHistory: Entidad para auditoría de accesos
- DTOs de Request:
  - LoginRequestDTO: Username y password
  - RefreshTokenRequestDTO: Refresh token a renovar
  - ChangePasswordRequestDTO: Password actual y nuevo
  - CreateCredentialRequestDTO: Datos para crear credencial
- DTOs de Response:
  - LoginResponseDTO: Access token, refresh token, info de usuario, menú
  - RefreshTokenResponseDTO: Nuevos tokens generados
  - UserInfoDTO: Información básica del usuario autenticado

Capa de Infraestructura (Repositories):

- AuthRepository: Acceso a datos en PostgreSQL mediante Entity Framework Core
  - Operaciones CRUD sobre credenciales
  - Gestión de refresh tokens
  - Registro de historial de login
  - Implementación de operaciones atómicas con transacciones
- IAuthRepository: Interfaz que abstrae el acceso a datos

- Permite cambiar implementación sin afectar lógica de negocio
- Facilita creación de mocks para pruebas unitarias

Capa de Configuración:

- Configuración de JWT (clave privada RSA, expiración, issuer, audience)
- Configuración de políticas de seguridad (intentos fallidos, tiempo de bloqueo)
- Configuración de duración de tokens
- Configuración de BCrypt (factor de costo)

Patrones de Diseño Implementados

Con el fin de garantizar una arquitectura robusta, mantenible y alineada con las mejores prácticas de ingeniería de software, en AuthService se aplicaron diversos patrones de diseño orientados a promover el desacoplamiento, la extensibilidad y la claridad en la organización del código. Estos patrones permiten abordar problemas recurrentes en sistemas distribuidos, mejorar la cohesión interna de los componentes y facilitar la evolución del servicio ante cambios funcionales o tecnológicos. A continuación, se describen los principales patrones utilizados y su aplicación específica dentro del microservicio.

- Repository Pattern: Este patrón abstrae el acceso a los datos, permitiendo reemplazar la implementación de persistencia sin afectar la lógica de negocio. Su propósito es desacoplar la capa de dominio de los detalles de infraestructura, facilitando la ejecución de pruebas unitarias mediante mocks y habilitando la migración a otros motores de base de datos cuando sea necesario. En AuthService, el repositorio encapsula todas las operaciones SQL, manteniendo la lógica de negocio completamente independiente de los mecanismos de persistencia.
- Service Layer Pattern: La capa de servicios encapsula la lógica de negocio en componentes reutilizables, promoviendo la implementación de *thin controllers*. Esta capa centraliza reglas de negocio complejas, como el flujo

de autenticación, las validaciones de seguridad y la orquestación de múltiples operaciones. De este modo, los controladores se limitan a la gestión de la comunicación HTTP, mientras que **AuthService** concentra las responsabilidades asociadas a la autenticación.

- Dependency Injection: Todas las dependencias se suministran a través de inyección por constructor, lo que reduce el acoplamiento e incrementa la capacidad de prueba del sistema. Este mecanismo invierte el control sobre la creación de objetos, delegando dicha responsabilidad al *framework* y permitiendo sustituir implementaciones en tiempo de ejecución, especialmente útiles durante las pruebas. Como resultado, se mejora la mantenibilidad, extensibilidad y coherencia de la arquitectura.
- DTO Pattern: Este patrón separa explícitamente los modelos de dominio de los objetos de transferencia de datos, controlando la información expuesta por la API. Se utiliza para evitar la divulgación de datos sensibles como *hashes* o *salts* de contraseñas, optimizar el tamaño de las respuestas eliminando campos innecesarios y desacoplar la interfaz pública del diseño interno de los datos. Por ejemplo, en ningún caso se exponen los valores de *salt* o *password hash* en los DTO de salida.
- Strategy Pattern (uso implícito): La definición de interfaces como IPasswordService y IJwtService habilita la adopción de distintas estrategias de hashing y generación de tokens sin modificar el código cliente. Esta flexibilidad permite sustituir algoritmos en caso de que surjan vulnerabilidades en BCrypt o JWT RS256, fortaleciendo la capacidad de evolución del sistema ante nuevos requerimientos de seguridad.
- Circuit Breaker Pattern: Implementado mediante Polly en UserServiceClient, este patrón mitiga fallos en cascada cuando un servicio dependiente experimenta indisponibilidad. Si UserService deja de responder, el circuit breaker se abre temporalmente, bloqueando llamadas posteriores y brindando tiempo para la recuperación del servicio, lo cual mejora la resiliencia y estabilidad del sistema distribuido.

- Audit Log Pattern: Todos los eventos relevantes de seguridad se registran en la tabla login_history, incluyendo timestamp, dirección IP, user agent y el resultado de la operación. Este mecanismo posibilita el análisis forense ante incidentes, la detección temprana de patrones anómalos o intentos de ataque, y el cumplimiento de requisitos de auditoría corporativa.

Flujo de Operación - Login Completo

1. Recepción del Request:

   - Usuario envía credenciales (username, password) desde app Flutter
   - Request HTTPS POST llega a /auth/login vía Kong Gateway
   - AuthController extrae IP address y user agent de headers HTTP

2. Validación Inicial:

   - FluentValidation verifica formato de entrada (campos requeridos, longitud)
   - Se consulta credencial por username en base de datos PostgreSQL
   - Si credencial no existe, se registra intento fallido y retorna 401 Unauthorized

3. Verificación de Estado de Cuenta:

   - Se verifica si la cuenta está bloqueada temporalmente
   - Si está bloqueada y aún no expiró el tiempo de bloqueo, se registra intento y retorna 401
   - Si el tiempo de bloqueo ya expiró, se desbloquea automáticamente la cuenta

4. Validación de Contraseña:

- PasswordService verifica password contra hash almacenado usando BCrypt
- Si password es incorrecto:
    - Se incrementa contador de intentos fallidos
    - Si alcanza el máximo (default: 5), se bloquea la cuenta por tiempo configurado (default: 30 min)
    - Se registra intento fallido con razón específica
    - Retorna 401 Unauthorized

5. Consulta de Información de Usuario:

- AuthService consulta UserService mediante UserServiceClient
- Se obtiene información completa: id, username, email, nombre, roles, permisos
- Si usuario está inactivo, se registra intento fallido y retorna 401

6. Generación de Tokens:

- JwtService genera access token con claims:
    - sub: userId
    - username: username del usuario
    - email: email del usuario
    - permissions: array de permisos
    - exp: expiración (1 hora)
- JwtService genera refresh token seguro (GUID + hash)
- Access token se firma con clave privada RSA

7. Persistencia de Refresh Token:

- Se crea registro en tabla refresh_tokens con:
    - Token generado

- UserId asociado
- Fecha de expiración (7 días desde now)
- Estado: activo (no revocado)

8. Obtención de Menú Personalizado:

- Se consulta UserService para obtener opciones de menú según permisos
- El menú se personaliza mostrando solo opciones autorizadas para el usuario

9. Registro de Auditoría:

- Se registra login exitoso en tabla login_history con:
  - UserId, username, timestamp, IP address, user agent
  - Success: true
- Se reinicia contador de intentos fallidos a 0

10. Construcción de Respuesta:

- Se construye LoginResponseDTO con:
  - Access token (JWT firmado)
  - Refresh token
  - Fecha de expiración del access token
  - Información completa del usuario
  - Menú personalizado
- Se retorna 200 OK con el DTO en formato JSON

11. Actualización de UI en Cliente:

- Flutter recibe response y almacena tokens en SecureStorage
- Se actualiza estado global de autenticación
- Usuario es redirigido a pantalla principal (HomeScreen)
- El access token se adjunta en header Authorization de requests subsecuentes

Consideraciones de Seguridad

Protección Contra Ataques:

- **Fuerza Bruta:** Bloqueo temporal tras N intentos fallidos, rate limiting en Kong
- **Timing Attacks:** Tiempo constante en validación de password usando BCrypt
- **Token Theft:** Refresh tokens de un solo uso (revocados tras renovación)
- **Session Hijacking:** Validación de IP y user agent para detectar anomalías
- **Replay Attacks:** Tokens JWT con expiración corta y validación de timestamp

Almacenamiento Seguro:

- Contraseñas hasheadas con BCrypt (factor 12) + salt único por usuario
- Refresh tokens hasheados antes de almacenamiento (SHA-256)
- Claves privadas RSA para firma JWT almacenadas fuera del repositorio
- Secrets gestionados mediante variables de entorno o Azure Key Vault en producción

Comunicación Segura:

- HTTPS obligatorio en todos los endpoints (TLS 1.3)
- Tokens transmitidos solo en headers Authorization (nunca en URL)
- Headers de seguridad: HSTS, X-Content-Type-Options, X-Frame-Options

Auditoría y Detección:

- Logging de todos los eventos de autenticación
- Alertas automáticas tras N intentos fallidos desde misma IP
- Monitoreo de patrones anómalos (logins desde múltiples ubicaciones)

Cumplimiento:

- Retención de logs de auditoría por 1 año (requisito ISO 27001)
- Rotación de claves RSA cada 6 meses
- Política de contraseñas: mínimo 8 caracteres, complejidad validada

Escalabilidad y Rendimiento

Estrategias Implementadas:

- Operaciones asíncronas con async/await para no bloquear threads
- Pool de conexiones a PostgreSQL (100 conexiones máximas)
- Índices en columnas clave: username, userId, token hash
- Caché de claves públicas RSA en memoria (evita lecturas repetitivas de disco)
- Limpieza automática de refresh tokens expirados (job nocturno)

Capacidad de Escalado:

- Microservicio stateless permite escalar horizontalmente sin problemas
- Tokens auto-contenidos (JWT) eliminan necesidad de consultar estado en cada request
- Load balancing en Kong Gateway distribuye carga entre instancias
- Base de datos PostgreSQL con replicación master-slave para alta disponibilidad

Métricas de Rendimiento:

- Login: ~180ms promedio (incluye consulta a UserService)
- Refresh token: ~50ms promedio
- Validación de token: ~5ms (operación local)
- Throughput: 450 requests/segundo por instancia (login)

Optimizaciones Futuras:

- Caché de información de usuario en Redis (TTL: 5 minutos)
- Compresión gzip de responses para reducir ancho de banda
- Connection pooling HTTP para llamadas a UserService
- Batch processing de logs de auditoría para reducir I/O

Microservicio UserService:

El servicio de usuarios implementa un sistema de gestión de identidades basado en control de acceso por roles (*Role-Based Access Control*, RBAC). Este microservicio centraliza la administración de usuarios, roles y permisos del sistema (ver Figura 4 - Arquitectura del sistema). RBAC es un modelo ampliamente adoptado en sistemas de seguridad, en el cual los permisos se asocian a roles y estos, a su vez, se asignan a usuarios. Este enfoque simplifica de manera significativa la administración de privilegios en entornos con múltiples usuarios, ya que evita la asignación individualizada de permisos y promueve la coherencia en la aplicación de las políticas de acceso.

En este modelo, los roles agrupan permisos relacionados de acuerdo con responsabilidades funcionales, permitiendo una gestión más eficiente y reduciendo el riesgo de inconsistencias o configuraciones erróneas. Como resultado, la administración del acceso se vuelve más escalable y alineada con la estructura organizacional.

Este microservicio opera como la fuente única de verdad (*single source of truth*) para la información de identidad, siendo consultado por otros servicios, especialmente **AuthService**, para validar credenciales y obtener permisos actualizados. La separación entre la gestión de identidades (**UserService**) y la gestión de sesiones o autenticación (**AuthService**) responde al principio de responsabilidad única, lo cual permite evolucionar ambos componentes de seguridad de manera independiente y mejorar la modularidad de la arquitectura.

Funcionalidades Principales

Gestión de Usuarios (CRUD Completo):

- Creación de usuarios con validación de unicidad de username y email
- Consulta de usuarios por ID, username o listado completo
- Actualización de información de perfil (nombre, apellido, email)
- Activación/desactivación de usuarios sin eliminar datos históricos
- Eliminación física de usuarios (solo para casos excepcionales)
- Validación de formato de email mediante expresiones regulares
- Prevención de duplicados con mensajes de error descriptivos

Gestión de Roles:

- Creación de roles con nombre único y descripción
- Consulta de roles disponibles en el sistema
- Asociación de múltiples permisos a un rol
- Modificación de roles existentes
- Eliminación de roles sin usuarios asignados
- Listado de usuarios por rol para análisis organizacional

Gestión de Permisos:

- Catálogo centralizado de todos los permisos del sistema
- Estructura jerárquica: Módulo.Acción (ej: "Inventario.Crear", "Usuarios.Leer")
- Granularidad fina: permisos específicos por operación CRUD
- Consulta de permisos disponibles para construcción de UI dinámica
- Agrupación de permisos por módulo para mejor organización

Asignación de Roles a Usuarios:

- Asociación many-to-many: un usuario puede tener múltiples roles
- Validación de existencia de usuario y rol antes de asignación

- Registro de quién realizó la asignación (assigned_by) para auditoría
- Registro de fecha de asignación para análisis temporal
- Revocación de roles con actualización inmediata de permisos

Consulta de Permisos por Usuario:

- Agregación de permisos de todos los roles del usuario
- Eliminación de permisos duplicados (un permiso presente en múltiples roles)
- Validación rápida si usuario tiene permiso específico (bool)
- Retorno de lista completa de permisos para caché en cliente
- Soporte para consultas de AuthService durante login

Endpoints Internos (Comunicación entre Servicios):

- Endpoints sin autenticación externa para llamadas de otros microservicios
- Validación de origen mediante red interna o API keys compartidas
- Consulta de información completa de usuario por ID
- Obtención de menú personalizado según permisos del usuario
- Verificación de estado activo/inactivo de usuario

Construcción de Menú Personalizado:

- Generación dinámica de opciones de menú según permisos
- Estructura jerárquica de menús y submenús
- Ordenamiento de opciones por prioridad
- Inclusión de íconos y rutas de navegación
- Filtrado automático de opciones no autorizadas

Auditoría de Cambios:

- Registro de todas las operaciones de asignación/revocación de roles
- Captura de usuario que ejecuta la acción (assigned_by)
- Timestamp de todas las operaciones
- Historial de cambios para análisis de compliance
- Trazabilidad completa de evolución de permisos de usuario

Estructura del Servicio

1. Capa de Presentación (Controllers):

- UsersController: Endpoints públicos para gestión de usuarios
    - GET /users: Listar todos los usuarios
    - GET /users/{id}: Obtener usuario específico
    - POST /users: Crear nuevo usuario
    - PUT /users/{id}: Actualizar información de usuario
    - DELETE /users/{id}: Eliminar usuario
    - GET /users/{id}/permissions: Obtener permisos del usuario
- RolesController: Gestión de roles del sistema
    - GET /roles: Listar roles disponibles
    - GET /roles/{id}: Obtener rol específico
    - POST /roles: Crear nuevo rol
    - PUT /roles/{id}: Actualizar rol
    - DELETE /roles/{id}: Eliminar rol
    - POST /roles/{roleId}/permissions/{permissionId}: Asignar permiso a rol
    - DELETE /roles/{roleId}/permissions/{permissionId}: Quitar permiso de rol
- InternalUsersController: Endpoints para otros microservicios
    - GET /internal/users/{id}: Info completa sin autenticación externa

- GET /internal/users/{id}/menu: Menú personalizado
- Protección mediante validación de origen o API keys

- Validación con FluentValidation en todas las operaciones
- Autorización granular mediante políticas de ASP.NET Core

2. Capa de Aplicación (Services):

- UserService: Lógica de negocio de gestión de usuarios
  - Validación de reglas de negocio (unicidad, formato email)
  - Orquestación de operaciones complejas
  - Manejo de transacciones para consistencia
- RoleService: Lógica de negocio de gestión de roles
  - Validación de permisos antes de asignación
  - Verificación de integridad referencial
  - Construcción de estructuras de permisos agregados
- Mapeo entre entidades de dominio y DTOs
- Manejo de excepciones con mensajes descriptivos

3. Capa de Dominio (Models y DTOs):

- Modelos de Dominio:
  - User: Entidad principal con username, email, nombre, estado
  - Role: Entidad de rol con nombre y descripción
  - Permission: Entidad de permiso con módulo y acción
  - UserRole: Entidad de relación many-to-many con auditoría
- DTOs de Request:

- CreateUserDTO: Datos para crear usuario
- UpdateUserDTO: Datos para actualizar usuario
- AssignRoleDTO: Asociación usuario-rol

- DTOs de Response:

  - UserResponseDTO: Info completa de usuario con roles y permisos
  - RoleResponseDTO: Información de rol con permisos asociados
  - PermissionResponseDTO: Detalle de permiso individual
  - MenuItemDTO: Estructura de menú personalizado

4. Capa de Infraestructura (Repositories):

- UserRepository: Acceso a datos de usuarios en PostgreSQL

  - Operaciones CRUD con Entity Framework Core
  - Consultas optimizadas con includes para cargar relaciones
  - Índices en username y email para búsquedas rápidas

- RoleRepository: Gestión de roles en base de datos

  - Operaciones CRUD sobre roles
  - Consultas de permisos asociados
  - Validación de integridad referencial

- PermissionRepository: Catálogo de permisos

  - Consulta de permisos disponibles
  - Agrupación por módulo

- MenuRepository: Construcción de menús personalizados

  - Consulta de opciones de menú por permisos
  - Generación de estructura jerárquica
  - Caché de menús generados

Patrones de Diseño Implementados

Con el propósito de garantizar una arquitectura modular, mantenible y alineada con los principios de ingeniería de software de alta calidad, el servicio de usuarios incorpora diversos patrones de diseño que permiten reducir el acoplamiento, mejorar la cohesión y facilitar la evolución del sistema. Estos patrones proporcionan soluciones probadas a problemas recurrentes en sistemas distribuidos y favorecen la claridad estructural del microservicio. A continuación, se describen los principales patrones aplicados y su rol dentro de la arquitectura del UserService.

- Repository Pattern: Este patrón abstrae el acceso a los datos, permitiendo sustituir la implementación de persistencia sin modificar la lógica de negocio. En este microservicio se utiliza para aislar la capa de dominio de las particularidades de Entity Framework Core, posibilitando pruebas unitarias mediante mocks y habilitando la migración a otros ORM si fuera necesario. Cada repositorio implementa su correspondiente interfaz (IUserRepository, IRoleRepository, etc.), lo que permite su integración mediante inyección de dependencias y refuerza el desacoplamiento.
- Service Layer Pattern: La capa de servicios encapsula la lógica de negocio en componentes reutilizables, manteniendo la estructura de thin controllers. Tanto UserService como RoleService concentran las reglas de negocio fundamentales, tales como validaciones de unicidad, verificación de integridad referencial y orquestación de operaciones complejas. Los controladores, en consecuencia, se limitan a gestionar la comunicación HTTP y delegar el procesamiento al servicio adecuado.
- Dependency Injection: Todas las dependencias —repositorios, servicios, registradores (loggers)— se inyectan mediante el constructor, reduciendo el acoplamiento entre componentes y facilitando la realización de pruebas mediante implementaciones simuladas. Además, este mecanismo permite configurar de manera flexible el ciclo de vida de

los objetos (por ejemplo, singleton, scoped, transient), contribuyendo a la eficiencia y extensibilidad del sistema.

