Especificación de requisitos de software

Proyecto: Caso Perfulandia

Revisión*: [99.99]*

[Seleccionar fecha]

Contenido

Ficha del documento

Fecha	Revisión	Autor	Modificación

Documento validado por las partes en fecha:

Por el cliente	Por la empresa suministradora
[Firma]	[Firma]
Sr./Sra.	Sr./Sra.

1. Introducción

En este documento se detallará el proceso y los pasos a seguir para que el actual software de Perfulandia pase de una arquitectura monolítica a una arquitectura de microservicios. Esto con el propósito de que el software sea más escalable y flexible en el tiempo, debido a que la empresa ha comenzado a expandirse.

1.1. Propósito

El propósito de este documento es brindar una guía detallada a los involucrados en este proyecto sobre que es lo que se va a hacer. Cuales son los requerimientos de alto nivel, y cuales son los requisitos funcionales y no funcionales.

Este documento va dirigido a los desarrolladores, stakeholders, gerentes y equipos de testing.

1.2. Ámbito del Sistema

- El nombre del sistema/software es OMNIA.

•Lo que hará:

- Gestionar usuarios: Creación de usuarios, modificación de usuarios.
- Gestión de Inventario: Control de stock, se podrá actualizar las cantidades de cada producto y se podrá visualizar el stock en tiempo real.
- Gestión de venta: Se podrá registrar las ventas, se generan facturas, se podrán hacer ventas de forma segura.
- Gestión de logística: Se podrá visualizar el estado de los envíos y supervisar y gestionar estos mismos.
- Portal web para clientes: Los clientes podrán registrarse, podrán hacer pedidos, consultar su historial de compras, agregar productos al carrito, dejar reseñas y recibir ayuda del soporte.

• Lo que no hará:

- No se incluirá fabricación de productos: Solo se harán ventas de productos, no la fabricación de estos.
- El sistema no será capaz de hacer funciones de trabajos de contabilidad.

•BENEFICIOS :

- Alta escalabilidad.
- Alta flexibilidad.
- Reducción de costos.
- Alta eficiencia en reparar fallos del software.

Nombre del software:

Omnia

Con este software se espera que la empresa “Perfulandia” consiga una expansión exitosa debido a sus altas demandas, y si en un futuro sigue creciendo, sea fácil de escalar los requisitos requeridos acorde a las necesidades.

1.3. Definiciones, Acrónimos y Abreviaturas

En este ERS ocuparemos acrónimos, definiciones y abreviaturas como:

*RF = Requisito funcional * BD = Base de datos

*RNF = Requisito no funcional

1.4. Referencias

https://spring.io/projects/spring-security

https://auth0.com/es/intro-to-iam/what-is-oauth-2

https://en.wikipedia.org/wiki/JSON_Web_Token

https://www.skysnag.com/es/blog/what-is-bcrypt/

https://www.theserverside.com/definition/object-relational-mapping-ORM#:~:text=Object%2Drelational%20mapping%20(ORM)%20is%20a%20way%20to%20align,is%20disabled%20in%20your%20browser.

https://ifgeekthen.nttdata.com/s/post/que-es-java-hibernate-por-que-usarlo-MC5FU56AIPGBGIHNJ677RBIXUHOI?language=es

https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/

https://www.redhat.com/es/topics/containers/what-is-docker

https://aws.amazon.com/es/compare/the-difference-between-rabbitmq-and-kafka/#:~:text=RabbitMQ%20es%20un%20agente%20de%20mensajes%20de%20uso%20general%20que,de%20macrodatos%20en%20tiempo%20real.

https://whitestack.com/es/blog/prometheus-y-grafana/#:~:text=Prometheus%20y%20Grafana%2C%20aunque%20poderosos,de%20manera%20intuitiva%20y%20atractiva.

https://www.ionos.com/es-us/digitalguide/paginas-web/creacion-de-paginas-web/lazy-loading

https://www.ionos.com/es-us/digitalguide/hosting/cuestiones-tecnicas/que-es-memcached/

https://www.dremio.com/wiki/database-indexing/

https://en.wikipedia.org/wiki/Payment_Card_Industry_Data_Security_Standard

1.5. Visión General del Documento

A continuación se describirán los factores relacionados con nuestro software (Apis de terceros, uso de nube) y posibles mejoras a futuro considerando la situación del mercado actual.

Luego en el punto 3 se especificarán con detalle los RF y los RNF.

2. Descripción General

En esta sección se describen todos aquellos factores que afectan al producto y a sus requisitos. No se describen los requisitos, sino su contexto. Esto permitirá definir con detalle los requisitos en la sección 3, haciendo que sean más fáciles de entender.

Normalmente, esta sección consta de las siguientes subsecciones: Perspectiva del producto, funciones del producto, características de los usuarios, restricciones, factores que se asumen y futuros requisitos.

