**Fit-Import**

**Soluciones fitness a través de la importación**

**Integrante:**

Angel Castillo

Diony Peguero

**Índice**

1. **Introducción**
2. **Desarrollo** 2.1 Definición del Problema  
    2.2 Descripción General del Proyecto  
    2.3 Análisis del Problema
3. **Diseño de la Solución** 3.1 Microservicios Clave  
    3.2 Stack Tecnológico
4. **Infraestructura** 4.1 Amazon Web Services (AWS)  
    4.2 API Gateway: Kong  
    4.3 PostgreSQL  
    4.4 Redis
5. **Frameworks Utilizados** 5.1 Spring Boot  
    5.2 Node.js  
    5.3 Quarkus
6. **Requisitos del Sistema** 6.1 Requisitos Funcionales (RF)  
    6.2 Requisitos No Funcionales (RNF)
7. **Herramientas Colaborativas** 7.1 Uso de Jira  
    7.2 Tabla de Tareas
8. **Documentación Técnica Swagger**

8.1 Carrito  
 8.2 Inventario  
 8.3 Usuario

1. **Conclusión**

**Introducción**

En un mercado cada vez más competitivo y con una alta exigencia por parte de los consumidores institucionales, Fit-Import se posiciona como una innovadora plataforma chilena dedicada a la importación y distribución de equipamiento fitness. Dirigida específicamente a gimnasios y centros de entrenamiento profesional, la empresa ha logrado crecer rápidamente dentro del ecosistema digital, diferenciándose por ofrecer productos de alto estándar, durabilidad comprobada y respaldo de marcas extranjeras de prestigio. Sin embargo, este crecimiento ha puesto en evidencia las limitaciones estructurales de su sistema informático actual, el cual fue concebido para una operación inicial de menor complejidad y volumen.

La problemática central radica en el uso de una arquitectura monolítica rígida, que concentra todas las operaciones —desde la gestión de stock hasta la facturación— en una sola estructura interdependiente. Este modelo genera constantes cuellos de botella durante períodos de alta demanda, ralentización de funciones críticas como la creación de pedidos o la actualización de inventario, e incluso caídas del sistema que comprometen la experiencia de los usuarios. Más allá del impacto técnico, estas deficiencias afectan directamente la confiabilidad del servicio, dificultan la escalabilidad operativa y restringen la capacidad de Fit-Import para responder ágilmente a nuevas oportunidades comerciales.

Este proyecto tiene como propósito diseñar e implementar una transformación arquitectónica basada en microservicios, con el fin de segmentar las responsabilidades del sistema en módulos independientes, autónomos y escalables. A través de esta nueva infraestructura, se busca optimizar el rendimiento del sistema, mejorar la mantenibilidad del código, aumentar la tolerancia a fallos y permitir una mayor flexibilidad a nivel tecnológico, operacional y comercial. La adopción de esta estrategia no solo resolverá los problemas actuales, sino que también habilitará la evolución constante del negocio mediante la incorporación progresiva de nuevas funcionalidades y servicios.

El presente documento expone en detalle el proceso de análisis, diseño y planificación de esta transición tecnológica. Además, se abordan los principales desafíos asociados, incluyendo aspectos de seguridad, colaboración entre equipos, migración de datos y consideraciones éticas como la protección de la privacidad, la sostenibilidad de la infraestructura y la inclusión de prácticas laborales responsables. A través de este enfoque integral, se pretende no solo modernizar la base tecnológica de Fit-Import, sino también consolidar su posición como un actor clave en la profesionalización del fitness en Chile.

**Desarrollo**

### **Definición del Problema**

Fit-Import enfrenta serias limitaciones operativas debido a su actual sistema monolítico centralizado. Esta arquitectura, basada en un código único para ventas, inventario, logística y gestión de clientes, impide escalar de forma modular, generando cuellos de botella en periodos de alta demanda. Además, cada actualización requiere reiniciar todo el sistema, provocando inactividad generalizada y dificultando el mantenimiento. Estas condiciones afectan tanto la eficiencia interna como la experiencia del cliente, con lentitud en la interfaz web, información inconsistente y fallos frecuentes en horarios críticos. Para superar estos desafíos, se propone migrar a una arquitectura de microservicios, que permita escalar por separado los módulos más exigidos, mejorar la disponibilidad y optimizar el rendimiento general de la plataforma.

### **Descripción General del Proyecto**

El presente proyecto implica la creación de una plataforma de comercio electrónico fullstack altamente especializada, concebida especialmente para la importación y distribución de equipo profesional de fitness de alta calidad, destinada exclusivamente a gimnasios dentro del mercado chileno. Nuestra propuesta marca un avance relevante en el ámbito B2B del fitness, al ofrecer una opción tecnológicamente avanzada que aborda las principales dificultades que actualmente enfrentan los gimnasios profesionales en la adquisición de sus equipos. La plataforma se destaca por su atención especializada en productos premium que son importados directamente desde los principales fabricantes a nivel internacional, asegurando autenticidad, calidad profesional y una durabilidad excepcional. A diferencia de los marketplaces generales, nuestro sistema incluye funciones diseñadas específicamente para satisfacer las necesidades operativas de las instalaciones deportivas, como un complejo módulo para cotizaciones masivas, herramientas avanzadas para comparar técnicamente distintos equipos y un innovador sistema de planificación de espacios que facilita la visualización de cómo se integrarían los equipos físicamente en el entorno del gimnasio. Uno de los aspectos clave de nuestra plataforma es su total integración con los procesos logísticos internacionales. Hemos concebido un motor de cálculo automatizado que ofrece estimaciones precisas sobre costos de importación, plazos de entrega y trámites aduaneros, eliminando sorpresas y costos adicionales que normalmente afectan estas transacciones. Este sistema se complementa con un módulo avanzado de seguimiento que permite la supervisión en tiempo real de cada fase del recorrido de los equipos, desde la fábrica hasta las instalaciones del gimnasio. Desde una perspectiva tecnológica, la plataforma utiliza una arquitectura de microservicios altamente escalable, desarrollada con Spring Boot para el procesamiento de transacciones