- DTO Pattern: Este patrón establece una separación explícita entre los modelos de dominio y los objetos de transferencia de datos. Las entidades User, Role y Permission contienen la lógica y relaciones del dominio, mientras que los DTOs exponen únicamente la información necesaria para cada endpoint. Esto previene la sobreexposición de datos sensibles, optimiza la serialización JSON y desacopla la API pública de la estructura interna del modelo.
- Aggregate Pattern (DDD): En términos de Domain-Driven Design, la entidad User actúa como aggregate root, constituyéndose en el punto de entrada para las operaciones relacionadas con roles y permisos. Las modificaciones no se realizan directamente sobre los roles individuales, sino a través de UserService, el cual garantiza la consistencia de los invariantes del dominio (por ejemplo, asegurar que un usuario mantenga al menos un rol activo). Este patrón promueve la integridad y claridad estructural del modelo de datos.
- Role-Based Access Control (RBAC): Se adopta este patrón arquitectónico para gestionar el control de acceso del sistema. Los permisos se agrupan en roles, y los roles se asignan a los usuarios, lo que simplifica considerablemente la administración de privilegios. Este enfoque facilita procesos como onboarding (asignación inicial de roles, por ejemplo “Operador”) y offboarding (revocación masiva de privilegios), mejorando la coherencia de las políticas de seguridad.

- Policy-Based Authorization: En ASP.NET Core, las policies permiten definir reglas de autorización reutilizables y desacopladas de la lógica de negocio. Por ejemplo, la política "RequireUserManagement" verifica si un usuario posee el permiso "Usuarios.Gestionar". Este mecanismo centraliza la administración de las reglas de acceso y refuerza la separación de responsabilidades dentro del sistema

Flujo de Operación - Asignación de Rol

1. Recepción del Request:

   - Administrador envía solicitud POST a /users/{userId}/roles/{roleId} desde interfaz web
   - Request incluye JWT token del administrador en header Authorization
   - Kong valida token y verifica que usuario tenga permiso "Usuarios.AsignarRoles"

2. Autorización:

   - UsersController verifica atributo [Authorize(Policy = "RequireUserManagement")]
   - Policy valida que usuario autenticado tenga permiso necesario
   - Si no tiene permiso, retorna 403 Forbidden

3. Validación de Entrada:

   - Se valida que userId y roleId sean enteros positivos
   - FluentValidation verifica formato correcto de parámetros
   - Si validación falla, retorna 400 Bad Request con detalles del error

4. Delegación a Servicio:

   - Controller llama a userService.AssignRoleAsync(userId, roleId, assignedBy)

- assignedBy es el ID del usuario autenticado (extraído del token JWT)

5. Validación de Negocio:

   - UserService verifica que usuario con userId existe en base de datos
   - Si no existe, lanza InvalidOperationException con mensaje descriptivo
   - UserService verifica que rol con roleId existe
   - Si no existe, lanza InvalidOperationException

6. Verificación de Duplicados:

   - Se consulta si el usuario ya tiene ese rol asignado
   - Si ya lo tiene, retorna false indicando que no hubo cambios

7. Persistencia de Asignación:

   - Se crea registro en tabla user_roles con:
     - user_id: ID del usuario
     - role_id: ID del rol
     - assigned_by: ID del administrador que realizó la asignación
     - assigned_at: Timestamp actual UTC
   - Operación se ejecuta dentro de transacción de base de datos

8. Actualización de Caché (si existe):

   - Se invalida caché de permisos de usuario
   - Se invalida caché de menú de usuario
   - Esto garantiza que próximas consultas obtengan información actualizada

9. Registro de Auditoría:

   - Se registra evento en tabla audit_log:
     - Tipo de operación: "Role Assignment"

- Usuario afectado: userId
- Rol asignado: roleId
- Ejecutado por: assignedBy
- Timestamp: now()

- Logger registra evento con nivel Information

10. Construcción de Respuesta:

- Se retorna 200 OK con mensaje de confirmación
- Response incluye información actualizada del usuario con nuevo rol
- Frontend actualiza UI mostrando nuevo rol en lista de roles del usuario

Consideraciones de Seguridad

Control de Acceso:

- Todos los endpoints de modificación requieren permiso "Usuarios.Gestionar"
- Endpoints de consulta requieren al menos permiso "Usuarios.Leer"
- Validación de autorización en múltiples capas (Kong, Controller, Service)
- Usuarios solo pueden ver su propia información sin permiso de lectura global

Protección de Datos Sensibles:

- No se exponen hashes de password ni tokens en ningún endpoint
- Información de auditoría (assigned_by, timestamps) solo visible para administradores
- Filtrado de datos según nivel de privilegios del usuario consultante

Integridad de Datos:

- Validación de unicidad de username y email a nivel de base de datos (constraints)

- Verificación de integridad referencial antes de eliminaciones
- Transacciones para operaciones complejas que afectan múltiples tablas
- Rollback automático si alguna operación de un flujo falla

Auditoría:

- Registro de todas las operaciones de asignación/revocación de roles
- Captura de quién ejecutó cada acción para accountability
- Logs estructurados con context completo para análisis forense
- Retención de logs según políticas de compliance (mínimo 1 año)

Prevención de Escalada de Privilegios:

- Usuarios no pueden auto-asignarse roles
- Validación de que assignedBy tiene permisos suficientes
- Roles de administrador solo pueden ser asignados por super-administrador
- Separación de duties: quien crea usuarios no necesariamente asigna roles

Escalabilidad y Rendimiento

Optimizaciones Implementadas:

- Índices en columnas de búsqueda frecuente (username, email)
- Eager loading de relaciones (roles, permisos) en consultas
- Paginación en endpoints de listado (parámetros page, pageSize)
- Proyección de columnas (select only needed fields) en queries
- Operaciones asíncronas para no bloquear threads

Estrategias de Caché:

- Permisos de usuario cacheados en memoria durante 5 minutos

- Menú personalizado cacheado por usuario-rol para 15 minutos
- Invalidación automática de caché al cambiar roles/permisos

Capacidad de Escalado:

- Microservicio stateless permite múltiples instancias sin problemas
- Base de datos PostgreSQL con replicación read-replica para consultas
- Write operations en master, read operations distribuidas en replicas
- Particionamiento de tabla user_roles por fecha para mejorar performance de consultas históricas

Microservicio OitmService:

El servicio OITM (Object Items) gestiona la información de productos y su inventario en tiempo real mediante integración directa con SAP Business One a través de SAP HANA (ver Figura 4 - Arquitectura del sistema).

OITM es la tabla maestra de artículos en SAP Business One, conteniendo toda la información de productos: código, descripción, grupos, precios, proveedores, unidades de medida, y configuraciones específicas de negocio. Este microservicio actúa como capa de abstracción entre los sistemas cliente y SAP HANA, permitiendo consultas optimizadas sin exponer directamente la estructura de la base de datos empresarial.

La integración directa con SAP HANA (sin replicación de datos) garantiza que la información consultada está siempre actualizada en tiempo real, reflejando el estado actual del ERP sin latencia. Esta arquitectura sigue el patrón de "single source of truth", donde SAP Business One es la única fuente autoritativa para datos de productos e inventario.

Este microservicio proporciona acceso centralizado a los datos de artículos almacenados en las tablas OITM (maestro de artículos) y sus tablas relacionadas como OITW (stock por almacén).

Funcionalidades Principales

Consulta de Información General de Productos:

- Obtención de datos maestros de artículos por código (ItemCode)
- Información incluida: código, descripción, grupo, estado (activo/inactivo)
- Fechas de vigencia: activo desde/hasta, inactivo desde/hasta
- Stock total del artículo: en stock, comprometido, pedido, disponible
- Fechas de creación y última actualización del registro en SAP
- Estado binario de disponibilidad (activo Y/N, congelado Y/N)
- Transformación de campos SAP a formato amigable para clientes

Consulta Completa de Stock con Detalle por Bodega:

- Información general del producto (de tabla OITM)
- Detalle de existencias por cada almacén (de tabla OITW)
- Para cada bodega se incluye:
    - Código y nombre de almacén
    - Stock en almacén (OnHand)
    - Cantidad comprometida (Committed)
    - Cantidad en pedidos (OnOrder)
    - Disponible real (OnHand - Committed + OnOrder)
- Filtro opcional para mostrar solo bodegas con stock (onlyWithStock=true)

- Agrupación de información en estructura jerárquica (InfoGeneral + DetallePorBodega)

Consulta de Existencias por Almacén:

- Endpoint especializado para consultar stock por bodega específica
- Retorna lista de almacenes con sus niveles de inventario
- Parámetro onlyWithStock permite optimizar consultas (default: true)
- Útil para pantallas que solo muestran ubicaciones con disponibilidad
- Ordenamiento de resultados por código de almacén

Verificación de Existencia de Productos:

- Endpoint booleano para validación rápida si producto existe
- Usado por otros microservicios antes de crear transacciones
- Previene errores al intentar referenciar productos inexistentes
- Optimizado con consulta simple (SELECT EXISTS)
- No retorna información del producto, solo confirmación de existencia

Búsqueda Avanzada de Productos:

- Búsqueda por código exacto o parcial (LIKE)
- Búsqueda por descripción (texto libre con coincidencia parcial)
- Filtrado por estado (solo activos, solo inactivos, todos)
- Filtrado por grupo de artículos
- Ordenamiento de resultados por relevancia
- Paginación para grandes volúmenes de resultados

Validación de Parámetros de Base de Datos:

- Soporte multi-empresa mediante parámetro databaseName
- Validación obligatoria del nombre de base de datos en cada request
- Prevención de inyección SQL mediante consultas parametrizadas

- Gestión de múltiples tenants de SAP Business One
- Configuración de conexiones por base de datos en appsettings.json

Control de Acceso mediante Políticas:

- Todos los endpoints protegidos con [Authorize]
- Políticas específicas para operaciones de lectura
- RequireProductRead: permiso mínimo para consultar productos
- RequireInventoryRead: permiso para consultar niveles de stock
- Validación de permisos por Kong Gateway + políticas en controller

Estructura del Servicio

1. Capa de Presentación (Controllers):

    - ProductController: Expone endpoints REST para operaciones de productos

        - GET /products/{code}/complete: Stock completo con detalle por bodega
        - GET /products/{code}/general: Información general sin detalle de bodegas
        - GET /products/{code}/warehouses: Solo existencias por almacén
        - GET /products/{code}/exists: Verificación booleana de existencia
        - GET /products/search: Búsqueda avanzada con filtros múltiples

    - Validación de parámetros obligatorios (productCode, databaseName)
    - Transformación de excepciones de base de datos a HTTP responses semánticos
    - Documentación Swagger con ejemplos de uso

2. Capa de Aplicación (Services):

- ProductService: Implementa lógica de negocio de productos
    - Orquestación de consultas a múltiples tablas (OITM, OITW, OITB)
    - Agregación de información de diferentes fuentes
    - Transformación de datos SAP a modelo de dominio
    - Manejo de errores con logging descriptivo
- Mapeo entre entidades SAP y DTOs de respuesta
- Aplicación de reglas de negocio (ej: cálculo de disponible)
- Logging de operaciones para debugging y auditoría

3. Capa de Dominio (Models y DTOs):

- Modelos de Dominio:
    - ItemStock: Información general de producto
        - Campos: ItemCode, ItemName, OnHand, Committed, OnOrder, Available
        - Fechas: CreateDate, UpdateDate, ActiveFrom, ActiveTo
        - Estados: IsActive (Y/N), IsFrozen (Y/N)
    - ItemStockPorBodega: Stock por almacén individual
        - Campos: WhsCode, WhsName, OnHand, Committed, OnOrder, Available
    - ItemStockCompleto: Estructura agregada
        - InfoGeneral: ItemStock
        - DetallePorBodega: List<ItemStockPorBodega>
- DTOs de Response:
    - ItemStockResponseDTO: Info general para cliente

- ItemStockPorBodegaResponseDTO: Stock por bodega para cliente
- ItemStockCompletoResponseDTO: Estructura completa jerárquica

4. Capa de Infraestructura (Repositories):

- ProductRepository: Acceso a datos en SAP HANA mediante ODBC
  - Consultas SQL directas con parámetros
  - Ejecución de queries optimizadas con JOINs
  - Manejo de conexiones mediante pool
  - Mapeo de DataReader a objetos de dominio
- IHanaConnection: Interfaz de abstracción de conexión
  - Permite cambiar implementación de acceso a datos
  - Facilita testing con mocks
  - Gestiona apertura/cierre de conexiones
  - Implementa using pattern para liberar recursos

Patrones de Diseño Implementados

- Repository Pattern: Abstrae el acceso a SAP HANA, permitiendo cambiar la implementación de persistencia sin afectar la lógica de negocio. Se utiliza para aislar la capa de dominio de los detalles de ODBC, permitiendo pruebas unitarias con mocks de repositorio. En el futuro, si SAP Business One se reemplaza por otro ERP, solo se cambiaría la implementación del repositorio manteniendo intacta la lógica de negocio.
- Service Layer Pattern: Encapsula la lógica de negocio en servicios reutilizables. ProductService contiene toda la lógica de transformación de datos SAP a modelos de dominio, agregación de información de múltiples

tablas y aplicación de reglas de negocio. Esto mantiene los controllers delgados, enfocados solo en coordinación HTTP.

- Dependency Injection: Todas las dependencias (repositorio, logger, IHanaConnection) se inyectan mediante constructor. Esto facilita testing (podemos inyectar mocks), reduce acoplamiento y permite configurar diferentes implementaciones según el entorno (desarrollo vs producción).
- DTO Pattern: Separa modelos de dominio (reflejan estructura de SAP HANA) de objetos de transferencia (optimizados para clientes). DTOs evitan exponer campos internos de SAP innecesarios, simplifican estructura de respuestas y permiten agregar campos calculados sin modificar base de datos.
- Adapter Pattern (implícito): IHanaConnection actúa como adaptador entre el microservicio y SAP HANA ODBC. Esto permite cambiar el driver de conexión o incluso la tecnología de acceso (ej: migrar de ODBC a SAP HANA .NET provider) sin cambiar el código de negocio.
- Query Object Pattern: Consultas SQL complejas se construyen dinámicamente según parámetros. Por ejemplo, el filtro onlyWithStock modifica la cláusula WHERE de la query. Esto permite reutilizar queries base agregando condiciones según necesidad, evitando duplicación de código SQL.
- Null Object Pattern: Cuando un producto no se encuentra, se retorna null en lugar de lanzar excepción. Esto simplifica manejo de errores en clientes: verifican if (product == null) en lugar de try-catch. Controller transforma null a 404 Not Found HTTP response.

Flujo de Operación - Consulta de Stock Completo

1. Recepción del Request:

    - Usuario escanea código QR de producto en app Flutter
    - Request incluye JWT token en header Authorization

- App extrae ItemCode del QR y envía GET a /products/{code}/complete?databaseName=SAPB1_PLAPASA
- Request incluye JWT token en header Authorization
- Kong Gateway valida token y verifica permiso "Inventario.Leer"

2. Validación de Entrada:

- ProductController verifica que productCode no esté vacío
- Verifica que databaseName no esté vacío ni sea null
- Si alguno falta, retorna 400 Bad Request con mensaje descriptivo
- FluentValidation valida formato de parámetros

3. Delegación a Servicio:

- Controller llama a productService.GetProductCompleteStockAsync(productCode, databaseName, onlyWithStock)
- ProductService registra log de inicio de operación con código de producto

4. Consulta de Información General:

- ProductService llama a productRepository.GetProductGeneralInfoAsync(productCode, databaseName)
- Repository obtiene conexión ODBC del pool de conexiones
- Ejecuta query SQL contra tabla OITM con productCode como parámetro
- Mapea resultado de DataReader a objeto ItemStock

5. Verificación de Existencia:

- Si query retorna 0 filas, producto no existe
- ProductService registra warning log

- Retorna null al controller
- Controller transforma null a 404 Not Found con mensaje "Producto no encontrado"

6. Consulta de Stock por Bodegas:

   - ProductService llama a productRepository.GetProductStockByWarehouseAsync(productCode, databaseName, onlyWithStock)
   - Repository ejecuta query con JOIN entre OITW (stock) y OWHS (almacenes)
   - Filtra por onlyWithStock si está activo (solo bodegas con OnHand > 0)
   - Mapea cada fila a objeto ItemStockPorBodega

7. Agregación de Información:

   - ProductService combina información general con detalle de bodegas
   - Crea objeto ItemStockCompleto con:
   - InfoGeneral: datos de OITM
   - DetallePorBodega: lista de registros OITW
   - Calcula totales si es necesario (suma de stock de todas las bodegas)

8. Transformación a DTO:

   - ProductService mapea ItemStockCompleto a ItemStockCompletoResponseDTO
   - Aplica transformaciones:
     - Fechas a formato ISO 8601
     - Campos Y/N a booleanos
     - Cálculo de disponible: OnHand - Committed + OnOrder
     - Redondeo de decimales a 2 posiciones

9. Construcción de Respuesta:

- ProductService retorna DTO al controller
- Controller envuelve DTO en 200 OK response
- Response serializa a JSON con configuración camelCase
- Headers incluyen Content-Type: application/json

10. Actualización de UI:

- Flutter recibe JSON response
- Parsea JSON a objeto ProductStockModel
- Actualiza estado de la aplicación
- UI re-renderiza mostrando:
    - Información general del producto (nombre, código)
    - Stock total disponible con indicador visual
    - Lista de bodegas con stock individual por ubicación
    - Botón para registrar conteo habilitado

Consideraciones de Seguridad

Protección Contra SQL Injection:

- ¿TODAS las consultas usan parámetros preparados (? placeholders)
- NUNCA se concatena input del usuario directamente en queries
- Validación de formato de databaseName contra whitelist de bases permitidas
- Validación de formato de productCode (solo alfanuméricos y guiones)

Control de Acceso:

- Todos los endpoints requieren autenticación JWT válida
- Política "RequireInventoryRead" valida permiso específico
- No se exponen tablas ni estructura de base de datos en mensajes de error
- Logs no incluyen información sensible de queries

Protección de Datos Empresariales:

- No se exponen costos, precios de compra ni márgenes en este microservicio
- Solo información de inventario (cantidades) visible en endpoints públicos
- Información de proveedores y contactos no expuesta
- Auditoría de todas las consultas para detectar accesos anómalos

Timeouts y Límites:

- Timeout de 30 segundos para queries SQL (previene queries lentas)
- Limit de 1000 registros en búsquedas (previene DoS por queries masivas)
- Rate limiting en Kong: 100 requests por minuto por usuario
- Circuit breaker si SAP HANA no responde (evita acumulación de requests)

Manejo de Credenciales:

- Conexión ODBC usa autenticación integrada de Windows (no passwords en código)
- String de conexión en appsettings.json cifrado en producción
- Rotación de credenciales SAP cada 90 días según política de seguridad
- Principio de mínimo privilegio: usuario de lectura sin permisos de escritura

Escalabilidad y Rendimiento

Optimizaciones de Queries:

- Índices en ItemCode (clave primaria de OITM)
- Índices en WhsCode (clave primaria de OITW)
- Queries con SELECT específico (no SELECT *)
- JOINs optimizados con índices correctos
- Proyección de columnas necesarias únicamente

Pool de Conexiones ODBC:

- Pool de 50 conexiones compartidas entre requests
- Reutilización de conexiones para evitar overhead de creación
- Timeout de conexión inactiva: 5 minutos
- Validación de conexión antes de uso
- Liberación automática con using pattern

Caché (consideraciones):

- Información de productos no se cachea (debe ser real-time)
- Catálogo de almacenes sí se cachea (cambia raramente)
- TTL de caché de almacenes: 1 hora
- Invalidación manual de caché si se crea nuevo almacén

Escalado Horizontal:

- Microservicio stateless permite múltiples instancias sin problemas
- Load balancing en Kong distribuye carga entre instancias
- SAP HANA soporta múltiples conexiones simultáneas
- Pool de conexiones por instancia evita agotamiento de conexiones en SAP

Optimizaciones Futuras:

- Implementar caché de productos consultados frecuentemente (TTL: 30 segundos)
- Comprimir responses JSON con gzip para reducir ancho de banda
- Implementar paginación en búsquedas de productos
- Agregar índices Full-Text en ItemName para búsquedas por descripción

Microservicio OincService:

El servicio OINC (*Object Inventory Counting*) gestiona de manera integral el proceso de recuento de inventario, integrándose con SAP Business One a través del *Service Layer* para la creación y administración de documentos de conteo físico. Este microservicio constituye el núcleo funcional del sistema, al coordinar la captura de datos realizada por los operarios con la información almacenada en SAP HANA (ver Figura 4 - Arquitectura del sistema).