2.1. Perspectiva del Producto

Uno de los alcances principales del proyecto es lograr la migración de una arquitectura monolítica a una arquitectura de microservicios, para ello necesitaremos una base de datos, una buena gestión de sucursales, un portal web de clientes y una buena capacitación para los empleados y administradores del software que les daremos.

Nuestro software funcionará con Apis de tipo Rest. Tales como:

- Webpay: Para el procesamiento de pagos en linea.
- Chilexpress: El sistema tendrá las api de Chilexpress para gestionar los envios.

2.2. Funciones del Producto

El sistema contará con las siguientes funciones clave:

- Módulo de administración: Gestión de usuarios, permisos, monitoreo del sistema, y respaldo de datos.

- Módulo de ventas: Registro de ventas, atención a devoluciones y reclamaciones, emisión de facturas y consultas de inventario.

- Módulo de sucursales: Gestión de inventario, creación de reportes, gestión de pedidos y configuración de sucursales.

- Módulo de logística: Gestión de envíos, optimización de rutas de entrega y gestión de proveedores.

- Portal web para clientes : Funcionalidades para que los clientes creen cuentas, naveguen por productos, realicen pedidos, consulten el historial de compras y gestionen su perfil.

2.3. Características de los Usuarios

Esta subsección describe las características generales de los usuarios del producto, incluyendo nivel educacional, experiencia y experiencia técnica.

Usuarios del software:

- Administrador del sistema:

Nivel educacional: Formación técnica o superior en administración o informática.

Experiencia técnica: Experiencia en administración, base de datos e informática.

- Gerente de sucursal:

Nivel educacional: Formación profesional en administración de empresas.

Experiencia técnica: Conocimiento básico en manejo de softwares de gestión.

- Empleado de ventas:

Nivel educacional: Educación media completa.

Experiencia técnica: Conocimientos básicos en el uso de softwares de ventas.

- Operario de logística:

Nivel educacional: Educación técnica o profesional en el área de logística.

Experiencia técnica: Conocimientos en el área de logística y softwares de logística.

- Clientes:

Nivel educacional: Ninguno.

Experiencia técnica: Conocimientos básicos de uso de internet y comercio electrónico.

2.4. Restricciones

Esta subsección describe aquellas limitaciones que se imponen sobre los desarrolladores del producto:

- Políticas de la empresa: Se deberán cumplir medidas de seguridad acorde a la empresa.
- Limitaciones del hardware: Se deberá conocer cuál es el presupuesto del que se dispone para conseguir hardware para llevar a cabo el software.
- Interfaces con otras aplicaciones: El sistema tendrá integrado Apis de terceros como webpays y chilexpress.
- Operaciones paralelas: El software podrá realizar operaciones paralelas.
- Funciones de auditoría: Se podrá conocer las acciones realizadas por cada usuario.
- Funciones de control: Se podrá pausar por partes o por completo el software en caso de mantenimiento.
- Lenguaje(s) de programación: Java (con Spring Boot), y utilizará tecnologías como Docker, Kubernetes y bases de datos relacionales
- Protocolos de comunicación: El sistema utilizara Apis de tipo Rest.

• Requisitos de habilidad: Todos los usuarios correspondientes a sus áreas deberán tener un básico conocimiento sobre el uso de un computador y dispositivo móvil.

• Criticalidad de la aplicación: El sistema debe ser altamente disponible (99.9% de uptime), con mecanismos de recuperación ante fallos.

• Consideraciones acerca de la seguridad: Implementación de SSL/TLS, autenticación multifactor (MFA), y protección contra ataques como XSS e inyección SQL.

2.5. Suposiciones y Dependencias

SUPOSICIONES

- Las sucursales tienen que tener conexión a internet de manera estable
- los empleados deben de tener una capacitación básica acerca del sistema

DEPENDENCIAS

- Apis de terceros : Webpay(Mercadopago) , chilexpress (envíos), Google maps.
- Nube de AWS.

Nuestro software utiliza Apis de terceros como webpay y chilexpress, por lo que si se quiere cambiar este sistema, se deberán alterar los requisitos.

También utilizaremos la nube para desplegar el software, en este caso AWS, por lo que si se quiere cambiar esto, se deberán revisar los requisitos.

2.6. Requisitos Futuros

Considerando cómo está avanzando la informática en cuanto a temas de IA, en un futuro se podría considerar implementar algún tipo de funcionalidad relacionada con IA. Como por ejemplo un asistente virtual que recomiende productos en función de lo que más busca en Google el cliente.

3. Requisitos Específicos

3.1 Requisitos comunes de los interfaces

- **Componentes Comunes:**
- Barra de navegación superior con acceso a módulos según rol.
- Menú lateral
- Páginas de login y registro con validación visual en tiempo real.
- ¨Panel de administración
- Interfaz de cliente: catálogo con filtros, carrito de compras, checkout paso a paso.