Desarrollamos una plataforma B2B especializada en la importación y comercialización de equipamiento fitness profesional para gimnasios en Chile. Nuestra solución tecnológica ofrece:

**Catálogo exclusivo** de marcas internacionales premium, con garantía de autenticidad y calidad profesional

**Herramientas especializadas**: cotizaciones masivas, comparación técnica de equipos y planificación virtual de espacios

**Logística integrada:** cálculo automático de costos de importación y tracking en tiempo real

**Arquitectura robusta**: microservicios (Spring Boot, Node.js, Python) + visualización 3D (Next.js, Three.js)

**Infraestructura segura:** AWS con certificación PCI DSS Nivel 1 y arquitectura Zero Trust

#### **Base de Datos**

**Se creara un script SQL que incluye la definición de las tablas necesarias para los tres microservicios, aplicando las siguientes buenas prácticas:**

* **Llaves primarias (PRIMARY KEY) y foráneas (FOREIGN KEY)**
* **Restricciones de unicidad (UNIQUE) y obligatoriedad (NOT NULL)**
* **Tipos de datos según la naturaleza de los campos (VARCHAR, DATE, INT)**

**Beneficios clave**

✓ Reducción de 30-40% en costos al eliminar intermediarios

✓ Plazos de entrega optimizados (15-20 días vs 60-90 tradicionales)

✓ Asesoría técnica especializada y gestión del ciclo de vida del equipamiento

Esta plataforma se posiciona como un partner tecnológico estratégico para el crecimiento de gimnasios profesionales en Chile.De esta manera, se transforma en una solución completa que va más allá de una transacción económica, convirtiéndose en un aliado tecnológico clave para el desarrollo de gimnasios profesionales en Chile.

## **Análisis del Problema**

El sistema actual de Fit-Import responde a una estructura monolítica que concentra todas las operaciones dentro de una misma base de código. Este enfoque, aunque suficiente en los primeros meses de operación, ha generado serias limitaciones a medida que la plataforma ha crecido. La arquitectura centralizada impide escalar módulos por separado, lo que complica la respuesta ante aumentos de carga o nuevas necesidades funcionales.

Además, el sistema presenta una fuerte dependencia entre componentes, por lo que cualquier modificación requiere revisar todo el conjunto. Esto ralentiza los procesos de actualización y hace más difícil mantener el servicio disponible. Problemas como caídas durante picos de tráfico, lentitud en la web, errores en la sincronización de inventario y dificultades logísticas se han vuelto frecuentes. Frente a este panorama, resulta fundamental migrar hacia una arquitectura distribuida que permita mayor control, escalabilidad y resiliencia.

## **Principales causas de fallos y sobrecargas en el sistema**

La siguiente tabla resume las principales debilidades técnicas y operativas del sistema actual de Fit-Import, derivadas de su arquitectura monolítica centralizada:

| **Causa identificada** | **Impacto en la operación** |
| --- | --- |
| Escalabilidad general limitada | No es posible reforzar módulos específicos, como pagos o stock, durante alta demanda. |
| Saturación por consultas simultáneas | El sistema se ralentiza cuando múltiples usuarios acceden al catálogo o realizan pedidos. |
| Actualizaciones globales obligatorias | Cada cambio requiere reiniciar todo el sistema, generando interrupciones en la operación. |
| Baja sincronización entre procesos | Inventario y logística no se actualizan en tiempo real, provocando errores en el stock. |
| Interfaz lenta bajo carga | Durante promociones o lanzamientos, la experiencia del usuario se ve afectada por demoras. |
| Código rígido y dependiente | Pequeñas modificaciones pueden generar errores en áreas no relacionadas. |

## 

## **Diseño de la solución**

Implementaremos la solución mediante una arquitectura basada en microservicios, ya que este enfoque nos permite desarrollar el sistema como un conjunto de servicios pequeños, independientes y especializados. Cada microservicio se encarga de una funcionalidad específica del sistema y puede ser desplegado, actualizado o escalado de manera autónoma sin afectar a los demás componentes.

Esta estructura ofrece múltiples ventajas frente a las arquitecturas monolíticas tradicionales, como una mayor flexibilidad para adaptarse a cambios, mejor tolerancia a fallos, y la posibilidad de utilizar distintas tecnologías en diferentes servicios según las necesidades. Además, facilita el trabajo en equipos distribuidos y promueve prácticas de integración y entrega continua, lo que acelera los ciclos de desarrollo y mejora la mantenibilidad del sistema a largo plazo.