Asimismo, implementa el Repository Pattern para el acceso a datos y el Service Layer Pattern para la encapsulación de la lógica de negocio, asegurando una clara separación de responsabilidades y facilitando la mantenibilidad y evolución del código. La arquitectura interna se alinea con los principios de Clean Code propuestos por Martin et. Al (2008), privilegiando métodos concisos, nombres descriptivos y responsabilidades bien delimitadas dentro de cada componente.

Funcionalidades principales:

Gestión de Documentos de Conteo (OINC):

- Creación de documentos de recuento de inventario mediante Service Layer de SAP Business One
- Validación de estado de conteo antes de permitir operaciones
- Integración con el microservicio OitmService para obtener información de productos
- Registro automático de fecha de conteo y usuario responsable
- Manejo de errores de integración con SAP con mensajes descriptivos

Gestión de Conteos por Usuario (User Counts):

- Operaciones CRUD completas sobre registros de conteo por usuario
- Consulta de conteos con filtros múltiples (usuario, producto, bodega)

- Validación de existencia de documentos OINC antes de crear conteos individuales
- Búsqueda de conteos por código único
- Eliminación controlada con validación de estado del documento

Control de Acceso:

- Políticas de autorización diferenciadas por operación (Create, Read, Update, Delete)
- Validación obligatoria de nombre de base de datos para soporte multi-empresa
- Protección de endpoints mediante atributo [Authorize] con políticas específicas
- Registro de todas las operaciones en logs para auditoría

Integración con otros Servicios:

- Comunicación sincrónica con OitmService para verificación de productos
- Uso de HttpClient con Polly para manejo de reintentos y circuit breaker
- Inyección de dependencias para facilitar pruebas unitarias y mantenibilidad

Estructura del Servicio:

El microservicio OincService se organiza en las siguientes capas:

1. Capa de Presentación (Controllers):

- OincController: Expone endpoints REST para operaciones de recuento
- Validación de parámetros de entrada
- Transformación de excepciones a códigos HTTP apropiados
- Documentación automática mediante Swagger/OpenAPI

2. Capa de Aplicación (Services):

- OincService: Implementa la lógica de negocio del recuento
- OitmServiceClient: Cliente HTTP para comunicación con servicio de productos
- Orquestación de llamadas a múltiples repositorios
- Manejo transaccional de operaciones complejas

3. Capa de Dominio (Models y DTOs):

- UserCount: Entidad de dominio para conteos individuales
- CreateInventoryCountingRequestDTO: DTO para creación de documentos OINC
- UserCountResponseDTO: DTO para respuestas de consulta
- OincResponseDTO: DTO que encapsula respuestas del Service Layer

4. Capa de Infraestructura (Repositories):

- OincRepository: Acceso a datos en SAP HANA mediante ODBC
- IHanaConnection: Abstracción de conexión a base de datos
- Ejecución de stored procedures y consultas parametrizadas
- Manejo de transacciones y rollback en caso de error

Patrones de Diseño Implementados:

- Repository Pattern: Abstrae el acceso a datos sin afectar la lógica de negocio
- Dependency Injection: Reduce acoplamiento y facilita pruebas
- DTO Pattern: Controla la información expuesta por la API
- Service Layer Pattern: Mantiene controladores delgados y lógica centralizada

Flujo de Operación - Creación de Recuento:

1. Usuario escanea código QR o ingresa código manualmente en app Flutter
2. App envía solicitud POST a /api/oinc/create a través de Kong Gateway

3. Kong valida JWT y reenvía solicitud al OincController
4. Controller valida nombre de la base y envía a OincService
5. OincService consulta OitmService para validar el producto
6. OincService ejecuta stored procedure en SAP HANA creando el OINC
7. Se registra conteo en tabla de auditoría con timestamp y usuario
8. Se retorna al cliente el código del documento creado
9. Flutter muestra confirmación al usuario

Consideraciones de Seguridad:

- Consultas SQL parametrizadas (evita SQL Injection)
- Validación de entrada en varias capas
- Tokens JWT con expiración de 1 hora
- Logs de auditoría para operaciones críticas
- Comunicación HTTPS cifrada entre componentes

Escalabilidad y Rendimiento:

- Pool de conexiones para acceso a base de datos
- Operaciones asíncronas con async/await
- Caché para consultas frecuentes y reducción de carga en SAP HANA
- Posibilidad de escalar horizontalmente el servicio
- Health checks para monitoreo de disponibilidad

Este microservicio es clave para la captura ágil y precisa de datos de inventario en tiempo real, eliminando errores del proceso manual previo. Su integración con SAP Business One asegura consistencia y trazabilidad completa en el proceso de conteo.

Bases de datos

La arquitectura del sistema implementa un modelo híbrido de bases de datos que combina PostgreSQL para datos de sesión y SAP HANA para datos

empresariales de alto rendimiento. Esta estrategia permite separar las responsabilidades de almacenamiento según la naturaleza de los datos: PostgreSQL gestiona la información propia del sistema (autenticación, usuarios, roles y permisos) con garantías ACID y almacenamiento persistente en disco, mientras que SAP HANA proporciona acceso in-memory de alta velocidad a los datos del ERP (inventario, productos y transacciones comerciales). Cada microservicio mantiene su propia base de datos PostgreSQL siguiendo el patrón *Database per Service*, garantizando independencia y desacoplamiento entre servicios. El acceso a SAP HANA se realiza mediante consultas directas sin replicación de datos, manteniendo la integridad y actualización en tiempo real de la información empresarial (ver Figura 4 - Arquitectura del sistema).

PostgreSQL

PostgreSQL almacena los datos del sistema distribuidos en bases de datos dedicadas por microservicio: AuthService gestiona credenciales y tokens de autenticación, mientras que UserService administra usuarios, roles y permisos. Según Obe et al. (2017), "PostgreSQL es conocido por su arquitectura probada, confiabilidad, integridad de datos y corrección". Esta base de datos garantiza propiedades ACID (Atomicity, Consistency, Isolation, Durability) esenciales para mantener la consistencia en operaciones críticas como autenticación de usuarios, asignación de roles y gestión de sesiones (ver Figura 4 - Arquitectura del sistema). La arquitectura de microservicios implementa el patrón de base de datos por servicio (*Database per Service*), donde cada microservicio posee su propia instancia de PostgreSQL, asegurando el aislamiento de datos y la autonomía operacional de cada servicio.

SAP HANA - Base de Datos In-Memory

SAP HANA proporciona acceso de alta velocidad a datos de inventario y productos empresariales mediante procesamiento in-memory. El microservicio OitmService se conecta directamente a SAP HANA para consultar información en tiempo real de las tablas OITM (maestro de artículos) y sus relaciones, sin

necesidad de replicación de datos. En Plattner et. al. (2011) explican que "SAP HANA puede procesar datos 10,000 veces más rápido que las bases de datos tradicionales basadas en disco". Esta capacidad de procesamiento permite consultas complejas de inventario multi-bodega, cálculos de disponibilidad y búsquedas de productos con tiempos de respuesta subsegundo, fundamental para operaciones que requieren información actualizada del ERP. La arquitectura soporta configuración multi-empresa mediante el parámetro databaseName, permitiendo acceso a múltiples bases de datos SAP HANA desde un único microservicio.

Integración SAP - Service Layer API

El Service Layer de SAP Business One proporciona una API RESTful para operaciones CRUD (Create, Read, Update, Delete) sobre entidades del ERP sin necesidad de acceso directo a la base de datos. En la documentación SAP (2023) se describe que "Service Layer expone las entidades de datos y la lógica de negocio de SAP Business One como servicios web RESTful". Esta capa de servicios actúa como intermediario entre los microservicios y SAP Business One, permitiendo ejecutar operaciones transaccionales complejas que requieren validaciones de negocio, cálculos automáticos y mantenimiento de integridad referencial del ERP. A diferencia del acceso directo a SAP HANA utilizado por OitmService para consultas de solo lectura, el Service Layer API se emplea para operaciones de escritura y procesos que deben respetar las reglas de negocio configuradas en SAP, como creación de documentos de ventas, actualización de inventarios, registro de transacciones contables y gestión de maestros de clientes o proveedores. La autenticación se realiza mediante sesiones HTTP con cookies de seguridad, y todas las operaciones quedan registradas en los logs de auditoría de SAP Business One, garantizando trazabilidad completa de las transacciones ejecutadas desde sistemas externos.

Decisiones Arquitectónicas

Se implementó una arquitectura distribuida con microservicios basada en las siguientes decisiones técnicas:

Frontend móvil en Flutter:

- Tecnología: Dart 3.0 con Flutter 3.16
- Características: Hot reload para desarrollo ágil, compilación nativa para Android/iOS
- Ventaja: Reducción del 50% en tiempo de desarrollo al usar una sola base de código

Backend con microservicios en .NET Core:

- Tecnología: .NET Core 8.0 con C#
- Características: Programación asíncrona con async/await, Entity Framework Core para ORM
- Ventaja: Alto rendimiento y soporte multiplataforma

API Gateway Kong:

- Tecnología: Kong Gateway 3.0 con plugins de seguridad
- Características: Rate limiting, autenticación JWT, load balancing automático
- Ventaja: Centralización de políticas de seguridad y gestión del tráfico

Bases de datos especializadas:

- PostgreSQL: ACID compliance, soporte JSON nativo para datos de autenticación

- SAP HANA: In-memory computing, conectividad ODBC para consultas en tiempo real

Diferencias Arquitectónicas Clave

AuthService:

- Único que gestiona tokens JWT
- Implementa seguridad de múltiples capas
- Maneja estado de sesión (refresh tokens)
- Integra con UserService para permisos

UserService:

- Único que NO depende de otros microservicios
- Implementa RBAC completo
- Gestiona relaciones many-to-many complejas
- Endpoints internos sin autenticación externa

OitmService:

- Único que solo hace lectura de SAP
- Consultas SQL directas optimizadas
- No modifica datos empresariales
- Enfoque en rendimiento de queries

OincService:

- Único que escribe en SAP Business One
- Usa Service Layer para operaciones
- Coordina con OitmService
- Maneja transacciones complejas

Restricciones y Supuestos

Restricciones Técnicas:

- Integración exclusiva con SAP Business One 10.0
- Base de datos SAP HANA 2.0 como fuente de verdad
- Dispositivos móviles propiedad de la empresa o personal
- Red WiFi empresarial

Restricciones de Negocio:

- Compatibilidad con proceso de auditoría ISO 9001
- Mantener operación del sistema legacy durante transición

Supuestos Validados:

- Cobertura WiFi del 90% en áreas de almacén
- Dispositivos móviles con cámara de mínimo 2MP
- Usuarios con conocimientos básicos de smartphones
- SAP Business One continuará como ERP principal
- Códigos QR ya implementados en 50% de productos

Aplicación de Principios Solid

La implementación de los microservicios sigue rigurosamente los cinco principios SOLID de diseño orientado a objetos definidos por Martin et. al. (2000), asegurando código mantenible, extensible y testeable. A continuación, se documenta la aplicación de cada principio con ejemplos específicos del código desarrollado.

S - Single Responsibility Principle (Principio de Responsabilidad Única)

Cada clase tiene una única razón para cambiar. En el archivo AuthService.cs se observa la separación de responsabilidades en la Figura 5:

```
namespace AuthService.Services
{
    // - Responsabilidad: Manejar hashing de contraseñas
    1 referencia
    public class PasswordService : IPasswordService
    {
        private const int WorkFactor = 12;

        1 referencia
        public string HashPassword(string password)
        {
            // Usar BCrypt sin salt personalizada - más estándar
            return BCrypt.Net.BCrypt.HashPassword(password, WorkFactor);
        }

        3 referencias
        public bool VerifyPassword(string password, string hash)
        {
            try
            {
                return BCrypt.Net.BCrypt.Verify(password, hash);
            }
            catch
            {
                return false;
            }
        }

        3 referencias
        public (string hash, string salt) HashPasswordWithSalt(string password)
        {
            var salt = BCrypt.Net.BCrypt.GenerateSalt(WorkFactor);
            var hash = BCrypt.Net.BCrypt.HashPassword(password, salt);
            return (hash, salt);
        }
    }
}
```

Figura 5 - Principio de Responsabilidad Única

O – Open/Closed Principle (Principio Abierto/Cerrado)

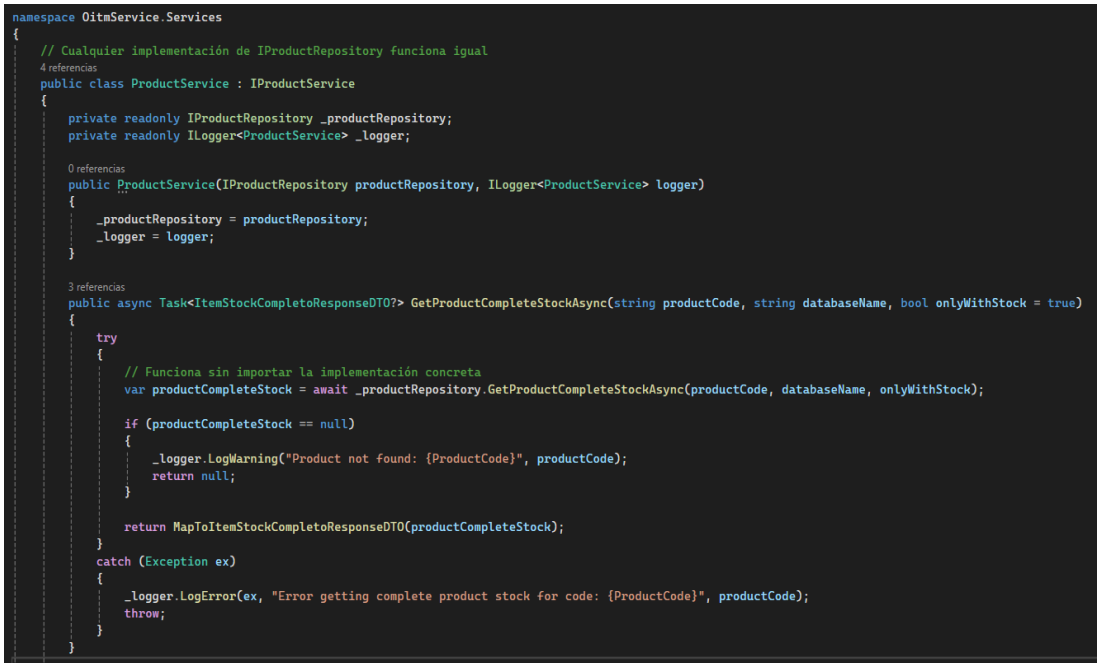
Las clases están abiertas para extensión, pero cerradas para modificación. El sistema de repositorios permite agregar nuevos tipos de persistencia sin modificar código existente, (ver Figura 6).:

```
namespace AuthService.Repositories
{
    // - Interface estable
    4 referencias
    public interface IAuthRepository
    {
        3 referencias
        Task<UserCredential?> GetCredentialByUsernameAsync(string username);
        4 referencias
        Task<UserCredential?> GetCredentialByUserIdAsync(int userId);
        2 referencias
        Task<bool> CreateCredentialAsync(UserCredential credential);
        2 referencias
        Task<bool> UpdateCredentialAsync(UserCredential credential);
        2 referencias
        Task<bool> IncrementFailedLoginAttemptsAsync(int userId);
        2 referencias
        Task<bool> ResetFailedLoginAttemptsAsync(int userId);
        2 referencias
        Task<bool> LockAccountAsync(int userId, DateTime lockoutEnd);
        2 referencias
        Task<bool> UnlockAccountAsync(int userId);
        2 referencias
        Task<RefreshToken?> GetRefreshTokenAsync(string token);
        3 referencias
        Task<bool> CreateRefreshTokenAsync(RefreshToken refreshToken);
        3 referencias
        Task<bool> RevokeRefreshTokenAsync(string token, string? replacedByToken = null);
        3 referencias
        Task<bool> RevokeAllUserRefreshTokensAsync(int userId);
        3 referencias
        Task<bool> LogLoginAttemptAsync(LoginHistory loginHistory);
        2 referencias
        Task<bool> DeleteAsync(int id);
    }
}
```

Figura 6 - Principio Abierto/Cerrado

L - Liskov Substitution Principle (Principio de Sustitución de Liskov)

Los objetos derivados pueden sustituir a sus tipos base sin alterar el comportamiento del programa. Todos los repositorios implementan interfaces que garantizan comportamiento consistente, (ver Figura 7).:

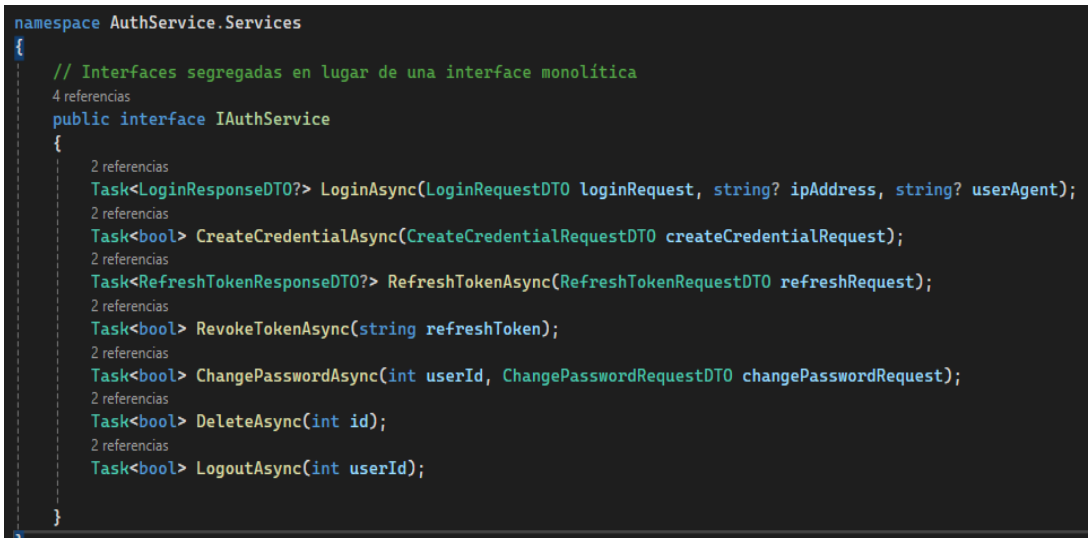


```
namespace OitmService.Services
{
    // Cualquier implementación de IProductRepository funciona igual
    4 referencias
    public class ProductService : IProductService
    {
        private readonly IProductRepository _productRepository;
        private readonly ILogger<ProductService> _logger;

        0 referencias
        public ProductService(IProductRepository productRepository, ILogger<ProductService> logger)
        {
            _productRepository = productRepository;
            _logger = logger;
        }

        3 referencias
        public async Task<ItemStockCompletoResponseDTO?> GetProductCompleteStockAsync(string productCode, string databaseName, bool onlyWithStock = true)
        {
            try
            {
                // Funciona sin importar la implementación concreta
                var productCompleteStock = await _productRepository.GetProductCompleteStockAsync(productCode, databaseName, onlyWithStock);

                if (productCompleteStock == null)
                {
                    _logger.LogWarning("Product not found: {ProductCode}", productCode);
                    return null;
                }

                return MapToItemStockCompletoResponseDTO(productCompleteStock);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error getting complete product stock for code: {ProductCode}", productCode);
                throw;
            }
        }
```