3.1.1 Interfaces de usuario

Por ahora no se ha tratado con el cliente el tema de la interfaz, pero tenemos la siguiente base:

- **Diseño responsive**: La ui se adaptara a distintos tamaños (PC, tablet, celular).
- **Estilo visual**: Paleta de colores personalizable, con diseño minimalista, uso de Material Design o Tailwind CSS.
- **Multilingüe y multi moneda:** El usuario podrá seleccionar idioma y moneda desde el header del sitio.

3.1.2 Interfaces de hardware

Servidores Backend:

- Procesador: mínimo 4 vCPU.
- Memoria RAM: 8 GB por instancia de microservicio.
- Almacenamiento: SSD, redundante, mínimo 100 GB por nodo.

Estaciones de trabajo (para operadores administrativos):

- Navegador actualizado (Chrome, Firefox, Edge).
- Conexión a internet estable.
- Impresoras compatibles para facturación (PDF o conexión a servicios de impresión).

Dispositivos móviles (clientes):

- Android e iOS: acceso vía navegador.
- Pantallas táctiles compatibles con el diseño UI responsivo.

3.1.3 Interfaces de software

- Spring Boot (Framework de java que permite crear microservicios, tambien simplifica el despliegue de aplicaciones web, apis de tipo rest, etc.):
- **Propósito del Interfaz**
- Desarrollo de microservicios backend
- **Definición del Interfaz**
- APIs RESTful sobre HTTP con formato JSON.
- PostgreSQL(es una base de datos relacional):
- **Propósito del Interfaz**
- Base de datos relacional principal
- **Definición del Interfaz**
- ORM con Hibernate / JDBC, acceso vía JPA.
- RabbitMQ / Kafka(

  Rabbit es un broker de mensajes, usado para comunicar microservicios entre si.

  Kafka: Sirve para procesar muchos eventos a tiempo real.):
- **Propósito del Interfaz**
- Comunicación asíncrona entre microservicios
- **Definición del Interfaz**
- Cola de mensajes con formatos JSON o Protobuf.
- Redis / Memcached(Redis: Es un sistema de almacenamiento que sirve para almacenar cache y acceder de forma rápida a los datos):

- **Propósito del Interfaz**
- Caché distribuida para consultas rápidas
- **Definición del Interfaz**
- Claves y valores serializados en JSON.
- OAuth 2.0 / JWT(OAuth2 es un protocolo de autorización

  JWT es un sistema de tokens para evitar solicitar credenciales al usuario al transmitir identidad entre servicios..):
- **Propósito del Interfaz**
- Gestión de autenticación y autorización
- **Definición del Interfaz**
- Cabeceras HTTP con tokens JWT Bearer.
- Google Maps API(Api de google para obtener mapa y rutas optimizadas):
- **Propósito del Interfaz**
- Cálculo y visualización de rutas de entrega
- **Definición del Interfaz**
- Llamadas HTTPS con autenticación por token.
- ElasticSearch / Kibana(

ElasticSearch: Motor de búsqueda que indexa logs y datos para poder consultarlos rápidamente.

Kibana: Herramienta visual para explorar los datos de ElasticSearch, hacer dashboards, alertas, etc.
):

- **Propósito del Interfaz**
- Monitoreo, logging y métricas
- **Definición del Interfaz**
- Logs estructurados enviados por Logstash.

- MercadoPago(Api que permite acceder a una plataforma de pago segura):
- **Propósito del Interfaz**
- Procesamiento de pagos seguros
- **Definición del Interfaz**
- APIs HTTPS con tokens y cumplimiento PCI-DSS.

3.1.4 Interfaces de comunicación

El software está basado en una arquitectura de microservicios, por lo que los protocolos de comunicación entre servicios y componentes son muy importantes.

Comunicación entre Microservicios:

- **REST (HTTP/HTTPS):** Principal protocolo de comunicación entre servicios, de forma segura.
- **Mensajería Asíncrona (RabbitMQ / Kafka):** Para eventos del sistema, colas de pedidos, facturación, y sincronización de inventario.
- **Formato de mensajes:** JSON.

Frontend – Backend:

- **HTTPS con JSON:** Intercambio de datos entre frontend web y microservicios.

Protocolos de Seguridad:

- **TLS/SSL:** Todas las comunicaciones deben estar cifradas.
- **OAuth 2.0 / OpenID Connect:** Para autenticación de usuarios y autorización basada en roles.

.

Red interna de servicios (en Kubernetes):

- Comunicación entre pods usando DNS interno, Service Mesh opcional para trazabilidad y control de tráfico.