**Microservicios claves:**

* Gestión de Usuarios:  
   Este microservicio se encarga de todo lo relacionado con la identidad y acceso de los usuarios. Incluye funciones esenciales como autenticación, autorización y gestión de perfiles, permitiendo el registro, inicio de sesión (con soporte para métodos avanzados como OAuth y autenticación multifactor) y administración de roles (clientes, administradores, repartidores). Además, almacena información personal, historial de compras y preferencias, lo que facilita la personalización de la experiencia del usuario. Su separación como servicio independiente garantiza seguridad, escalabilidad y facilidad de integración con otros módulos, como ventas o logística, sin comprometer el rendimiento del sistema en momentos de alta demanda.
* Inventario y Catálogo:  
   Este módulo gestiona toda la información relacionada con productos, stock y disponibilidad. Permite la creación, edición y eliminación de artículos, así como la organización por categorías y la gestión de variantes (tallas, colores, etc.). También se encarga de controlar el inventario en tiempo real, evitando ventas de productos agotados y generando alertas cuando los niveles son bajos. Al estar desacoplado, puede utilizar bases de datos optimizadas para consultas rápidas, como MongoDB para el catálogo o PostgreSQL para transacciones de inventario, asegurando consistencia y disponibilidad incluso en períodos de alta actividad.
* Ventas:  
   El microservicio de ventas abarca todo el proceso comercial, desde la selección de productos en el carrito hasta la finalización del pedido. Incluye funcionalidades como checkout, integración con pasarelas de pago (Stripe, PayPal), manejo de transacciones y generación de facturas legales. Su diseño como servicio independiente permite garantizar la consistencia de las transacciones financieras (mediante bases de datos ACID) y manejar picos de demanda mediante colas de mensajería (Kafka o RabbitMQ). Además, al estar separado de otros módulos, un fallo en el soporte o la logística no interrumpe el proceso de compra.
* Logística:  
   Este servicio gestiona el envío y seguimiento de pedidos, integrando sistemas de transporte externos (FedEx, DHL) y proporcionando actualizaciones en tiempo real sobre el estado de los envíos. También puede incluir algoritmos de optimización de rutas para repartidores, utilizando APIs como Google Maps. Al ser un dominio complejo y con requisitos específicos, su separación permite adaptarse a diferentes proveedores de logística sin afectar a otros componentes del sistema. Además, puede escalar de manera independiente durante temporadas altas, como Navidad o eventos promocionales.
* Analítica y Soporte:  
   Este microservicio se enfoca en el análisis de datos y la atención al cliente. Procesa información como ventas por región, productos más vendidos y métricas de rendimiento, generando reportes para la toma de decisiones. También incluye herramientas de soporte, como chatbots con capacidades de IA para resolver consultas automáticamente o derivar casos a agentes humanos. Al estar separado, puede utilizar bases de datos analíticas (ClickHouse, Snowflake) y técnicas de procesamiento masivo (Apache Spark) sin impactar el rendimiento de los servicios transaccionales.

## 

## **Stack tecnológico**

El sistema usa una arquitectura de microservicios con Spring Boot para lógica transaccional, Node.js para servicios I/O y Quarkus para componentes de alto rendimiento. La persistencia combina PostgreSQL y Redis, mientras que Kong actúa como API Gateway para enrutamiento, seguridad y métricas, además de nuestra infraestructura basada en Amazon Web Service(AWS) .

## **Infraestructura**

Amazon Web Service (AWS)

Para el desarrollo e implementación de nuestra solución, se optó por utilizar la plataforma Amazon Web Services (AWS) debido a su solidez, disponibilidad global y amplia oferta de servicios especializados. AWS permite construir arquitecturas escalables, seguras y altamente disponibles, lo cual resulta fundamental para sistemas modernos que requieren adaptarse a cambios en la demanda, garantizar la continuidad operativa y facilitar la innovación tecnológica.

La solución se basa en una arquitectura de microservicios orquestados mediante Amazon ECS (Elastic Container Service) o Amazon EKS (Elastic Kubernetes Service), dependiendo del entorno de ejecución. Estos servicios permiten desplegar y gestionar contenedores de forma eficiente, asegurando que cada microservicio pueda escalar y evolucionar de manera independiente. Esto no solo mejora la mantenibilidad, sino que también permite una mayor agilidad en los ciclos de desarrollo y despliegue.

Para manejar variaciones en la carga de trabajo, se emplean Auto Scaling Groups, los cuales ajustan automáticamente el número de instancias de cómputo (EC2) según las necesidades del sistema. Esta capacidad de escalado automático garantiza una respuesta eficiente ante picos de tráfico sin incurrir en costos innecesarios durante momentos de baja demanda.

El almacenamiento de archivos estáticos, medios digitales y otros recursos se gestiona a través de Amazon S3, un servicio altamente duradero y escalable. Gracias a su integración con otros servicios de AWS y su compatibilidad con redes de distribución de contenido (CDN), S3 es ideal para entregar contenido de forma rápida y segura a usuarios en distintas regiones.

El monitoreo y observabilidad del sistema se centraliza mediante AWS CloudWatch, una herramienta que recopila métricas, eventos y logs en tiempo real. Esto permite detectar anomalías de forma proactiva, establecer alertas personalizadas y automatizar respuestas ante incidentes, lo que contribuye directamente a la estabilidad del sistema.

En términos de seguridad, AWS Certificate Manager se encarga de la gestión automática de certificados SSL/TLS. Este servicio garantiza comunicaciones seguras entre clientes y servidores, sin necesidad de realizar renovaciones manuales, lo que reduce riesgos de expiración de certificados y simplifica la operación.

Finalmente, se incorpora Amazon Personalize, un servicio basado en machine learning que permite ofrecer recomendaciones personalizadas a los usuarios, como productos sugeridos o contenido relevante. Esta integración potencia la experiencia del usuario y mejora las tasas de conversión en plataformas que manejan gran volumen de información personalizada.

En conjunto, esta infraestructura no solo proporciona una base sólida para la operación de la solución, sino que también permite escalar y adaptarse con facilidad a medida que evolucionan los requerimientos del negocio o crece la base de usuarios.

### 

### **API Gateway: Kong**

Kong se selecciona como API Gateway por su sistema extensible de plugins y su alto rendimiento. Este sistema permite a los desarrolladores personalizar y agregar funcionalidades adicionales según las necesidades del sistema sin comprometer la eficiencia ni la flexibilidad. Kong está diseñado para soportar grandes volúmenes de tráfico, lo que lo convierte en una opción ideal para gestionar múltiples microservicios de manera eficiente.