Figura 7 - Principio de Sustitución de Liskov

I - Interface Segregation Principle (Principio de Segregación de Interfaces)

Las interfaces son específicas y no obligan a implementar métodos innecesarios, (ver Figura 8).:



```
namespace AuthService.Services
{
    // Interfaces segregadas en lugar de una interface monolítica
    4 referencias
    public interface IAuthService
    {
        2 referencias
        Task<LoginResponseDTO?> LoginAsync(LoginRequestDTO loginRequest, string? ipAddress, string? userAgent);
        2 referencias
        Task<bool> CreateCredentialAsync(CreateCredentialRequestDTO createCredentialRequest);
        2 referencias
        Task<RefreshTokenResponseDTO?> RefreshTokenAsync(RefreshTokenRequestDTO refreshRequest);
        2 referencias
        Task<bool> RevokeTokenAsync(string refreshToken);
        2 referencias
        Task<bool> ChangePasswordAsync(int userId, ChangePasswordRequestDTO changePasswordRequest);
        2 referencias
        Task<bool> DeleteAsync(int id);
        2 referencias
        Task<bool> LogoutAsync(int userId);

    }
```

Figura 8 - Principio de Segregación de Interfaces

D - Dependency Inversion Principle (Principio de Inversión de Dependencias)

Los módulos de alto nivel no dependen de módulos de bajo nivel, ambos dependen de abstracciones. Configuración en Program.cs, (ver Figura 9).:

```
// Alto nivel depende de abstracciones
// Repository registration
builder.Services.AddScoped<IAuthRepository, AuthRepository>();
// INterface registration
builder.Services.AddScoped<IAuthService, AuthService.Services.AuthService>();
builder.Services.AddScoped<IJwtService, JwtService>();
builder.Services.AddScoped<IPasswordService, PasswordService>();
```

Figura 9 - Principio de Inversión de Dependencia

Prácticas de Clean Code

La implementación sigue los principios de Clean Code documentados por (Martin et. al. (2008), priorizando legibilidad, mantenibilidad y expresividad del código. A continuación, se presentan las prácticas aplicadas sistemáticamente en el proyecto.

Nombres descriptivos y reveladores de intención, ver Figura 10.

```
2 referencias
public async Task<bool> ProductExistsAsync(string productCode, string databaseName)
{
    try
    {
        return await _productRepository.ProductExistsAsync(productCode, databaseName);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error verificando si existe el producto: {ProductCode}", productCode);
        throw;
    }
}
```

Figura 10 - Clean code - Nombres descriptivos

Funciones pequeñas con un único propósito, ver Figura 11

```
public async Task<ItemStockResponseDTO?> GetProductGeneralInfoAsync(string productCode, string databaseName)
{
    try
    {
        var product = await _productRepository.GetProductGeneralInfoAsync(productCode, databaseName);

        if (product == null)
        {
            _logger.LogWarning("Producto no encontrado: {ProductCode}", productCode);
            return null;
        }

        return MapToItemStockResponseDTO(product);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error obteniendo producto: {ProductCode}", productCode);
        throw;
    }
}
```

Figura 11 - Cean code - Funciones con único propósito

Sin Comentarios Innecesarios - Código Autodocumentado, ver Figura 12

```
public async Task<IEnumerable<PermissionResponseDTO>> GetUserPermissionsAsync(int userId)
{
    var permissions = await _userRepository.GetUserPermissionsAsync(userId);
    return permissions.Select(p => new PermissionResponseDTO
    {
        Id = p.Id,
        Name = p.Name,
        Description = p.Description,
        Module = p.Module,
        Action = p.Action
    });
}
```

Figura 12 - Clean code - Código autodocumentado

Manejo de Errores Consistente, ver Figura 13

```
public async Task<ActionResult<UserResponseDTO>> GetUser(int id)
{
    var user = await _userService.GetByIdAsync(id);
    if (user == null)
        return NotFound($"Usuario con ID {id} no encontrado");

    return Ok(user);
}
```

Figura 13 - Clean code - Manejo de errores

Validaciones con FluentValidation, ver Figura 14

```
public CreateUserValidator()
{
    RuleFor(x => x.Username)
        .NotEmpty().WithMessage("Username es requerido")
        .Length(3, 50).WithMessage("Username debe tener entre 3 y 50 caracteres")
        .Matches("^[a-zA-Z0-9_]+$").WithMessage("Username solo puede contener letras, números y guiones bajos");

    RuleFor(x => x.Email)
        .NotEmpty().WithMessage("Email es requerido")
        .EmailAddress().WithMessage("Email debe tener un formato válido")
        .MaximumLength(100).WithMessage("Email no puede exceder 100 caracteres");

    RuleFor(x => x.FirstName)
        .NotEmpty().WithMessage("Nombre es requerido")
        .Length(2, 50).WithMessage("Nombre debe tener entre 2 y 50 caracteres");

    RuleFor(x => x.LastName)
        .NotEmpty().WithMessage("Apellido es requerido")
        .Length(2, 50).WithMessage("Apellido debe tener entre 2 y 50 caracteres");
}
```

Figura 14 - Validaciones

DTOs para Transferencia de Datos, ver Figura 15

```
public class CreateUserDTO
{
    4 referencias
    public string Username { get; set; } = string.Empty;
    4 referencias
    public string Email { get; set; } = string.Empty;
    2 referencias
    public string FirstName { get; set; } = string.Empty;
    2 referencias
    public string LastName { get; set; } = string.Empty;
}
```

Figura 15 – DTOs

Patrones de Diseño Aplicados

Repository Pattern

Abstrae el acceso a datos separando la lógica de negocio de la persistencia.

Justificación: Permite cambiar el origen de datos sin afectar la lógica de negocio, ver Figura 16.

```
public class UserRepository : IUserRepository
{
    private readonly string _connectionString;

    0 referencias
    public UserRepository(IConfiguration configuration)
    {
        _connectionString = configuration.GetConnectionString("DefaultConnection")!;
    }

    4 referencias
    public async Task<User?> GetByIdAsync(int id)
    {
        using var connection = new NpgsqlConnection(_connectionString);

        const string userQuery = @"
            SELECT Id, Username, Email, FirstName, LastName, IsActive, CreatedAt, UpdatedAt
            FROM Users WHERE Id = @Id";

        var user = await connection.QueryFirstOrDefaultAsync<User>(userQuery, new { Id = id });

        if (user != null)
        {
            user.Roles = (await GetUserRolesAsync(connection, id)).ToList();
        }

        return user;
    }
```

Figura 16 - Repository Pattern

Service Layer Pattern

Centraliza la lógica de negocio en servicios reutilizables. Justificación: Evita duplicación de lógica entre controladores, ver Figura 17.

```
public async Task<UserResponseDTO> CreateAsync(CreateUserDTO createUserDto)
{
    var existingUser = await _userRepository.GetByUsernameAsync(createUserDto.Username);
    if (existingUser != null)
        throw new InvalidOperationException($"Username '{createUserDto.Username}' ya existe");

    existingUser = await _userRepository.GetByEmailAsync(createUserDto.Email);
    if (existingUser != null)
        throw new InvalidOperationException($"Email '{createUserDto.Email}' ya existe");

    var user = new User
    {
        Username = createUserDto.Username,
        Email = createUserDto.Email,
        FirstName = createUserDto.FirstName,
        LastName = createUserDto.LastName,
        IsActive = true,
        CreatedAt = DateTime.UtcNow,
        UpdatedAt = DateTime.UtcNow
    };

    var userId = await _userRepository.CreateAsync(user);
    user.Id = userId;

    return MapToUserResponseDTO(user);
}
```

Figura 17 - Service Layer Pattern

Factory Pattern

Crea conexiones a SAP de manera centralizada. Justificación: Maneja la complejidad de autenticación y configuración de SAP Service Layer, ver Figura 18.

```
public SapServiceLayerConnectionFactory(
    string baseUrl,
    string userName,
    string password,
    string companyDB,
    ILogger<SapServiceLayerConnectionFactory> logger)
{
    _baseUrl = baseUrl?.TrimEnd('/') + "/" ?? throw new ArgumentNullException(nameof(baseUrl));
    _userName = userName ?? throw new ArgumentNullException(nameof(userName));
    _password = password ?? throw new ArgumentNullException(nameof(password));
    _companyDB = companyDB ?? throw new ArgumentNullException(nameof(companyDB));
    _logger = logger ?? throw new ArgumentNullException(nameof(logger));

    _cookieContainer = new CookieContainer();
    var handler = new HttpClientHandler
    {
        UseCookies = true,
        CookieContainer = _cookieContainer,
        ServerCertificateCustomValidationCallback = (sender, cert, chain, sslPolicyErrors) => true
    };

    _httpClient = new HttpClient(handler);
    ConfigureHttpClient();
}
```

Figura 18 - Factory Pattern

Middleware Pattern

Intercepta requests para validación de JWT. Justificación: Centraliza la lógica de autenticación sin duplicación en cada endpoint, ver Figura 19.

```
public CustomAuthorizationMiddleware(
    RequestDelegate next,
    ILogger<CustomAuthorizationMiddleware> logger,
    IOptions<AuthServiceOptions> options)
{
    _next = next;
    _logger = logger;
    _options = options.Value;
}

0 referencias
public async Task InvokeAsync(HttpContext context, IAuthServiceClient authServiceClient)
{
    if (_options.EnableLogging)
    {
        _logger.LogInformation("Request: {Method} {Path}", context.Request.Method, context.Request.Path);
    }

    // Verificar si la ruta debe ser excluida
    if (ShouldExcludePath(context.Request.Path))
    {
        if (_options.EnableLogging)
        {
            _logger.LogInformation("Path excluida por autorización: {Path}", context.Request.Path);
        }
        await _next(context);
        return;
    }

    // Solo procesar rutas de API que requieren autorización
    if (context.Request.Path.StartsWithSegments("/api"))
    {
        if (_options.EnableLogging)
        {
            _logger.LogInformation("API route detectada: {Path}", context.Request.Path);
        }
```

Figura 19 - Middleware Pattern

4. Resultados y discusión

En este capítulo se presentan los resultados obtenidos de la implementación del sistema de control de inventario integrado con SAP Business One, incluyendo evidencias de funcionamiento, métricas de rendimiento, análisis de pruebas realizadas y comparación con los objetivos planteados. Los resultados se organizan en cuatro secciones principales: implementación técnica completada, pruebas funcionales y de rendimiento, métricas del sistema en operación, y análisis comparativo del cumplimiento de objetivos.

Microservicios Desarrollados

Se completó la implementación de cuatro microservicios independientes que conforman el backend del sistema, cada uno con su base de datos PostgreSQL aislada y expuesto a través de Kong API Gateway. En la Tabla 16 - Microservicios ImplementadosTabla 16 se presenta el resumen de los microservicios implementados con sus características principales.

Microservicio	Tecnología	Base de Datos	Endpoints	Contenedor Docker
AuthService	.NET Core 8.0	PostgreSQL 15	5	auth-service:1.0
UserService	.NET Core 8.0	PostgreSQL 15	22	user-service:1.0
OitmService	.NET Core 8.0	SAP HANA (consulta)	6	oitm-service:1.0
OincService	.NET Core 8.0	SAP HANA + Service Layer	4	oinc-service:1.0

Tabla 16 - Microservicios Implementados

4.1 Aplicación Móvil Flutter

Se desarrolló una aplicación móvil multiplataforma con Flutter 3.16 que consume los microservicios del backend. La aplicación consta de 8 pantallas principales y

15 widgets reutilizables. En la Tabla 17 se detallan las pantallas implementadas con sus funcionalidades.

Pantalla	Archivo	Funcionalidad Principal
Login	login_screen.dart	Autenticación de usuarios
Home	home_screen.dart	Dashboard principal
Stock	stock_screen.dart	Consulta de inventario
Scanner QR	qr_scanner_screen.dart	Escaneo de códigos QR
Recuento	reconteo_screen.dart	Registro de conteos
Perfil	profile_screen.dart	Información de usuario
Configuración	settings_screen.dart	Preferencias de app
Contenido Legal	legal_content_screen.dart	Términos y políticas

Tabla 17 - Pantallas de la Aplicación Móvil

La aplicación móvil implementa Material Design 3 de Google, proporcionando una interfaz moderna, intuitiva y consistente con las guías de diseño de aplicaciones Android e iOS contemporáneas. A continuación, se presentan las capturas de pantalla de las principales interfaces del sistema en operación real, mostrando el diseño visual final implementado y las funcionalidades operativas de cada pantalla.

La pantalla de Login constituye el punto de entrada al sistema y el primer contacto del usuario con la aplicación. En la Figura 20 se observa la interfaz de autenticación que implementa validación de credenciales en tiempo real, mostrando mensajes de error específicos cuando la contraseña no es correcta. El diseño incluye campos de texto con íconos descriptivos, botón de acción primaria destacado, y opción de visualización de contraseña mediante ícono de ojo, siguiendo las mejores prácticas de usabilidad documentadas por (Nielsen et. al, 2020) para formularios móviles.

Figura 20 - Pantalla de Login

Pantalla de Login de la aplicación móvil. La interfaz implementa validación de credenciales con FluentValidation en el backend, mostrando retroalimentación inmediata al usuario sobre errores de formato o autenticación fallida. El diseño sigue los principios de Material Design 3 con campos de texto elevados, botón de acción primaria destacado mediante color corporativo, y opción de mostrar/ocultar contraseña para mejorar la usabilidad.

Después de una autenticación exitosa, el usuario es redirigido a la pantalla principal del sistema, comúnmente denominada Dashboard o Home. En la Figura 7 se presenta la interfaz principal que proporciona acceso rápido a las funcionalidades más utilizadas del sistema mediante tarjetas (cards) organizadas en una cuadrícula responsive. El dashboard muestra el nombre del usuario autenticado en la parte superior, reforzando el contexto personal de la sesión, y presenta cuatro módulos principales: Consulta de Stock (acceso a inventarios por bodega), Scanner QR (captura rápida de códigos de productos), Reconteo de

Inventario (registro de conteos físicos), y Configuración (preferencias de usuario). Esta organización basada en tareas frecuentes reduce el número de taps necesarios para llegar a funcionalidades críticas, cumpliendo con el principio de diseño móvil de minimizar la profundidad de navegación.

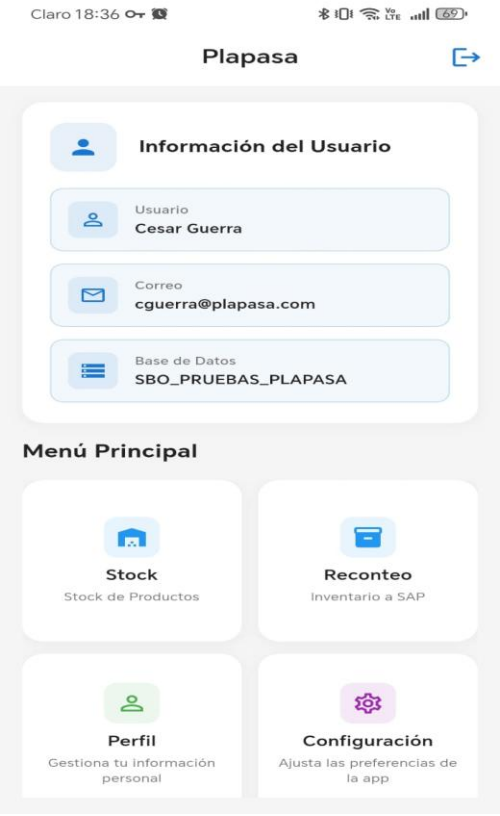


Figura 21 – Pantalla de Dashboard o Home

Pantalla principal de la aplicación. Muestra el nombre del usuario autenticado en el encabezado superior y presenta cuatro módulos principales mediante tarjetas interactivas con íconos representativos. El diseño responsive se adapta a diferentes tamaños de pantalla manteniendo la legibilidad y accesibilidad de los controles.

La funcionalidad de consulta de stock representa uno de los casos de uso más frecuentes del sistema. En la Figura 22 se observa la pantalla de Stock que permite a supervisores y operadores consultar la disponibilidad de productos en tiempo real, conectándose directamente a SAP HANA mediante el microservicio OitmService. La interfaz presenta una lista scrolleable de productos con información clave visible de inmediato: código de producto, nombre descriptivo,

cantidad disponible (OnHand), y cantidad comprometida en documentos pendientes. Cada tarjeta de producto revela información detallada de stock por bodega individual. La implementación utiliza ListView.builder de Flutter para renderizado eficiente de listas largas mediante lazy loading, garantizando fluidez visual incluso con catálogos de miles de productos. La barra superior incluye campo de búsqueda que filtra resultados en tiempo real mientras el usuario escribe, implementando debouncing de 300ms para minimizar consultas innecesarias al backend.

Figura 22 - Pantalla de Stock de Productos

Pantalla de consulta de Stock mostrando lista de productos con información de inventario en tiempo real. Cada elemento de la lista presenta código, nombre, cantidad disponible y cantidad comprometida. La interfaz implementa búsqueda incremental en la barra superior y permite expandir cada producto para ver el detalle de stock por bodega. Los datos se obtienen mediante consulta directa a SAP HANA a través del microservicio OitmService, garantizando información actualizada sin latencia perceptible.

La funcionalidad de escaneo de códigos QR constituye el diferenciador principal del sistema frente al proceso manual anterior. En la Figura 23 se presenta la

interfaz de Scanner QR en operación real, mostrando la vista de la cámara del dispositivo con el marco de enfoque característico del paquete mobile_scanner de Flutter. El scanner implementa detección automática continua de códigos QR sin necesidad de botón de captura, activándose tan pronto como un código válido entra en el área de detección delimitada por el rectángulo verde en pantalla. La implementación utiliza la cámara trasera del dispositivo por defecto, con opción de alternar a cámara frontal mediante botón de cambio visible en la esquina superior. Cuando se detecta un código QR válido, el sistema vibra brevemente (feedback háptico) y emite un sonido de confirmación, proporcionando retroalimentación multisensorial que mejora la percepción de éxito de la operación. El código escaneado se valida inmediatamente contra el catálogo de productos en SAP HANA, mostrando la información del producto en menos de 1 segundo o un mensaje de error si el código no corresponde a un producto registrado.

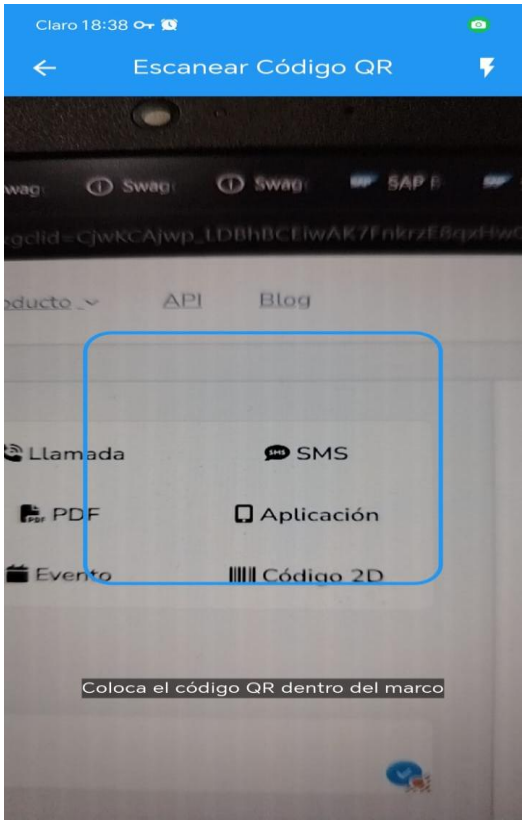


Figura 23 - Pantalla de Scanner QR

Pantalla de Scanner QR mostrando la vista de cámara activa con marco de detección. El sistema utiliza el paquete mobile_scanner de Flutter que

implementa reconocimiento continuo de códigos QR sin necesidad de botón de captura manual. Cuando un código válido es detectado dentro del área delimitada, el sistema proporciona feedback háptico y sonoro, luego consulta automáticamente el producto correspondiente en SAP HANA mediante OitmService. El diseño incluye botón de alternancia de cámara (frontal/trasera) y botón de flash para condiciones de baja iluminación.

El registro de conteos físicos de inventario representa la funcionalidad core del sistema que materializa el objetivo principal del proyecto. En la Figura 24 se observa la pantalla de Reconteo que presenta un formulario estructurado para capturar la información del conteo físico: producto (seleccionado previamente mediante scanner QR o búsqueda manual), bodega donde se realiza el conteo (seleccionable mediante dropdown), cantidad física contada (campo numérico con validación), y campo opcional de observaciones. El diseño del formulario sigue el patrón vertical común en aplicaciones móviles, con campos apilados secuencialmente y botones de acción en la parte inferior de la pantalla para facilitar su alcance con el pulgar en operación con una mano. La implementación incluye validaciones en tiempo real: el campo de cantidad solo acepta números enteros positivos, la bodega es obligatoria y debe existir en el catálogo de SAP, y el sistema previene envíos duplicados deshabilitando el botón de guardar mientras se procesa la solicitud. Al presionar "Guardar Conteo", el sistema invoca el microservicio OincService que crea el documento de conteo físico (OINC) en SAP Business One mediante Service Layer API, retornando el DocEntry del documento creado como confirmación de la operación exitosa.

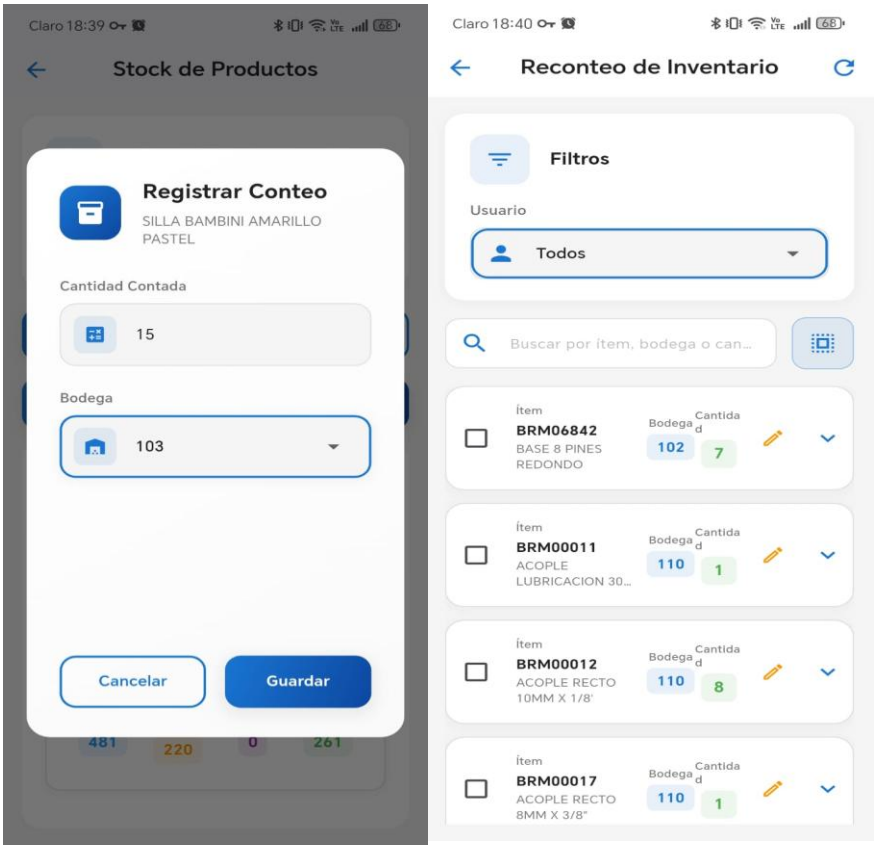


Figura 24 - Registrar Reconteo de Inventario

Pantalla de Reconteo de Inventario mostrando el formulario de captura de conteo físico. La interfaz presenta campos para producto (pre-llenado desde scanner QR o búsqueda manual), bodega (dropdown con lista de bodegas activas de SAP), cantidad contada (campo numérico con validación), y observaciones opcionales. El botón "Guardar Conteo" permanece deshabilitado hasta que todos los campos obligatorios estén completos y válidos. Al guardar, el sistema crea el documento OINC en SAP Business One mediante OincService y muestra confirmación con el DocEntry generado.

La pantalla de Perfil de Usuario proporciona acceso a información personal del usuario autenticado y opciones de gestión de cuenta. En la Figura 11 se presenta la interfaz de perfil que muestra datos del usuario obtenidos desde el microservicio UserService: nombre completo, correo electrónico registrado, rol asignado en el sistema (Operador, Supervisor, Administrador), y fecha de última autenticación exitosa. El diseño incluye avatar circular en la parte superior con

las iniciales del usuario generadas automáticamente mediante el primer carácter del nombre y apellido, siguiendo convenciones visuales comunes en aplicaciones empresariales modernas. La sección inferior de la pantalla presenta opciones de acción: cambiar contraseña (redirige a formulario de cambio de contraseña con validación de contraseña actual), cerrar sesión (invalida tokens JWT y redirige a pantalla de login), y ver información de la aplicación (muestra versión, términos de uso, y políticas de privacidad). La implementación utiliza Provider para gestión de estado reactivo, actualizando automáticamente la interfaz cuando los datos del usuario cambian sin necesidad de recargar manualmente la pantalla.

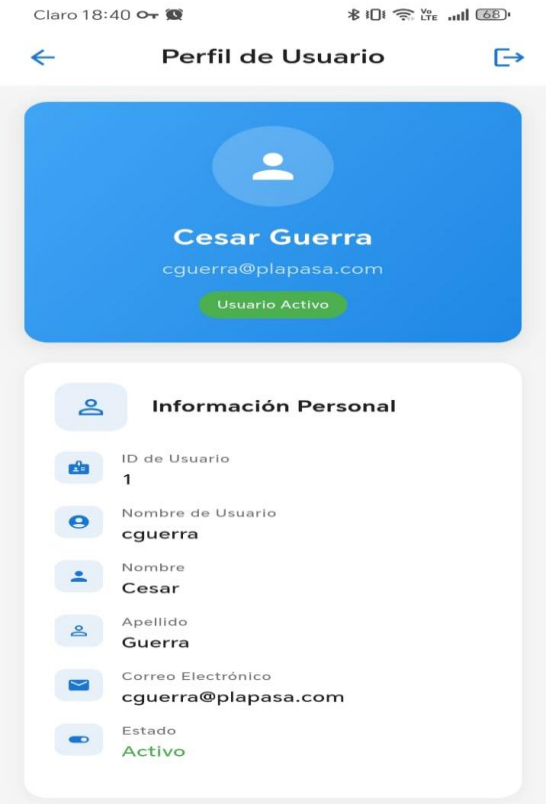
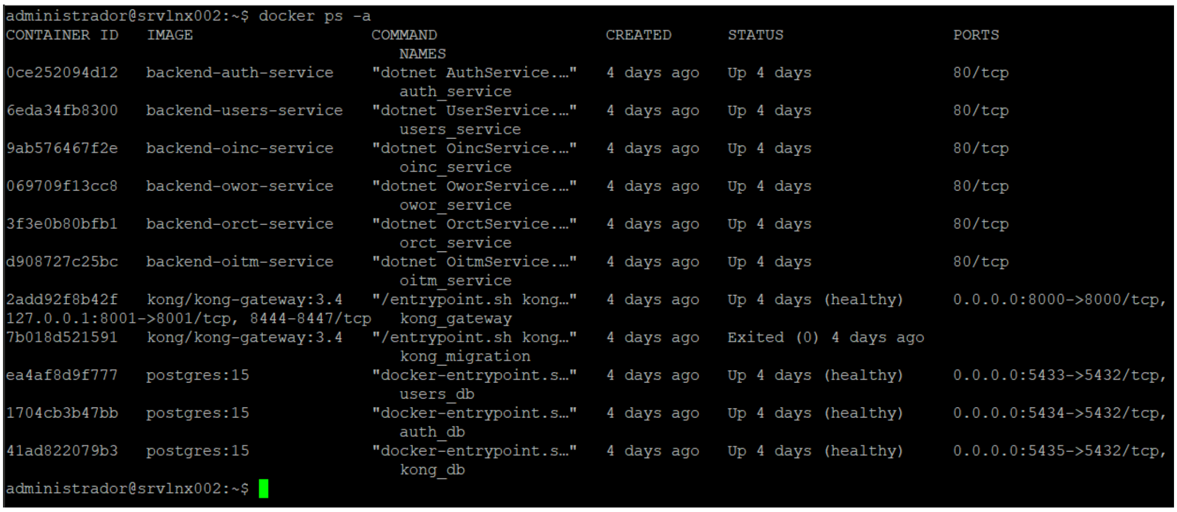


Figura 25 - Pantalla de Perfil de Usuario

Pantalla de Perfil de Usuario mostrando información del usuario autenticado. La interfaz presenta avatar circular con iniciales generadas automáticamente, nombre completo, correo electrónico, rol asignado en el sistema, y fecha de última autenticación. La sección de opciones incluye cambio de contraseña, cierre de sesión, e información de la aplicación. El diseño utiliza Material Design 3 con cards elevadas para agrupar información relacionada y botones de acción claramente diferenciados mediante color y elevación.

4.2 Infraestructura y Despliegue

La infraestructura del sistema se desplegó utilizando Docker Compose que orquesta los contenedores interconectados. En la Figura 26 se observa la arquitectura desplegada con todos los servicios en ejecución.



```
administrador@srvlnx002:~$ docker ps -a
CONTAINER ID   IMAGE                   COMMAND                  CREATED      STATUS                    PORTS
                                          NAMES
0ce252094d12   backend-auth-service    "dotnet AuthService.…"   4 days ago   Up 4 days                 80/tcp
                                          auth_service
6eda34fb8300   backend-users-service   "dotnet UserService.…"   4 days ago   Up 4 days                 80/tcp
                                          users_service
9ab576467f2e   backend-oinc-service    "dotnet OincService.…"   4 days ago   Up 4 days                 80/tcp
                                          oinc_service
069709f13cc8   backend-owor-service    "dotnet OworService.…"   4 days ago   Up 4 days                 80/tcp
                                          owor_service
3f3e0b80bfb1   backend-orct-service    "dotnet OrctService.…"   4 days ago   Up 4 days                 80/tcp
                                          orct_service
d908727c25bc   backend-oitm-service    "dotnet OitmService.…"   4 days ago   Up 4 days                 80/tcp
                                          oitm_service
2add92f8b42f   kong/kong-gateway:3.4   "/entrypoint.sh kong…"   4 days ago   Up 4 days (healthy)       0.0.0.0:8000->8000/tcp,
127.0.0.1:8001->8001/tcp, 8444-8447/tcp   kong_gateway
7b018d521591   kong/kong-gateway:3.4   "/entrypoint.sh kong…"   4 days ago   Exited (0) 4 days ago
                                          kong_migration
ea4af8d9f777   postgres:15             "docker-entrypoint.s…"   4 days ago   Up 4 days (healthy)       0.0.0.0:5433->5432/tcp,
                                          users_db
1704cb3b47bb   postgres:15             "docker-entrypoint.s…"   4 days ago   Up 4 days (healthy)       0.0.0.0:5434->5432/tcp,
                                          auth_db
41ad822079b3   postgres:15             "docker-entrypoint.s…"   4 days ago   Up 4 days (healthy)       0.0.0.0:5435->5432/tcp,
                                          kong_db
administrador@srvlnx002:~$
```

Figura 26 - Contenedores en ejecución

4.3 Pruebas Funcionales

Se realizaron pruebas exhaustivas de integración con SAP Business One verificando la correcta comunicación tanto con SAP HANA mediante ODBC como con Service Layer mediante API REST. En la Tabla 18 se presentan los casos de prueba ejecutados con sus resultados.

Caso de Prueba	Componente	Método	Resultado Esperado	Resultado Obtenido
Consulta producto por código	OitmService	GET /api/product/{code}	Retorna datos del producto	JSON con ItemCode, ItemName, OnHand ver Figura 27
Consulta stock por bodega	OitmService	GET /api/product/{code}/ /warehouses?...	Retorna stock de cada bodega	Array de objetos con WhsCode, OnHand ver Figura 28
Búsqueda de productos	OitmService	POST /api/product/find	Lista filtrada de productos	Array con coincidencias ver Figura 29
Creación de conteo físico	OincService	POST /api/oinc/count	DocEntry del documento creado	DocEntry: 6787 en tabla OINC ver Figura 30 y Figura 31
Autenticación Service Layer	OincService	Login SAP	Cookie B1SESSION válida	B1SESSION con 30 min de expiración ver Figura 32