Punto Único de Entrada para Todas las APIs:

Una de las principales ventajas de usar Kong es que actúa como un punto único de entrada para todas las APIs del sistema. Esto centraliza el acceso a los servicios y simplifica la gestión de las conexiones entre los clientes y los microservicios, lo que ayuda a mantener un control más efectivo sobre el tráfico y las solicitudes.

Capacidades Avanzadas de Autenticación:

Kong ofrece capacidades avanzadas de autenticación, permitiendo a las organizaciones implementar mecanismos robustos de control de acceso a sus APIs. Esto incluye la integración con sistemas de autenticación como OAuth, JWT y otros estándares de seguridad, lo que garantiza que solo los usuarios o servicios autorizados puedan acceder a las APIs.

Limitación de Tasa y Balanceo de Carga:

Una característica clave de Kong es la capacidad de gestionar la limitación de tasa (rate limiting) y el balanceo de carga. Esto permite controlar la cantidad de solicitudes que un cliente puede realizar en un período determinado, protegiendo las APIs contra abusos y sobrecargas. Además, Kong distribuye el tráfico entre los microservicios de manera eficiente, asegurando que ningún servicio quede sobrecargado y que el rendimiento general del sistema sea óptimo.

Soporte para 10,000 Conexiones Concurrentes:

Kong es capaz de manejar hasta 10,000 conexiones concurrentes, lo que lo hace ideal para entornos de alta demanda. Su arquitectura escalable y distribuida permite que el sistema se mantenga estable y rápido, incluso cuando se enfrenta a grandes volúmenes de tráfico y solicitudes simultáneas.

Plugins para Integración con Keycloak:

Kong también incluye soporte para la integración con Keycloak, una solución de gestión de identidades y acceso. Mediante el uso de plugins, Kong puede gestionar la autenticación y autorización de usuarios, delegando estas tareas a Keycloak, lo que simplifica la administración de usuarios y la seguridad en la plataforma.

Analíticas Detalladas sobre Uso de APIs:

Kong ofrece herramientas analíticas avanzadas que permiten obtener métricas detalladas sobre el uso de las APIs. Esto incluye datos sobre el número de solicitudes, el tiempo de respuesta, la latencia y otros indicadores de rendimiento clave. Estas métricas ayudan a los desarrolladores y administradores a monitorear el sistema y a identificar posibles áreas de mejora o problemas de rendimiento.

Enrutamiento Inteligente entre Microservicios:

El enrutamiento inteligente es otra de las funcionalidades clave de Kong. Este sistema permite dirigir las solicitudes de manera eficiente a los microservicios adecuados, utilizando reglas y políticas definidas por los administradores. Esto optimiza el rendimiento del sistema y garantiza que las solicitudes sean procesadas por el servicio más adecuado en cada momento.

### **PostgreSQL**

Base de Datos Relacional Principal:

PostgreSQL se utiliza como la base de datos relacional principal del sistema. Es ideal para manejar datos estructurados y relaciones complejas entre entidades, lo que la convierte en la mejor opción para módulos que requieren integridad referencial y operaciones transaccionales seguras.

Transaccionalidad y Confiabilidad:

Gracias a su cumplimiento del estándar ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), PostgreSQL asegura la integridad de los datos en cada operación, siendo clave para microservicios como Ventas, Gestión de Usuarios o Inventario. Su fiabilidad comprobada lo convierte en la columna vertebral del almacenamiento de datos persistentes del sistema.

PostGIS: Soporte para Datos Espaciales:

Además, PostgreSQL incluye la extensión PostGIS, que permite el almacenamiento y consulta de datos geo-espaciales. Esto es especialmente útil en microservicios como Logística, donde se requiere procesar ubicaciones, calcular distancias, o realizar operaciones basadas en coordenadas geográficas.

### **Redis**

Almacén de Datos en Memoria para Baja Latencia:

Redis se utiliza como una base de datos en memoria que actúa como un sistema de cacheo ultrarrápido. Su principal ventaja es su capacidad de ofrecer tiempos de respuesta extremadamente bajos, lo que mejora significativamente la experiencia del usuario final, especialmente en escenarios donde la velocidad es crítica.

Caché para Consultas Frecuentes de Inventario:

Uno de los principales casos de uso de Redis es el almacenamiento en caché de consultas frecuentes, como la disponibilidad de productos en el inventario. Al mantener estos datos en memoria, se evita la necesidad de acceder a la base de datos principal en cada solicitud, lo que reduce la carga sobre PostgreSQL y mejora el rendimiento general del sistema.

Datos Geo-Espaciales:

Redis también ofrece capacidades para trabajar con datos geo-espaciales, como almacenamiento de coordenadas, cálculo de distancias y búsqueda por proximidad. Esta funcionalidad es útil en aplicaciones como la asignación de repartidores, seguimiento de entregas y sugerencia de puntos de entrega cercanos.

## **Frameworks**

La arquitectura del sistema distribuye responsabilidades entre distintos frameworks tecnológicos, seleccionados en función de las características y necesidades de cada microservicio. Esta estrategia permite optimizar rendimiento, escalabilidad y facilidad de mantenimiento, aprovechando los puntos fuertes de cada stack tecnológico.

### **Spring Boot**

Servicios Transaccionales: Inventario, Pedidos

Spring Boot es utilizado para el desarrollo de microservicios que requieren alta fiabilidad y gestión de transacciones complejas, como los servicios de Inventario y Pedidos. Gracias a su sólido soporte para operaciones ACID, manejo de errores, y conexión directa con bases de datos relacionales como PostgreSQL, es ideal para garantizar consistencia e integridad en operaciones críticas del sistema.

Su ecosistema maduro (Spring Security, Spring Data, etc.) y compatibilidad con arquitecturas basadas en microservicios hacen de Spring Boot una elección robusta para manejar lógica de negocio compleja y flujos de trabajo secuenciales, como los que se presentan durante el procesamiento de pedidos o la actualización de stock.

### **Node.js**

Servicios I/O Intensivos: Notificaciones, Chatbot

Node.js se emplea en servicios que son intensivos en entrada/salida y requieren manejo eficiente de conexiones simultáneas, como los sistemas de notificaciones en tiempo real (email, push, SMS) y chatbots de atención al cliente.

Gracias a su modelo de ejecución asíncrono y no bloqueante, Node.js es excelente para gestionar múltiples solicitudes simultáneas con un bajo consumo de recursos, sin necesidad de hilos adicionales. Esto lo convierte en una opción óptima para aplicaciones que deben responder rápidamente sin realizar procesamiento complejo en el backend.

Además, su integración sencilla con servicios externos vía HTTP y WebSocket, así como su ecosistema de librerías (npm), permite construir soluciones ligeras y modulares en menor tiempo.

### 

### **Quarkus**

Servicios Críticos en Rendimiento: Búsquedas, Pagos

Quarkus se reserva para microservicios que requieren máximo rendimiento y tiempos de respuesta mínimos, como los módulos de búsqueda de productos y procesamiento de pagos. Su diseño nativo para la nube, junto con tiempos de arranque ultrarrápidos y bajo consumo de memoria, lo convierten en la mejor opción para aplicaciones con alta carga concurrente.

Además, Quarkus permite la compilación nativa mediante GraalVM, lo que mejora aún más la eficiencia y la velocidad de ejecución, ideal para escenarios sensibles al tiempo, como las operaciones de pago donde la latencia debe mantenerse al mínimo. También es compatible con bibliotecas del ecosistema Java (JAX-RS, Hibernate, etc.), lo que facilita la transición o integración con otros servicios Java existentes.

| **Framework** | **Tipo de servicio** | **Casos de usos** | **Características claves** |
| --- | --- | --- | --- |
| Spring boot | Servicios transaccionales | Inventarios, Pedidos | Robusto, transaccionalidad, ecosistema completo |
| Node.js | Servicios y/o Intensivos | Notificaciones | Concurrencia, baja latencia, asincronía |
| Quarkus | Servicios críticos en rendimientos | Búsquedas, Pagos | Alta performance, tiempos de arranque rápidos |

**Requisitos Funcionales (RF)**

Gestión de Usuarios y Autenticación Segura: El sistema debe permitir el registro y autenticación de usuarios (clientes B2B, administradores y repartidores) con soporte para OAuth 2.0, autenticación multifactor (MFA) y gestión de roles. Cada perfil debe almacenar datos personales, historial de compras y preferencias, garantizando una experiencia personalizada y segura.

Catálogo de Productos con Gestión de Inventario en Tiempo Real: La plataforma debe ofrecer un catálogo dinámico de equipos fitness, con filtros avanzados, búsqueda por características técnicas y gestión de stock en tiempo real. Debe evitar ventas de productos agotados y generar alertas automáticas cuando los niveles de inventario sean críticos.

Proceso de Venta con Checkout y Pasarelas de Pago Integradas: El sistema debe incluir un carrito de compras, flujo de checkout optimizado y soporte para múltiples pasarelas de pago (Stripe, PayPal). Además, debe generar facturas legales automáticamente y manejar transacciones con consistencia ACID para evitar errores financieros.

Logística Internacional con Seguimiento en Tiempo Real: La plataforma debe calcular automáticamente costos de importación, plazos de entrega y trámites aduaneros. Además, debe integrarse con transportistas (FedEx, DHL) para ofrecer tracking en tiempo real y notificaciones de estado a los clientes.

Herramientas B2B para Cotizaciones Masivas y Planificación de Espacios: El sistema debe permitir a los gimnasios generar cotizaciones masivas, comparar equipos técnicamente y visualizar cómo se integrarían físicamente en sus instalaciones mediante un módulo de planificación 3D (con Three.js).

Chatbot de Soporte con IA para Consultas Automatizadas: Debe implementarse un chatbot con capacidades de procesamiento de lenguaje natural (NLP) para resolver consultas frecuentes, derivar casos complejos a agentes humanos y ofrecer asistencia 24/7.

Reportes y Analítica Comercial Personalizada: La plataforma debe generar dashboards con métricas clave (ventas por región, productos más vendidos, conversión) y permitir la exportación de reportes en formatos estándar (PDF, Excel) para la toma de decisiones estratégicas.

**Requisitos No Funcionales (RNF)**

Arquitectura de Microservicios Escalable y Tolerante a Fallos: El sistema debe estar dividido en microservicios independientes (Spring Boot, Node.js, Quarkus) con capacidad de escalar horizontalmente en AWS (ECS/EKS) durante picos de demanda, garantizando alta disponibilidad (99.9% SLA).

Rendimiento Óptimo con Baja Latencia: Los tiempos de respuesta deben ser inferiores a 500 ms para operaciones críticas (búsqueda de productos, checkout), utilizando Redis como caché y bases de datos optimizadas (PostgreSQL para transacciones, MongoDB para catálogos).

Seguridad Avanzada y Cumplimiento de Estándares: Implementación de arquitectura Zero Trust, encriptación end-to-end (TLS 1.3), certificación PCI DSS Nivel 1 para pagos, y gestión centralizada de identidades con Keycloak integrado en Kong API Gateway.

Sincronización en Tiempo Real entre Módulos: Event-driven architecture (con Kafka o RabbitMQ) para garantizar consistencia eventual entre servicios (ej: actualización inmediata de inventario tras una venta).