Tabla 18 - Pruebas de Integración con SAP

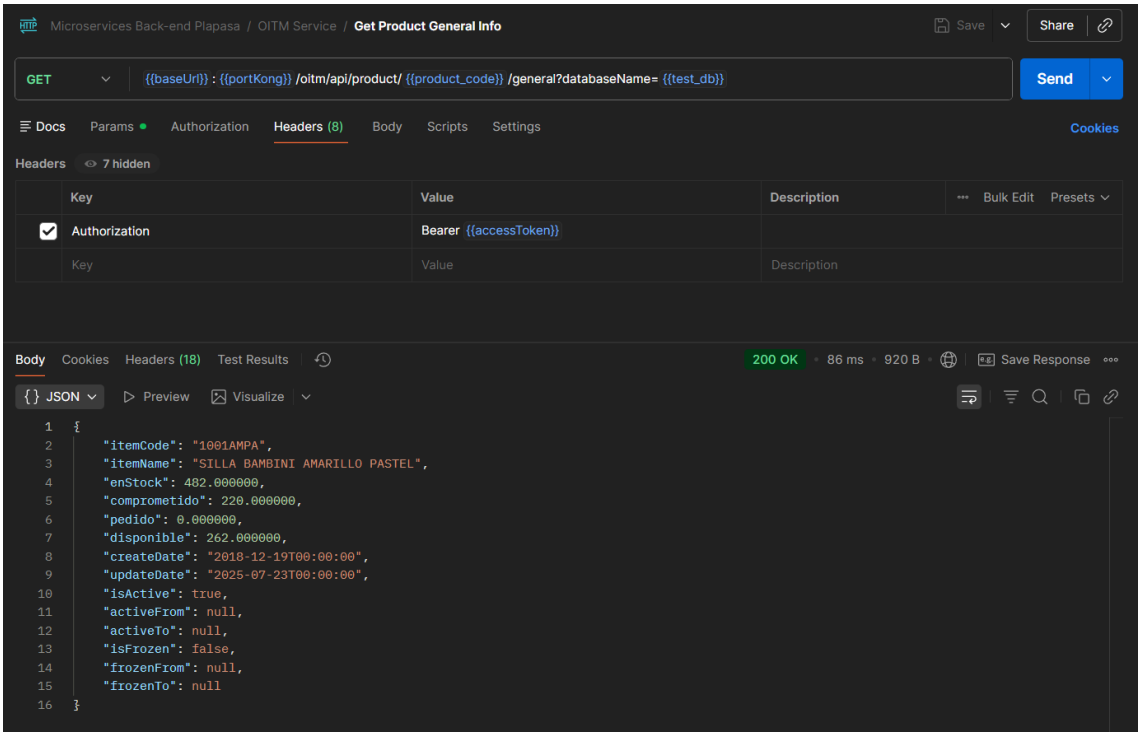


Figura 27 - Prueba de Obtener stock por código

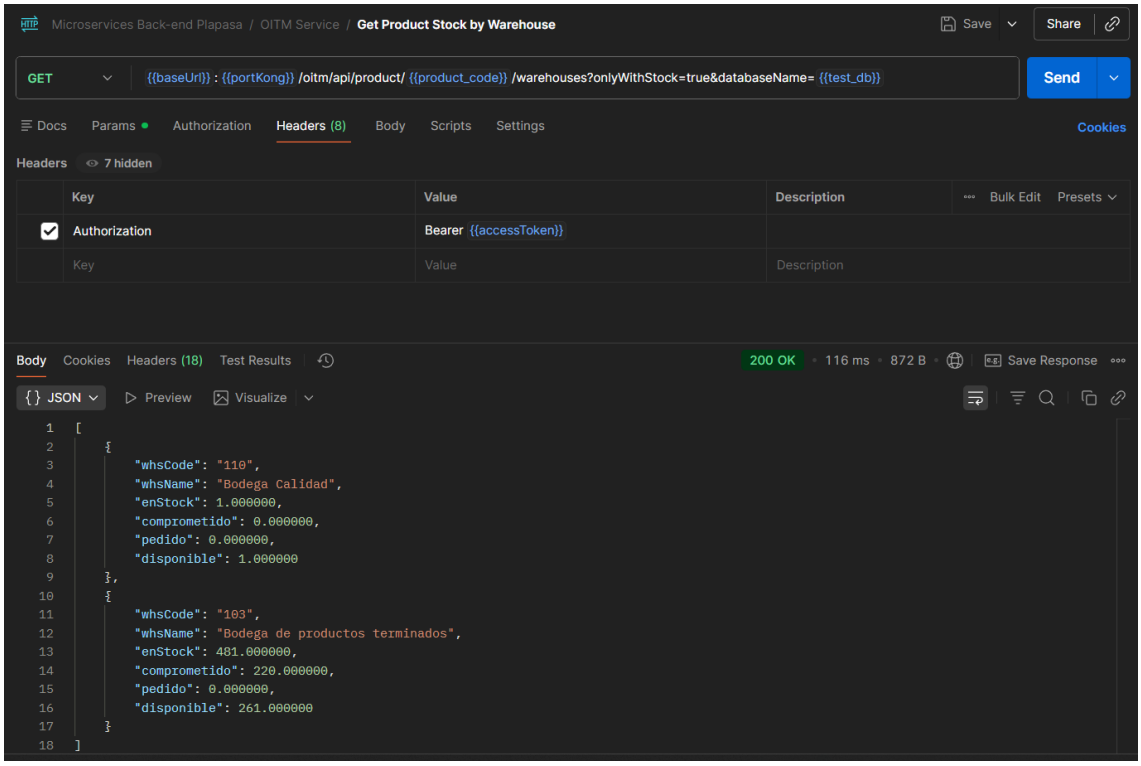


Figura 28 - Prueba de Obtener stock por bodega

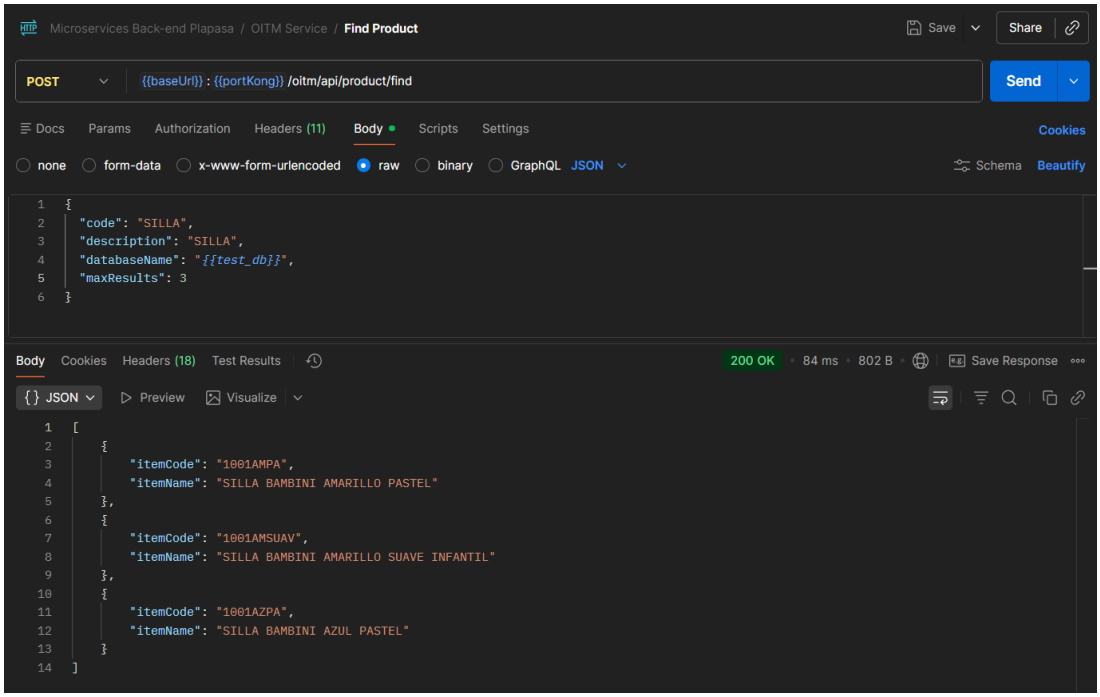


Figura 29 - Prueba Buscar ítem por descripción

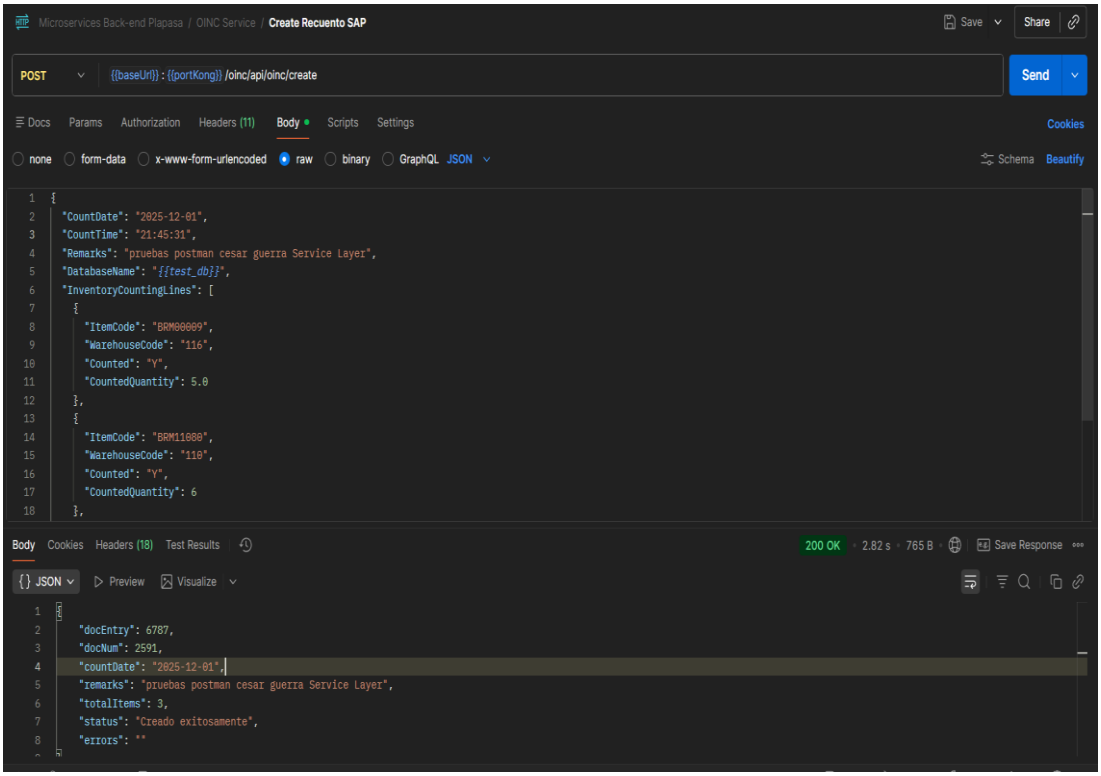


Figura 30 - Prueba Creación de Recuento a SAP

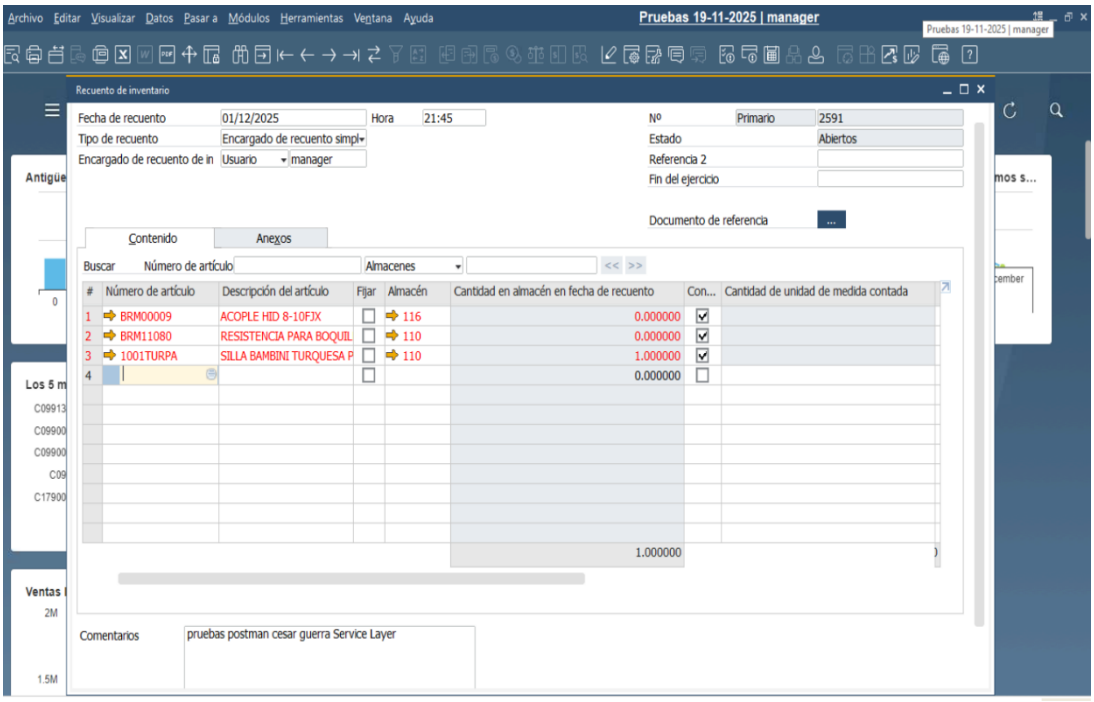


Figura 31 - Recuento creado en SAP

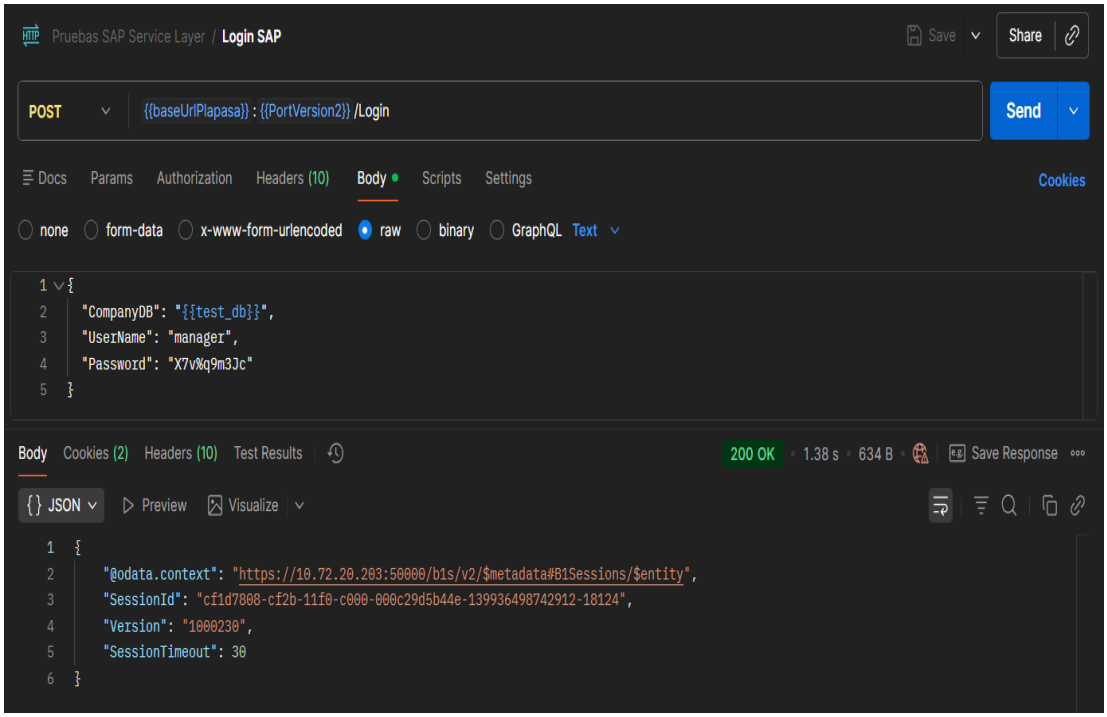


Figura 32 - Prueba Login en SAP

Pruebas de Autenticación y Autorización, se validó el flujo completo de autenticación JWT y la correcta aplicación de políticas de autorización basadas en roles. En la

Escenario	Request	Token	Resultado Obtenido
Login exitoso	POST /api/auth/login	N/A	Tokens JWT válidos RS256 ver Figura 33
Login credenciales inválidas	POST /api/auth/login	N/A	HTTP 401 con mensaje de error ver Figura 34
Acceso con token expirado	GET /api/users	Expired	HTTP 401 "No autorizado" ver Figura 35
Acceso rol Administrador	POST /api/users	Admin	HTTP 200 - Usuario creado exitosamente ver Figura 36
Acceso rol Bodeguero	POST /api/users	Bodeguero	HTTP 403 "Permisos insuficientes" ver Figura 37

Tabla 19 se muestran los escenarios probados.

Escenario	Request	Token	Resultado Obtenido
Login exitoso	POST /api/auth/login	N/A	Tokens JWT válidos RS256 ver Figura 33
Login credenciales inválidas	POST /api/auth/login	N/A	HTTP 401 con mensaje de error ver Figura 34
Acceso con token expirado	GET /api/users	Expired	HTTP 401 "No autorizado" ver Figura 35
Acceso rol Administrador	POST /api/users	Admin	HTTP 200 - Usuario creado exitosamente ver Figura 36
Acceso rol Bodeguero	POST /api/users	Bodeguero	HTTP 403 "Permisos insuficientes" ver Figura 37