Disponibilidad Multi-Región y Recuperación ante Desastres: La infraestructura en AWS debe incluir réplicas en al menos dos regiones (ej: São Paulo y Santiago) con backups automatizados y estrategias de failover para garantizar continuidad operativa.

Accesibilidad y Experiencia de Usuario Responsiva: Interfaz web compatible con dispositivos móviles y desktop, siguiendo pautas WCAG 2.1 para accesibilidad, y carga de página en menos de 2 segundos (optimizada con CDN y S3).

Sostenibilidad y Optimización de Costos: Uso de auto-scaling en AWS para ajustar recursos según demanda real, monitorización continua con CloudWatch para identificar ineficiencias, y compilación nativa (Quarkus + GraalVM) para reducir consumo energético.

### **Herramientas colaborativa: Jira**

Para la planificación, seguimiento de avances y coordinación del equipo, utilizamos Jira como herramienta principal de gestión de proyectos. A través de esta plataforma, organizamos el trabajo en tableros Scrum, definimos tareas, bugs y sprints, y asignamos responsables.

Jira nos permite documentar el estado de cada tarea, registrar el historial de cambios y mantener una trazabilidad clara del progreso del proyecto. Además, se utiliza en reuniones de seguimiento para revisar el avance, identificar bloqueos y actualizar prioridades, facilitando una comunicación fluida y una mejor toma de decisiones.

| **Tabla de tareas** | | | | |
| --- | --- | --- | --- | --- |
| **Tarea** | **Responsable** | **Descripción** | **Objetivo** | **Prioridad** |
| **Análisis del sistema monolítico actual** | Equipo de arquitectura. | Mapear los módulos actuales del sistema, identificar interdependencias y puntos críticos de rendimiento. | Documentar cuellos de botella y definir puntos de desacoplamiento. | Alta. |
| **Diseño inicial de arquitectura basada en microservicios** | Arquitecto de software. | Proponer una estructura modular de servicios , definir límites de contexto y flujos de comunicación. | Garantizar escalabilidad, autonomía de servicios y separación de responsabilidades. | Alta. |
| **Evaluación de requisitos de seguridad y privacidad** | Especialista DevSecOps. | Analizar necesidades de autenticación, cifrado de datos, control de acceso y cumplimiento normativo. | Proteger la información de los usuarios y mantener la confianza comercial. | Media. |

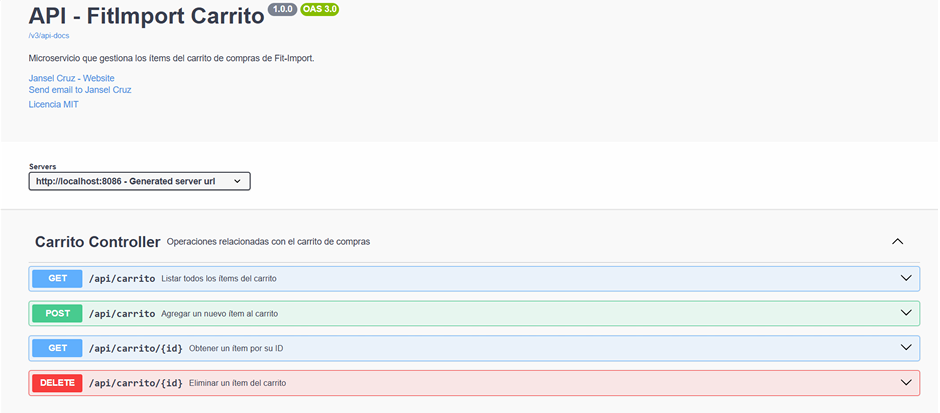
### **Documentación Técnica Swagger**

Con el propósito de respaldar el desarrollo y verificar el funcionamiento de los microservicios implementados, se realizaron pruebas utilizando **Swagger UI**. Esta herramienta facilita la visualización y prueba de endpoints expuestos por cada microservicio, asegurando el correcto intercambio de datos entre el frontend y el backend. A continuación se presenta la evidencia gráfica correspondiente a cada microservicio:

### **8.1 Carrito**

El microservicio de **Carrito** permite operaciones CRUD sobre los elementos que los usuarios agregan para futuras compras. Incluye endpoints para listar todos los ítems, obtener un ítem específico, insertar nuevos productos y eliminar registros.

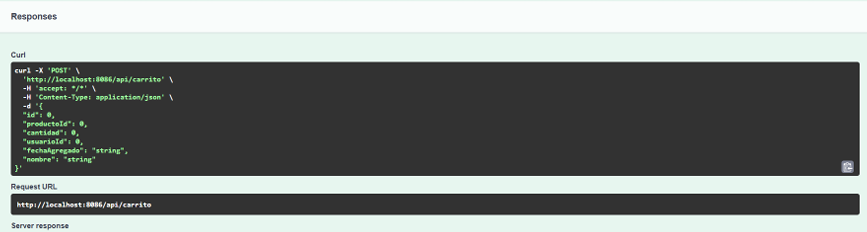
**Vista general de Swagger para Carrito**



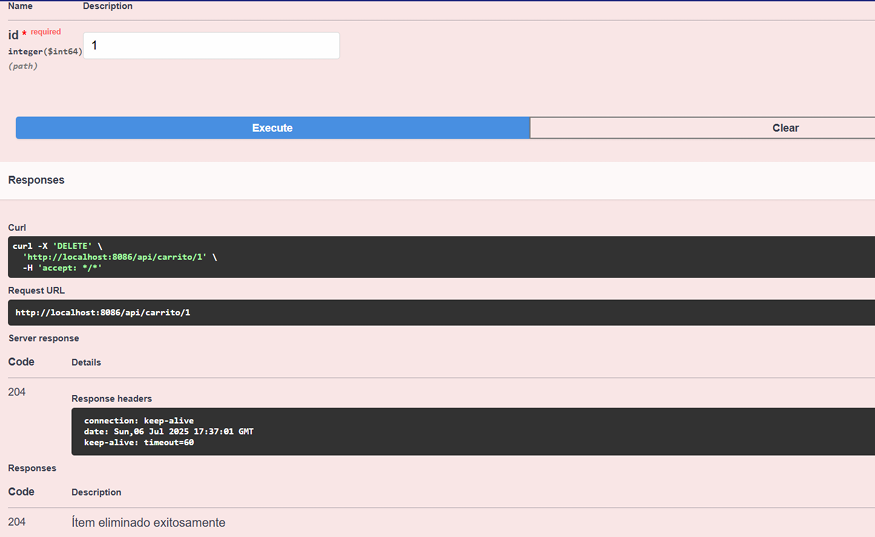
**Vista del GET para obtener todo el carrito**



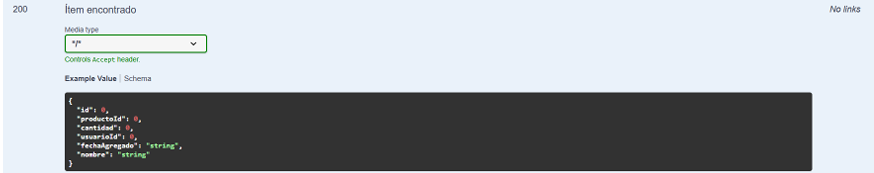
**Vista del POST**



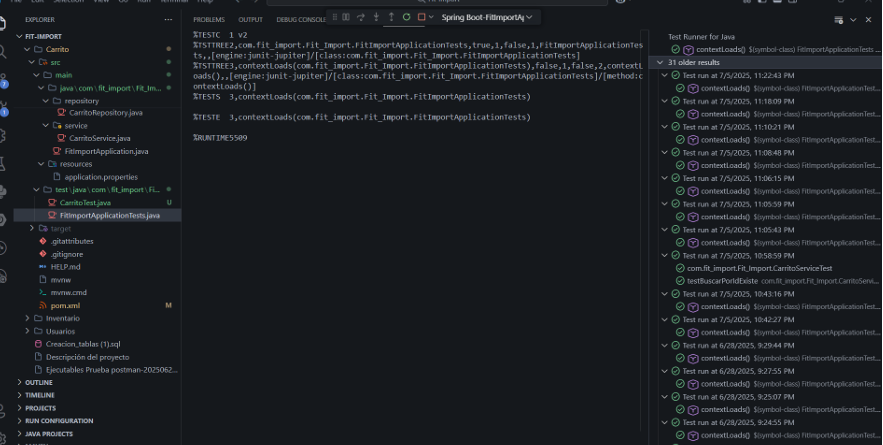
**Vista del DELETE**



**Vista del GET para obtener un ítem específico**



**Tests realizados**

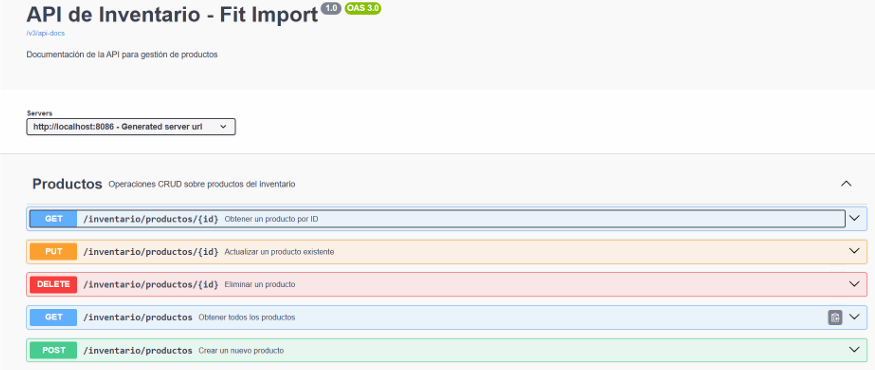


Todos los endpoints del microservicio Carrito responden correctamente con códigos HTTP 200 o 201 para las operaciones exitosas, y retornan mensajes de error claros en caso de solicitudes incorrectas. Esto garantiza una correcta integración con el frontend, especialmente durante operaciones críticas como checkout.

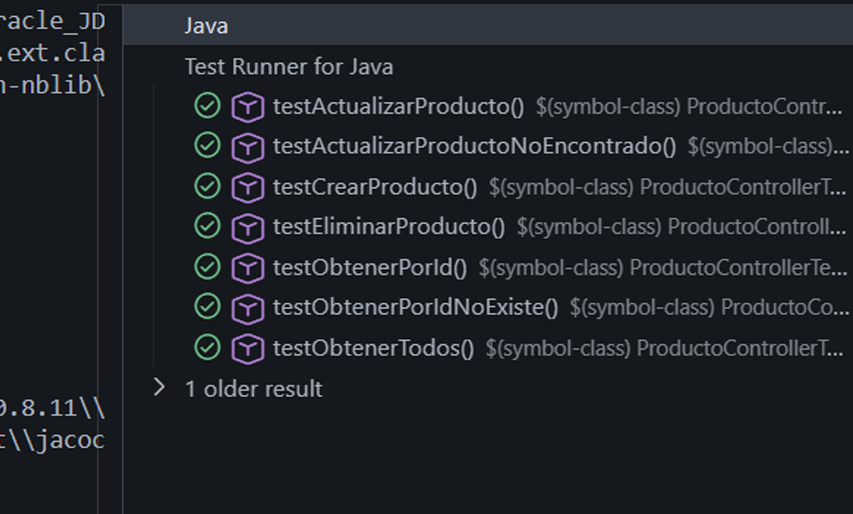
### **8.2 Inventario**

El microservicio de **Inventario** es fundamental para la gestión de stock. Permite registrar productos, actualizar existencias y consultar disponibilidad en tiempo real. Está conectado a bases de datos optimizadas para consultas rápidas y permite prevenir ventas de productos agotados.