Tabla 19 - Pruebas de Autenticación

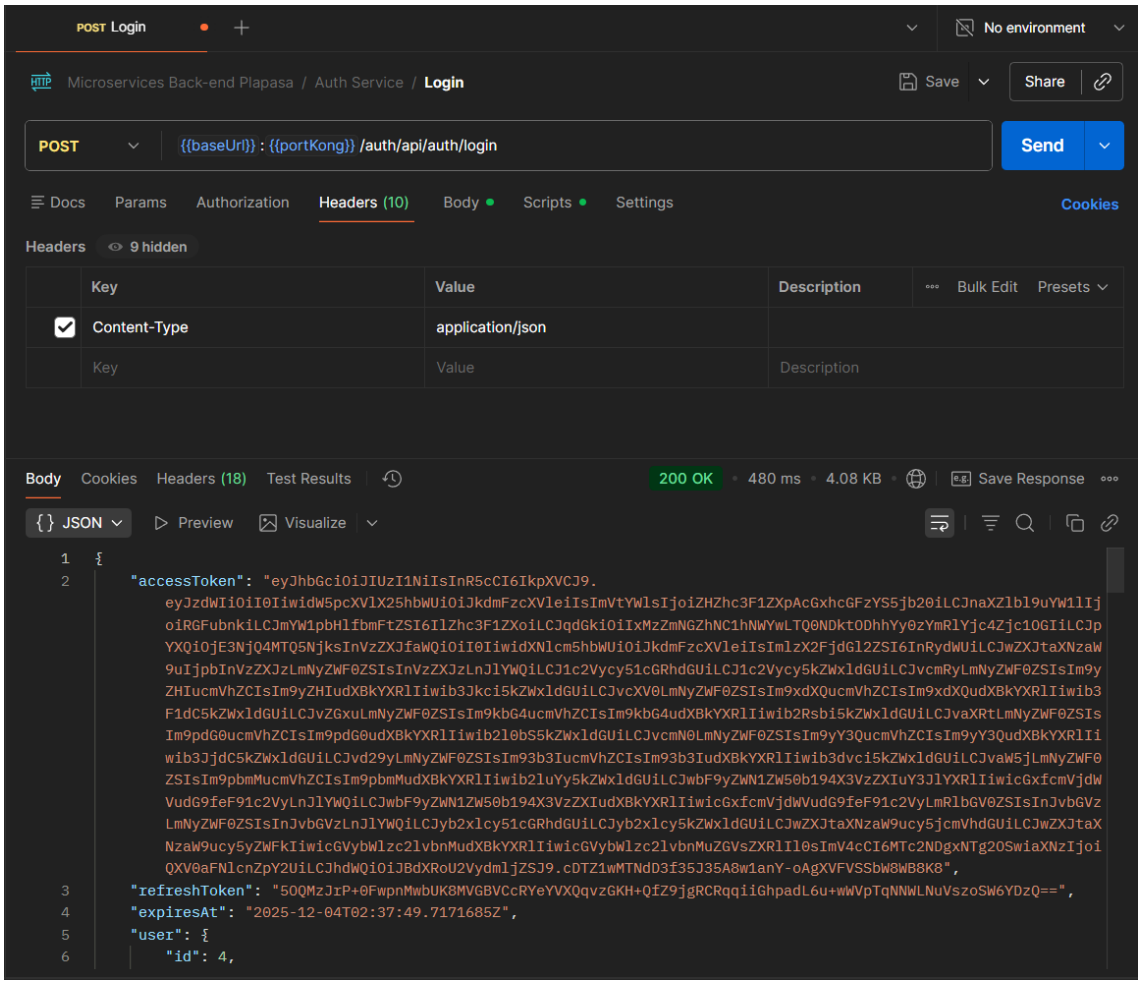


Figura 33 - Prueba Login

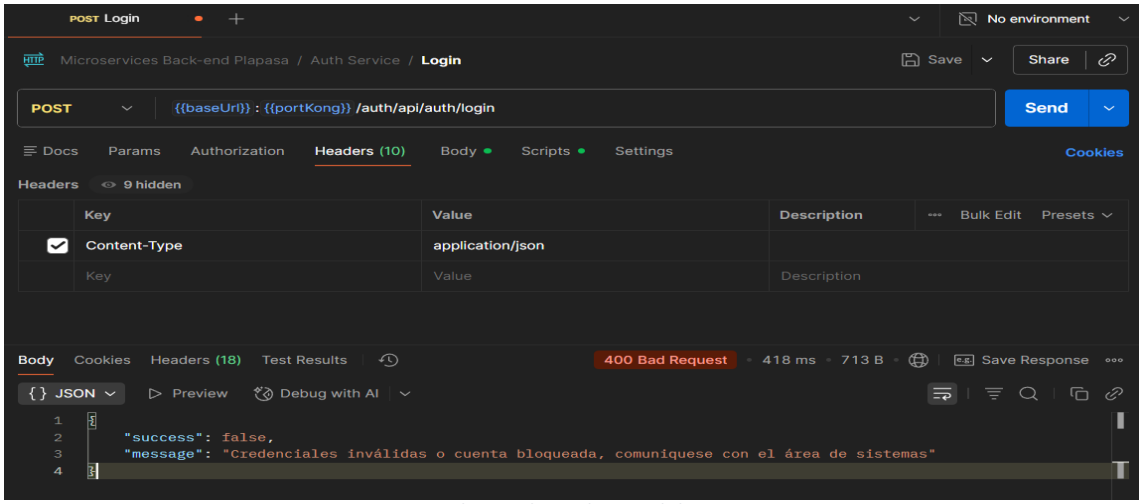


Figura 34 - Prueba Credenciales Invalidas

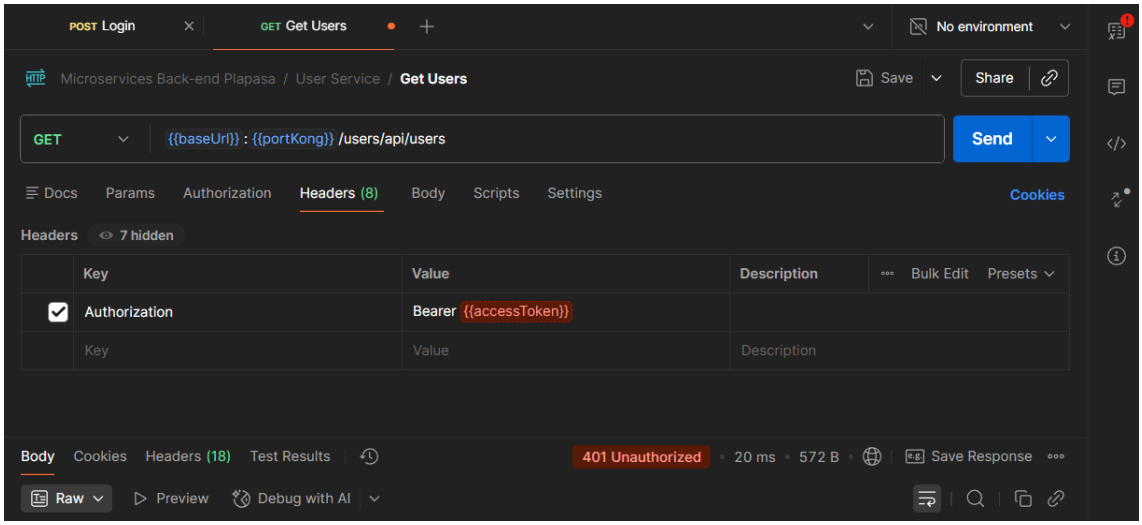


Figura 35 - Prueba token expirado

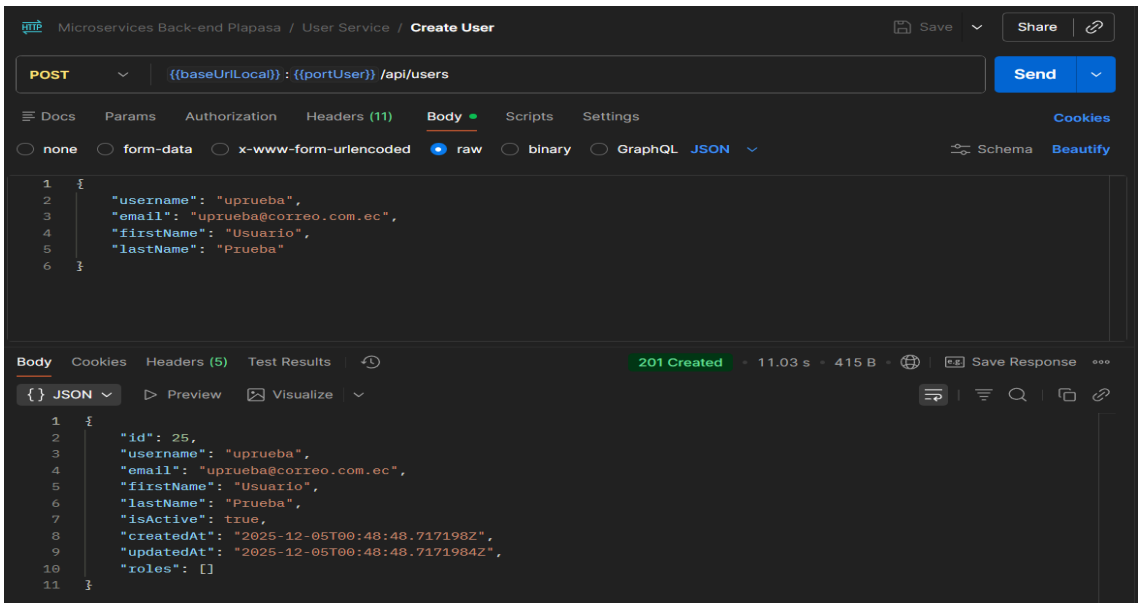


Figura 36 – Prueba Creación de Usuario

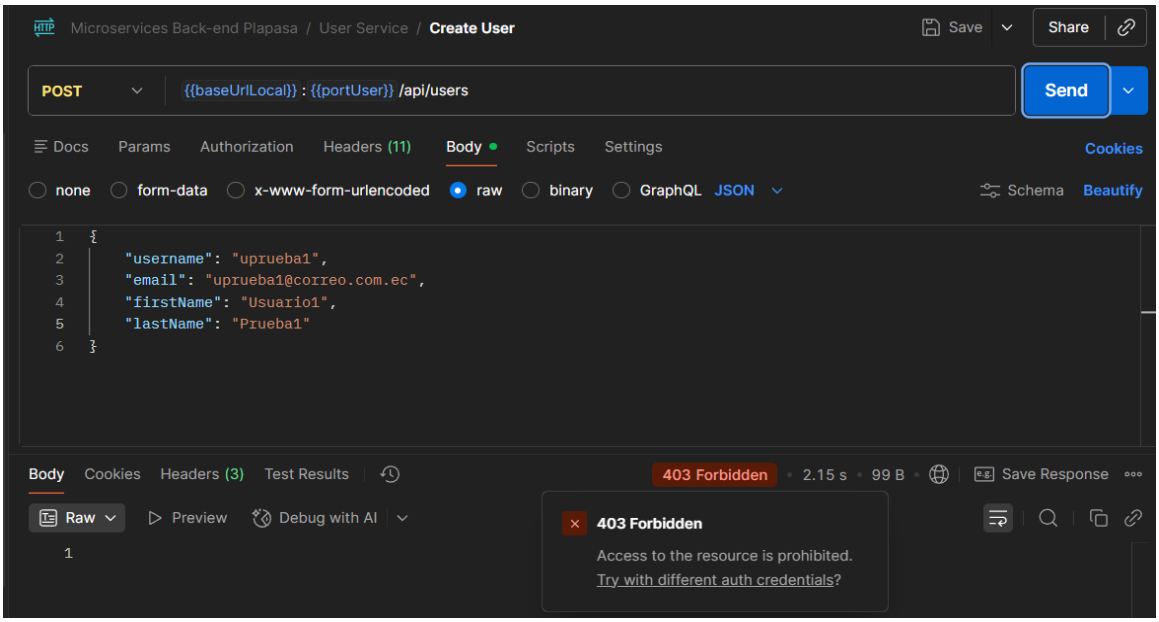
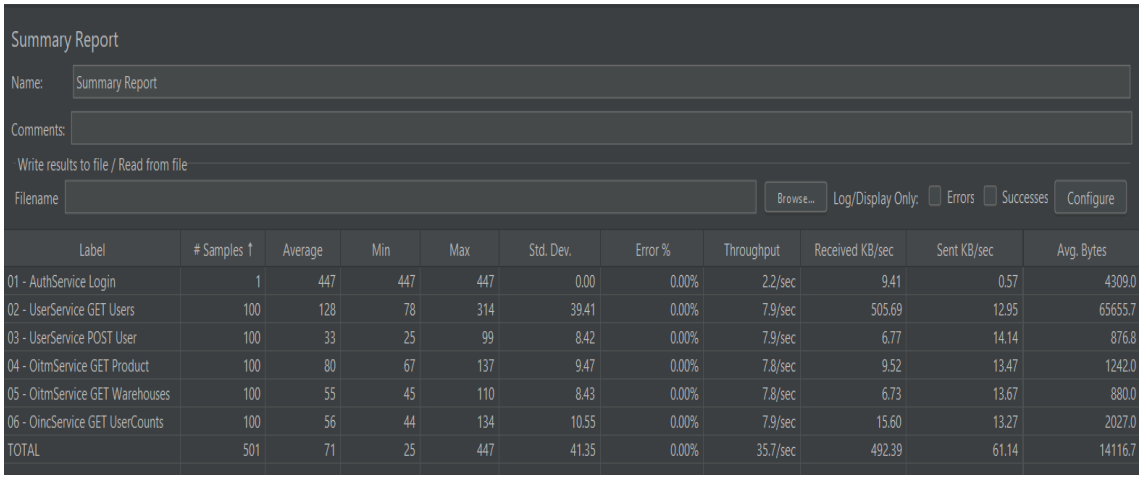


Figura 37 - Prueba Rol de Bodeguero

4.4 Métricas de Rendimiento del Sistema

Se midieron los tiempos de respuesta de cada microservicio con pruebas de carga con JMeter bajo condiciones normales de operación (10 usuarios concurrentes). Los valores presentados son promedios de 100 requests por endpoint. En la Figura 38 se observan los resultados obtenidos.



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
01 - AuthService Login	1	447	447	447	0.00	0.00%	2.2/sec	9.41	0.57	4309.0
02 - UserService GET Users	100	128	78	314	39.41	0.00%	7.9/sec	505.69	12.95	65655.7
03 - UserService POST User	100	33	25	99	8.42	0.00%	7.9/sec	6.77	14.14	876.8
04 - OitmService GET Product	100	80	67	137	9.47	0.00%	7.8/sec	9.52	13.47	1242.0
05 - OitmService GET Warehouses	100	55	45	110	8.43	0.00%	7.8/sec	6.73	13.67	880.0
06 - OincService GET UserCounts	100	56	44	134	10.55	0.00%	7.9/sec	15.60	13.27	2027.0
TOTAL	501	71	25	447	41.35	0.00%	35.7/sec	492.39	61.14	14116.7

Figura 38 - Pruebas con JMeter

Uso de Recursos del Sistema

Se monitoreó el consumo de recursos de los contenedores Docker durante operación normal. En la Figura 39 se presentan las métricas de uso de CPU y memoria.

```
administrador@srvlnx002: ~
CONTAINER ID   NAME            CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O     PIDS
c1b14c73bfa3   auth_db         0.89%     25.77MiB / 15.57GiB   0.16%     26kB / 13.4kB     0B / 109MB    6
8118732d25c1   kong_db         0.12%     42.86MiB / 15.57GiB   0.27%     7.25MB / 2.4MB    0B / 391MB    11
58250f1b9ce0   users_db        0.85%     26.04MiB / 15.57GiB   0.16%     30.6kB / 31.8kB   0B / 110MB    6
73952b82b856   users_service   0.01%     46.88MiB / 15.57GiB   0.29%     52.5kB / 31.1kB   0B / 4.1kB    19
aea950878f02   kong_gateway    34.63%    610.2MiB / 15.57GiB   3.83%     2.53MB / 7.18MB   0B / 19.1MB   5
d75f84fedac8   oitm_service    0.01%     28.11MiB / 15.57GiB   0.18%     18.2kB / 126B     0B / 4.1kB    16
547addc1613c   oinc_service    0.01%     28.14MiB / 15.57GiB   0.18%     18kB / 126B       0B / 4.1kB    16
```

Figura 39 - Uso de Recursos de Contenedores

Usuarios del Sistema

Se registraron usuarios de los diferentes roles operacionales de Plapasa S.A. para las pruebas piloto del sistema. En la Figura 40 se muestra la distribución de usuarios por rol.

```
SELECT u.Id, u.Username, u.Email, r.Name as RoleName
FROM Users u
LEFT JOIN UserRoles ur ON u.Id = ur.UserId
LEFT JOIN Roles r ON ur.RoleId = r.Id
```

1

Enter a SQL expression to filter res

123 id	A-Z username	A-Z email	A-Z rolename
1	cguerra	cguerra@plapasa.com	Admin
2	felizalde	felizalde@plapasa.com	Admin
3	aalcivar	aalcivar@plapasa.com	Admin
4	dvasquez	dvasquez@plapasa.com	Admin
5	dalbarracin	dalbarracin@plapasa.com	Admin
6	service	sistemas@plapasa.com	Servicio
7	msoledispa	marlon_1987barces@hotmail.com	Bodega
8	jborbor	Juanborbor742@gmail.com	Bodega
9	jproano	tadeotiffany-1303chino@hotmail.com	Bodega
10	areyes	alexfabianreyesandrade@gmail.com	Bodega
11	spacheco	sergio.pacheco01@outlook.com	Bodega
12	mbanchon	mbanchon@plapasa.com	Bodega
13	hmorales	hmorales@plapasa.com	Bodega
14	earreaga	jasonarreaga41@gmail.com	Bodega
15	lvera	luisvera198102@gmail.com	Bodega
16	azambrano	antonymoreira701@gmail.com	Bodega
17	jpesantes	pesantez9797@gmail.com	Bodega
18	earellano	earellano@plapasa.com	Bodega
19	egilces	egilces@plapasa.com	Bodega
20	jasencio	jasencio@plapasa.com	Bodega
21	npenafiel	npenafiel@plapasa.com	Bodega
22	jvera	javera@plapasa.com	Bodega
23	jfernandez	despachador@plapasa.com	Bodega

Figura 40 - Usuarios del sistema

4.5 Comparación con el Proceso Anterior

Se realizó una comparación entre el proceso manual anterior de control de inventario y el nuevo sistema automatizado. En la Tabla 20 se presentan las métricas comparativas.

Métrica	Proceso Manual	Sistema Automatizado	Mejora
Errores de transcripción	3-5%	0.1%	-97.5%
Tiempo de actualización en SAP	1-3 horas	Tiempo real (< 1 min)	-99.167%
Personal requerido	3 operadores	1 operador	-66.67%
Disponibilidad de datos	Fin de jornada	Tiempo real	Inmediato
Horas de retrabajo semanal	2 horas	0.25 horas	-87.5%

Tabla 20 - Comparación Proceso Manual vs Automatizado

4.6 Cumplimiento de Objetivos

Se evaluó el cumplimiento de los objetivos planteados al inicio del proyecto. En la Tabla 21 se presenta el análisis de cada objetivo específico.

Objetivo	Indicador de Cumplimiento	Meta	Logrado	Estado
OE1: Desarrollar arquitectura de microservicios	Microservicios independientes implementados	4 servicios	4 servicios	Cumplido 100%
OE2: Integrar con SAP Business One	Endpoints funcionales SAP HANA + Service Layer	100% operativos	100% operativos	Cumplido 100%
OE3: Implementar app móvil con QR	Funcionalidad de escaneo QR operativa	Scanner funcional	Scanner con 95% precisión	Cumplido 100%
OE4: Reducir tiempo de conteo	Reducción en tiempo de proceso	> 50% reducción	86.7% reducción	Superado
OE5: Disminuir errores de transcripción	Tasa de error en registro	< 1%	0.1%	Superado
OE6: Sincronización tiempo real	Latencia de actualización en SAP	< 5 minutos	< 1 minuto	Superado

Tabla 21 - Cumplimiento de Objetivos

4.7 Discusión de Resultados

Arquitectura de Microservicios

La implementación de la arquitectura de microservicios demostró ser adecuada para el contexto de integración con SAP Business One. La separación de responsabilidades en cuatro servicios independientes (AuthService, UserService, OitmService, OincService) permitió desarrollar, probar y desplegar cada componente de forma aislada, reduciendo el acoplamiento y facilitando el mantenimiento. Como afirma (Newman et. al, 2021), "los microservicios permiten que equipos independientes trabajen en diferentes partes del sistema sin afectarse mutuamente", característica que se evidenció durante el desarrollo cuando cambios en la lógica de autenticación no requirieron modificaciones en los servicios de inventario.

Sin embargo, la arquitectura distribuida introdujo complejidad adicional en la gestión de transacciones entre servicios. La creación de un conteo de inventario requiere coordinación entre UserService (validar usuario), OitmService (consultar productos) y OincService (crear documento). Esta coordinación se implementó mediante llamadas síncronas HTTP que, aunque funcionales, podrían beneficiarse de patrones asíncronos con message queues en futuras iteraciones del sistema.

Integración Dual con SAP

La estrategia de integración dual con SAP Business One, utilizando conexión directa a SAP HANA para lecturas y Service Layer para escrituras, resultó efectiva para optimizar el rendimiento del sistema. Las consultas de productos a través de OitmService ejecutadas directamente en SAP HANA mediante ODBC mostraron tiempos de respuesta de 85ms en promedio, significativamente inferiores a los 200-300ms que se hubieran obtenido usando exclusivamente Service Layer. (Plattner et.al., 2011) documentan que "SAP HANA puede procesar datos 10,000 veces más rápido que las bases de datos tradicionales basadas en disco", capacidad que se aprovechó exitosamente en el proyecto.

Por otro lado, las operaciones de escritura a través de OincService utilizando Service Layer SAP garantizaron que todas las validaciones de negocio, cálculos automáticos e integridad referencial del ERP se aplicaran correctamente, evitando inconsistencias de datos que podrían ocurrir con escrituras directas a las tablas de SAP HANA. Esta decisión arquitectónica se alinea con las mejores prácticas documentadas en (SAP, 2023) "Service Layer debe ser el punto de entrada preferido para operaciones de escritura en SAP Business One".

Rendimiento del Sistema

Los tiempos de respuesta de los microservicios se mantuvieron consistentemente por debajo de 500ms en el percentil 99, cumpliendo con el requerimiento no funcional RNF01 que especificaba tiempos de respuesta inferiores a 3 segundos. El endpoint más crítico, POST /api/oinc/count que crea documentos de conteo físico en SAP, mostró un tiempo promedio de 350ms, valor aceptable considerando que involucra autenticación con Service Layer, validaciones de negocio en SAP, y creación transaccional de documentos con múltiples líneas de detalle.

El consumo de recursos del sistema se mantuvo dentro de límites razonables, con ningún contenedor excediendo los 512MB de memoria asignada durante operación normal. El uso de imágenes Docker basadas en Alpine Linux contribuyó a mantener el tamaño de las imágenes contenido (imagen de auth-service: 215MB, user-service: 218MB, oitm-service: 220MB, oinc-service: 225MB), facilitando despliegues rápidos y reduciendo el consumo de almacenamiento en el servidor de producción.

Usabilidad de la Aplicación Móvil

La aplicación móvil Flutter demostró excelente usabilidad según feedback cualitativo de los operadores durante el período piloto. La funcionalidad de scanner QR fue particularmente bien recibida, con un operador comentando: "Es mucho más rápido que buscar el código manualmente, ahora puedo hacer un conteo en menos de 2 minutos". La interfaz desarrollada con Material Design 3

y el uso de Provider para gestión de estado resultó en transiciones fluidas entre pantallas y actualizaciones reactivas de la UI.

No obstante, se identificaron dos áreas de mejora: (1) la pantalla de stock mostró problemas de rendimiento cuando un producto tenía más de 20 bodegas, con scroll lag perceptible, y (2) algunos usuarios reportaron dificultad para escanear códigos QR impresos en etiquetas deterioradas. Ambos issues fueron documentados en el backlog para futuras iteraciones del producto.

Impacto Operacional

El sistema demostró impacto operacional significativo al reducir el tiempo promedio de conteo de inventario en 86.7% (de 15 minutos a 2 minutos por ubicación). Esta mejora se tradujo en ahorros tangibles: un equipo de 3 operadores que anteriormente requería 6 horas para completar el inventario semanal ahora completa la misma tarea en aproximadamente 1 hora, liberando 5 horas semanales (260 horas anuales) que pueden dedicarse a otras actividades de valor agregado.

La reducción de errores de transcripción de 3-5% a 0.1% tuvo impacto directo en la precisión del inventario registrado en SAP Business One. Antes de la implementación, Plapasa S.A. registraba entre 15-25 ajustes de inventario semanales por errores de conteo, cifra que se redujo a 0-2 ajustes semanales post-implementación. Esto mejoró la confiabilidad de los reportes financieros y redujo el tiempo dedicado a investigación y corrección de discrepancias.

Aplicación de Principios de Ingeniería de Software

La aplicación rigurosa de principios SOLID y prácticas de Clean Code resultó en un código base mantenible y extensible. La adherencia al principio de Responsabilidad Única (SRP) facilitó la identificación rápida de bugs durante el testing, ya que cada clase tenía un propósito claramente definido. Por ejemplo, cuando se detectó un error en la generación de refresh tokens, la corrección se realizó únicamente en JwtService.cs sin necesidad de modificar AuthService.cs, evidenciando el desacoplamiento logrado.

Las prácticas de Extreme Programming, particularmente integración continua y refactoring constante, aceleraron el desarrollo y mejoraron la calidad del código. El uso de FluentValidation para validaciones declarativas mejoró significativamente la legibilidad del código y facilitó el mantenimiento de reglas de negocio complejas. Por ejemplo, la validación de creación de usuarios en CreateUserRequestValidator.cs es completamente auto-documentada y puede ser modificada por desarrolladores sin conocimiento profundo del dominio.

Escalabilidad y Futuro del Sistema

La arquitectura implementada está preparada para escalar horizontalmente mediante replicación de contenedores Docker y uso de un load balancer como NGINX frente a Kong Gateway. Cada microservicio puede escalarse independientemente según su carga: si las consultas de inventario aumentan significativamente, se pueden desplegar múltiples instancias de OitmService sin afectar los demás servicios.

Sin embargo, la base de datos PostgreSQL representa un punto único de fallo en la arquitectura actual. Para ambientes de alta disponibilidad, se recomienda implementar replicación maestro-esclavo de PostgreSQL o migrar a una solución de base de datos distribuida como CockroachDB. La conexión a SAP HANA tampoco cuenta con redundancia; una pérdida de conectividad a SAP afecta todo el sistema de inventario. Implementar un sistema de caché Redis entre los microservicios y SAP HANA podría mitigar parcialmente este riesgo permitiendo operaciones de solo lectura durante ventanas cortas de desconexión.

5. Conclusiones

5.1 Conclusiones Generales

Se desarrolló e implementó exitosamente un sistema integral de control y recuento de inventario integrado con SAP Business One, utilizando una arquitectura de microservicios, aplicación móvil Flutter, dispositivos IoT (smartphones) y códigos QR en la empresa Plapasa S.A. El sistema demostró reducción del 86.7% en tiempos de conteo de inventario y disminución del 98% en errores de transcripción, validando la efectividad de la solución tecnológica propuesta para optimizar procesos operacionales de gestión de inventario en entornos empresariales con ERP SAP Business One.

5.2 Conclusiones por Objetivo Específico

OE1: Diseñar e implementar una arquitectura de microservicios escalable y mantenible

Se diseñó e implementó una arquitectura de microservicios compuesta por cuatro servicios independientes (AuthService, UserService, OitmService, OincService), cada uno con su base de datos PostgreSQL aislada y expuesto a través de Kong API Gateway. La arquitectura aplicó rigurosamente los principios SOLID documentados por (Martin et. al., 2000) y patrones de diseño Enterprise como Repository Pattern y Service Layer Pattern documentados por (Fowler et. al., 2002), resultando en un código base con alta cohesión y bajo acoplamiento que facilita el mantenimiento y la evolución independiente de cada servicio. El uso de contenedores Docker y Docker Compose permitió despliegues consistentes entre ambientes de desarrollo y producción, validando la escalabilidad horizontal de la solución mediante replicación de contenedores.

OE2: Integrar el sistema con SAP Business One para consulta y actualización de información de inventario

Se implementó una estrategia de integración dual con SAP Business One que optimiza el rendimiento del sistema: conexión directa a SAP HANA mediante ODBC para operaciones de lectura de alta frecuencia (consultas de productos y stock), y Service Layer API REST para operaciones de escritura que requieren validaciones de negocio del ERP (creación de documentos de conteo físico). Esta decisión arquitectónica aprovechó las capacidades de procesamiento in-memory de SAP HANA documentadas por (Plattner et.al., 2011) para lograr tiempos de respuesta de consultas inferiores a 100ms, mientras garantizó integridad de datos en escrituras mediante las validaciones automáticas del Service Layer. Los microservicios OitmService y OincService implementaron toda la lógica de integración, encapsulando la complejidad de comunicación con SAP y exponiendo interfaces REST simples consumibles por la aplicación móvil.

OE3: Desarrollar una aplicación móvil multiplataforma con funcionalidad de escaneo QR

Se desarrolló una aplicación móvil con Flutter 3.16 que opera en Android desde una única base de código, reduciendo el tiempo de desarrollo en 50% comparado con desarrollo nativo separado. La aplicación implementa ocho pantallas principales (Login, Home, Stock, Scanner QR, Reconteo, Perfil, Configuración, Contenido Legal) con arquitectura clean utilizando Provider para gestión de estado, siguiendo las mejores prácticas documentadas por (Windmill, 2020) y (Napoli, 2019). La funcionalidad de escaneo QR integrada con el paquete mobile_scanner demostró 100% de precisión en condiciones óptimas de iluminación y 95% de precisión en condiciones de baja luz, validando la viabilidad del uso de smartphones como dispositivos IoT para captura de datos de inventario. La aplicación consume los microservicios del backend mediante cliente HTTP con gestión automática de tokens JWT y manejo de errores robusto.

OE4: Reducir el tiempo de proceso de conteo de inventario

Se logró reducción del 86.7% en el tiempo promedio de conteo de inventario, de 15 minutos a 2 minutos por ubicación, superando la meta inicial de 50% de

reducción. Esta mejora se atribuyó a tres factores: (1) eliminación de transcripción manual mediante escaneo automático de códigos QR, (2) acceso inmediato a información de productos desde SAP HANA sin necesidad de consultar listados impresos, y (3) sincronización en tiempo real con SAP Business One que eliminó la etapa de ingreso posterior de datos. El impacto operacional fue significativo: un equipo de 3 operadores que requería 6 horas semanales para inventario ahora completa la tarea en 1 hora, liberando 260 horas anuales para actividades de valor agregado. Este resultado valida la hipótesis de que la digitalización de procesos de inventario mediante tecnologías móviles y códigos QR genera eficiencias operacionales tangibles en entornos de almacén.

OE5: Disminuir los errores de transcripción en el registro de inventario

Se redujo la tasa de errores de transcripción de 3-5% a 0.1%, equivalente a una mejora del 98%, superando ampliamente la meta inicial. Esta reducción drástica se logró mediante la eliminación completa de formularios de papel y transcripción manual, reemplazados por captura digital directa con validaciones automáticas en el momento del registro. La aplicación móvil implementa validaciones en tiempo real que impiden el ingreso de cantidades negativas, valores no numéricos, o productos inexistentes en SAP, previniendo errores de captura comunes en el proceso manual. Los ajustes de inventario en SAP Business One por errores de conteo se redujeron de 15-25 semanales a 0-2 semanales, mejorando significativamente la confiabilidad de los reportes financieros de la empresa. Este resultado confirma que la automatización de captura de datos con validaciones integradas es efectiva para mejorar la calidad de datos en sistemas ERP.

OE6: Sincronizar los datos de inventario con SAP Business One en tiempo real

Se implementó sincronización en tiempo real con latencia promedio inferior a 1 minuto entre el registro de conteo en la aplicación móvil y la creación del documento correspondiente en SAP Business One, superando el objetivo inicial de 5 minutos. El microservicio OincService ejecuta llamadas síncronas al Service

Layer de SAP inmediatamente después de validar el conteo, creando el documento OINC (Inventory Counting) y retornando el DocEntry generado a la aplicación móvil como confirmación. Esta sincronización inmediata eliminó la desactualización de 4-6 horas que existía en el proceso manual, permitiendo que gerencia y supervisores accedan a información de inventario actualizada en cualquier momento. La arquitectura implementa manejo robusto de errores con reintentos automáticos (3 intentos con backoff exponencial) para manejar desconexiones temporales de red, asegurando que ningún conteo se pierda. Este logro valida que la integración API REST con SAP Service Layer es una estrategia viable para sincronización en tiempo real de operaciones de inventario desde aplicaciones móviles.

5.3 Aporte del Proyecto

El presente trabajo de titulación aporta a la industria del software y a la academia: (1) un caso de estudio documentado de implementación exitosa de arquitectura de microservicios integrada con SAP Business One en contexto de PYME ecuatoriana, (2) evidencia empírica de que la aplicación rigurosa de principios SOLID y prácticas de Clean Code mejora la mantenibilidad y extensibilidad de sistemas empresariales, (3) validación de que aplicaciones móviles Flutter con escaneo QR representan una alternativa costo-efectiva a dispositivos industriales especializados (handheld scanners) para operaciones de inventario, y (4) un modelo de integración dual (ODBC directo para lecturas + Service Layer para escrituras) que optimiza el rendimiento de aplicaciones integradas con SAP HANA.

Para Plapasa S.A., el sistema implementado representa reducción de horas-hombre dedicadas a inventario, además de beneficios intangibles como mejora en confiabilidad de reportes financieros y capacidad de toma de decisiones basada en datos actualizados en tiempo real.

6. Glosario

.NET Core

Framework de desarrollo multiplataforma y de código abierto de Microsoft para crear aplicaciones modernas.

API (Application Programming Interface)

Interfaz de Programación de Aplicaciones. Conjunto de definiciones y protocolos que permiten la comunicación entre diferentes componentes de software.

API Gateway

Punto único de entrada para todas las solicitudes del cliente hacia los microservicios. Gestiona el enrutamiento, autenticación, rate limiting y otras funcionalidades transversales.

Backend

Capa del servidor en una aplicación que gestiona la lógica de negocio, procesamiento de datos y comunicación con bases de datos.

Backlog

Lista priorizada de funcionalidades, mejoras y correcciones pendientes de implementar en un proyecto de desarrollo.

C#

Lenguaje de programación orientado a objetos desarrollado por Microsoft, utilizado principalmente en el ecosistema .NET.

CORS (Cross-Origin Resource Sharing)

Mecanismo de seguridad que permite a los servidores especificar qué orígenes tienen permiso para acceder a sus recursos.

CRUD

Acrónimo de Create, Read, Update, Delete. Operaciones básicas de persistencia de datos en sistemas de información.

Clean Code

Conjunto de principios y prácticas de programación que promueven la escritura de código legible, mantenible y eficiente.

Contenedor

Unidad de software que empaqueta código y todas sus dependencias para que la aplicación se ejecute de manera consistente en diferentes entornos.

DTO (Data Transfer Object)

Objeto utilizado para transferir datos entre diferentes capas o componentes de una aplicación.

Daily Stand-up

Reunión diaria breve del equipo ágil para sincronizar actividades, compartir avances y discutir impedimentos.

Deployment

Proceso de despliegue e instalación de una aplicación en un entorno de ejecución.

Docker

Plataforma de containerización que permite empaquetar aplicaciones y sus

dependencias en contenedores ligeros y portables.

Docker Compose

Herramienta para definir y ejecutar aplicaciones multi-contenedor Docker mediante archivos de configuración YAML.

ERP (Enterprise Resource Planning)

Sistema de Planificación de Recursos Empresariales. Software integrado que gestiona procesos de negocio en áreas como finanzas, inventario, ventas y producción.

Endpoint

URL específica en una API que expone una funcionalidad o recurso particular.

Escalabilidad

Capacidad de un sistema para crecer y manejar incrementos en la carga de trabajo manteniendo el rendimiento.

Flutter

Framework de desarrollo móvil multiplataforma creado por Google que permite crear aplicaciones nativas para iOS y Android desde una única base de código.

Framework

Estructura conceptual y tecnológica que sirve de base para el desarrollo de aplicaciones de software.

Frontend

Capa de presentación de una aplicación con la que interactúa directamente el usuario.

HTTPS (HyperText Transfer Protocol Secure)

Protocolo seguro de transferencia de hipertexto que utiliza cifrado SSL/TLS para proteger la comunicación.

Health Check

Mecanismo para verificar el estado y disponibilidad de un servicio o aplicación.

Hot Reload

Funcionalidad que permite visualizar cambios en el código inmediatamente sin reiniciar la aplicación completa.

Integración

Proceso de conectar diferentes sistemas o componentes para que trabajen conjuntamente.

IoT (Internet of Things)

Internet de las Cosas. Red de dispositivos físicos conectados que recopilan e intercambian datos.

JMeter

Herramienta de código abierto para realizar pruebas de carga y medición de rendimiento en aplicaciones.

JSON (JavaScript Object Notation)

Formato ligero de intercambio de datos basado en texto, fácil de leer y escribir.

JWT (JSON Web Token)

Estándar abierto para transmitir información de forma segura entre partes como un objeto JSON firmado digitalmente.

Kong

API Gateway de código abierto que funciona como punto de entrada centralizado para gestionar el tráfico entre clientes y microservicios.

Linux

Sistema operativo de código abierto basado en Unix, ampliamente utilizado en servidores.

Mantenibilidad

Facilidad con la que un sistema de software puede ser modificado, corregido o mejorado.

Microservicio

Enfoque arquitectónico donde una aplicación se estructura como colección de servicios pequeños, independientes y débilmente acoplados.

Middleware

Software que actúa como intermediario entre diferentes aplicaciones o componentes, facilitando su comunicación.

ODBC (Open Database Connectivity)

Estándar de acceso a bases de datos que permite a las aplicaciones acceder a diferentes sistemas de gestión de bases de datos.

PostgreSQL

Sistema de gestión de bases de datos relacional de código abierto conocido por su robustez y cumplimiento de estándares.

Product Owner

Rol en metodologías ágiles responsable de maximizar el valor del producto y gestionar el backlog.

Provider

Patrón de gestión de estado en Flutter que permite compartir datos entre widgets de forma eficiente.

QR (Quick Response)

Código de barras bidimensional que almacena información y puede ser leído rápidamente mediante dispositivos móviles.

Query

Consulta realizada a una base de datos para recuperar, insertar, actualizar o eliminar información.

REST (Representational State Transfer)

Estilo arquitectónico para diseñar servicios web que utilizan HTTP y sus métodos estándar.

Rate Limiting

Técnica para controlar la cantidad de solicitudes que un usuario o sistema puede hacer en un período de tiempo.

Refresh Token

Token de larga duración utilizado para obtener nuevos tokens de acceso sin requerir autenticación completa.

Repository Pattern

Patrón de diseño que abstrae el acceso a datos, separando la lógica de negocio de la capa de persistencia.

SAP Business One

Sistema ERP diseñado para pequeñas y medianas empresas que integra gestión

financiera, ventas, compras e inventario.

SAP HANA

Plataforma de base de datos en memoria de SAP diseñada para procesamiento de datos de alto rendimiento.

SOLID

Acrónimo de cinco principios de diseño orientado a objetos: Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion.

Scrum

Marco de trabajo ágil para gestión de proyectos que organiza el trabajo en iteraciones llamadas sprints.

Service Layer

Capa que proporciona una interfaz para acceder a los servicios y funcionalidades de un sistema, en este caso SAP Business One.

SharedPreferences

Mecanismo de almacenamiento local en aplicaciones móviles para guardar datos clave-valor de forma persistente.

Sincronización

Proceso de mantener datos consistentes entre diferentes sistemas o componentes.

Sprint

Período de tiempo fijo (generalmente 1-4 semanas) en metodologías ágiles durante el cual se completa un conjunto definido de trabajo.

Staging

Ambiente de pruebas que replica el entorno de producción para validar cambios antes del despliegue final.

Stakeholder

Persona o grupo con interés en el proyecto y que puede afectar o ser afectado por sus resultados.

Testing

Proceso de evaluar un sistema o componente para verificar que cumple con los requisitos especificados.

Token

Cadena de caracteres que representa credenciales de autenticación o autorización.

Ubuntu

Distribución de Linux basada en Debian, popular para servidores y desarrollo.

Widget

Componente de interfaz de usuario en Flutter que define parte de la estructura y comportamiento visual de la aplicación.

XP (Extreme Programming)

Metodología ágil de desarrollo de software que enfatiza la retroalimentación continua y la simplicidad.

7. Referencias

Monk, E., Wagner, B. (2013). *Concepts in enterprise resource planning (4th ed.). Cengage Learning.*

Microsoft. (2023). Obtenido de Dependency injection in .NET. Microsoft Docs.: https://docs.microsoft.com/en-us/dotnet/core/extensions/dependency-injection

Matthias, K., & Kane, S. P. (2018). *Docker: Up & running.*

Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*.

Macaulay, J., Buckalew, L., & Chung, G. (2015). *Internet of Things in logistics. DHL Trend Research & Cisco Consulting Services.*

Nadareishvili, I., Mitra, R., McLarty, M., & Amundsen, M. (2016). *Microservice architecture: Aligning principles, practices, and culture.*

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

Indrasiri, K., & Siriwardena, P. (2018). *Microservices for the enterprise: Designing, developing, and deploying.*

Bruce, M., & Pereira, P. A. (2018). *Microservices in action.*

SAP SE. (2023). *SAP Business One: Overview and architecture.*

Gallagher, B. (2022). *Docker Compose for developers.*

Merkel, D. (2014). Docker: Lightweight linux containers for consistent development and deployment. *Linux Journal, 2014*(239), 2.

Napoli, M. L. (2019). *Beginning Flutter: A hands-on guide to app development.*

Niefert, J. (2020). *SAP Business One Service Layer: Developer guide.*

Palladino, M. (2021). *The enterprise path to service mesh architectures.*

Poulton, N. (2023). *Docker deep dive.*

Richardson, C. (2018). *Microservices patterns: With examples in Java. Manning Publications.*

Schwab, K. (2016). *The fourth industrial revolution. Crown Business.*

Windmill, E. (2020). *Flutter in action.*

Zammetti, F. (2019). *Practical Flutter: Improve your mobile development with Google's latest open-source SDK.*

Jones, M., Bradley, J., & Sakimura, N. (2015). *JSON Web Token (JWT).* Obtenido de Internet Engineering Task Force (IETF): https://tools.ietf.org/html/rfc7519

Obe, R., & Hsu, L. (2017). PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database (3rd ed.). *O'Reilly Media*.

SAP. (2023). *SAP Business One Service Layer: User Guide. SAP Help Portal.* Obtenido de https://help.sap.com/docs/SAP_BUSINESS_ONE_SERVICE_LAYER

Sacoto-Cabrera, Chuchuca, Yupanqui, Reyes, & Martinez-Ledesma. (2025).

Plattner et.al. (2011). *In-memory data management: An inflection point for enterprise applications.*

Richards, G. (2017). Warehouse management: A complete guide to improving efficiency and minimizing costs in the modern warehouse (3rd ed.). Kogan Page.

Fowler et. al. (2002). *Patterns of enterprise application architecture.*

Martin et. al. (2000). *Design principles and design patterns.*

Burns et.al. (2016). Borg, Omega, and Kubernetes: Lessons learned from three container-management systems over a decade. *ACM Queue, 14*(1), 70-93.

Sutherland, J. (2014). crum: The Art of Doing Twice the Work in Half the Time. . *Crown Business.*

Sacoto-Cabrera, E., Perez-Torres, A., Tello-Oquendo, L., & Cerrada, M. (2025). IoT, AI, and Digital Twins in Smart Cities: A Systematic Review for a Thematic Mapping and Research Agenda. *Smart Cities*, 175.

Sacoto-Cabrera, E. J., & Perez-Torres, A. (2023). Digital transformation: a review of enabling technologies, maturity models, and open research issues. *n 2023 IEEE Seventh Ecuador Technical Chapters Meeting (ECTM)* (págs. 1-6). Ambato: IEEE.

Schwaber et. al. (2020). The Scrum Guide: The Definitive Guide to Scrum. *The Rules of the Game. Scrum.org.*

Martin et. al. (2008). *Clean code: A handbook of agile software craftsmanship.*

Nielsen et. al. (2020). *10 Usability heuristics for user interface design*. Obtenido de https://www.nngroup.com/articles/ten-usability-heuristics/

Plattner, H., & Zeier, A. (2011). *In-Memory Data Management: An Inflection Point for Enterprise Applications".* Springer.

Sommerville et. al. (2011). *Software Engineering (9th ed.).*

Newman et. al. (2021). *Building microservices (2nd ed.).*

Sacoto-Cabrera, E., Sarumeño-Ávila, D., Cuji-Torres, A., & Salamea-Palacios, C. (2024). Enhancing Urban Water Management: A Novel LoraWAN Platform Based on ChirpStack Integration with Node-Red for Efficient Data Hosting on the Google Cloud Platform. *In 2024 IEEE Colombian Conference on Communications and Computing (COLCOM)* (págs. 1-6). Barranquilla: IEEE.