**Vista general de Swagger para Inventario**



**Tests realizados**

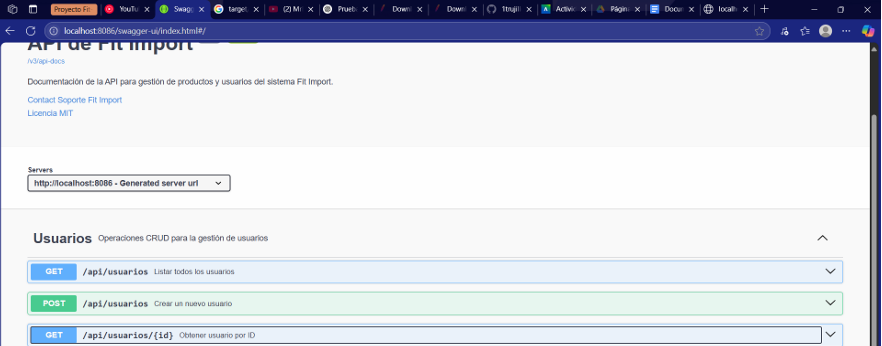


El microservicio Inventario está protegido por mecanismos de autenticación y expone endpoints RESTful para integrarse con otros servicios, como el Carrito y el módulo de logística, permitiendo mantener sincronizada la información de stock.

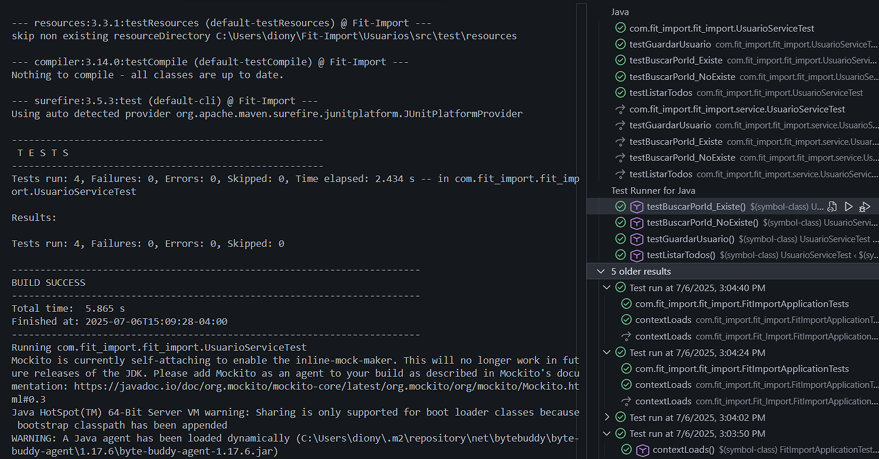
### **8.3 Usuario**

El microservicio de **Usuario** gestiona toda la información relacionada con las cuentas de los clientes y su seguridad. Incluye autenticación, registro y gestión de perfiles. Está diseñado para integrarse con herramientas como Keycloak y el API Gateway Kong para asegurar el control de acceso.

**Vista general de Swagger para Usuario**



**Tests realizados**



Gracias a la arquitectura basada en microservicios, este módulo puede escalar de forma independiente para atender picos de tráfico, asegurando tiempos de respuesta bajos y una experiencia segura para el usuario final.

### **Conclusión**

El desarrollo de la plataforma e-commerce Full Stack para FIT-IMPORT representa un hito estratégico que transformará radicalmente la operación y el crecimiento del negocio. Con la documentación técnica generada y validada a través de Swagger, se ha comprobado la correcta funcionalidad de los microservicios clave: Carrito, Inventario y Usuario. Esto refuerza la confianza en la solución propuesta y garantiza que FIT-IMPORT contará con una plataforma robusta, escalable y segura, preparada para enfrentar las demandas del mercado fitness en Chile y eventualmente de toda la región.

A través de la implementación de una arquitectura basada en microservicios y tecnologías de vanguardia, hemos creado no solo una solución tecnológica robusta, sino un ecosistema completo que resuelve los desafíos críticos de importación, logística, gestión de inventario y experiencia del cliente. Esta plataforma permitirá a FIT-IMPORT escalar de manera eficiente, reducir costos operativos mediante procesos automatizados y ofrecer una experiencia de compra superior con funcionalidades innovadoras como visualización 3D de productos, recomendaciones personalizadas y seguimiento en tiempo real de pedidos internacionales. La adopción de infraestructura cloud en AWS garantiza flexibilidad, seguridad y alta disponibilidad, mientras que nuestro enfoque en desarrollo ágil asegura la capacidad de adaptación continua a las cambiantes demandas del mercado. Más allá de los beneficios técnicos, este proyecto refleja nuestro compromiso con la excelencia operacional, la satisfacción del cliente y la sostenibilidad empresarial. Al integrar prácticas responsables como la optimización de recursos y la reducción de huella de carbono, FIT-IMPORT no solo crecerá como negocio, sino que lo hará de manera ética y alineada con los valores de sus clientes.

En conclusión, esta plataforma no es solo una herramienta de ventas, sino la base tecnológica que impulsará el futuro de FIT-IMPORT, permitiéndole expandirse, innovar y mantener su ventaja competitiva en el dinámico mundo del fitness. Con una implementación sólida, un equipo comprometido y una visión clara, estamos seguros de que este proyecto marcará un antes y un después en la industria, beneficiando tanto a la empresa como a sus clientes en Chile y, potencialmente, en toda la región.