3.2 Requisitos funcionales

R1: Administrador del Sistema

• Gestionar Usuarios: Crear, actualizar, desactivar y eliminar cuentas de usuarios del sistema.

• Configurar Permisos: Asignar y modificar permisos de acceso a diferentes módulos y funciones del sistema.

• Monitorización del Sistema: Visualizar el estado del sistema, recibir alertas sobre posibles fallos y monitorizar el rendimiento.

• Respaldar y Restaurar Datos: Realizar copias de seguridad periódicas y restaurar datos en caso de pérdida o fallo.

R2: Gerente de Sucursal

• Gestionar Inventario: Agregar, actualizar y eliminar productos del inventario, y ajustar cantidades de stock.

• Generar Reportes: Crear reportes de ventas, inventario y rendimiento de la sucursal.

• Gestionar Sucursales: Configurar detalles específicos de la sucursal, como horarios de apertura, personal asignado, y políticas locales.

• Gestionar Pedidos: Supervisar y autorizar pedidos de productos para reabastecer el inventario de la sucursal.

R3: Empleado de Ventas

• Registrar Ventas: Procesar transacciones de venta en el sistema, aplicando descuentos y ofertas cuando sea necesario.

• Atender Devoluciones y Reclamaciones: Procesar devoluciones de productos y gestionar reclamaciones de clientes.

• Consultar Inventario: Verificar disponibilidad de productos en tiempo real y buscar productos en el inventario.

• Generar facturas: Emitir facturas electrónicas y enviarlas por correo electrónico a los clientes.

R4: Logística

• Gestionar Envíos: Crear, actualizar y seguir envíos de productos desde la bodega hasta las sucursales y los clientes.

• Optimizar Rutas de Entrega: Planificar y optimizar rutas de entrega para los pedidos locales y regionales.

• Actualizar Estado de Pedidos: Cambiar el estado de los pedidos en el sistema, desde el procesamiento hasta la entrega final.

• Gestionar Proveedores: Mantener y actualizar la información de proveedores, realizar pedidos de reabastecimiento y gestionar la recepción de mercancías.

R5: Acciones de los Clientes vía Web

• Crear Cuenta: Registrarse en la plataforma web proporcionando la información necesaria como nombre, dirección de correo electrónico y dirección de envío.

• Iniciar sesión: Acceder a la cuenta utilizando las credenciales de usuario.

• Navegar y Buscar Productos: Explorar el catálogo de productos, utilizando filtros y la barra de búsqueda para encontrar productos específicos.

• Agregar Productos al Carrito: Seleccionar productos y añadirlos al carrito de compras.

• Realizar pedidos: Completar el proceso de compra, proporcionando detalles de pago y seleccionando opciones de envío.

• Consultar Historial de Pedidos: Ver el historial de compras anteriores y el estado actual de los pedidos en curso.

• Gestionar Perfil: Actualizar información personal, direcciones de envío y detalles de pago.

• Solicitar Soporte: Enviar consultas o problemas a través de un formulario de contacto o chat en línea.

• Dejar Reseñas y Calificaciones: Evaluar productos comprados y dejar comentarios en el sitio web.

• Aplicar Cupones y Descuentos: Ingresar códigos promocionales durante el proceso de compra para recibir descuentos.

3.3 Requisitos no funcionales

R1: Administrador del Sistema

- **Paginación de Usuarios**: Uso de paginación en los listados de usuarios para evitar sobrecargar la base de datos y el servidor.
- **Hashing de contraseñas**: Utilización de hashing de contraseñas con algoritmos como BCrypt en el backend.
- **Seguridad contra Inyecciones y XSS**: Protección contra ataques de inyección SQL y Cross-Site Scripting (XSS) a través de validaciones adecuadas y la utilización de ORM (como Hibernate) para interactuar con la base de datos.
- **Roles y Permisos Dinámicos**: Implementar roles y permisos dinámicos en base de datos, utilizando Spring Security y RBAC (Role-Based Access Control), para escalar sin afectar el rendimiento.
- **Patrones DTO y DAO**: Utilizar patrones como DTO (Data Transfer Object) y DAO (Data Access Object) para desacoplar la lógica de negocio de la base de datos, facilitando futuras actualizaciones.
- **Monitoreo 100% del Tiempo**: Los sistemas de monitoreo deben estar activos en 100% del tiempo, con un tiempo de inactividad inferior a 1 hora por año.
- **Backups Geográficamente Distribuidos**: Los backups deben ser distribuidos en diferentes zonas geográficas para evitar la pérdida de datos.

R2: Gerente de Sucursal

- **Transacciones Rápidas de Inventario**: Las transacciones de inventario deben completarse en menos de 500 ms, incluso durante picos de carga.
- **Caché Distribuida**: Uso de caché distribuida con Redis o Memcached para almacenar información de inventario más frecuente y mejorar el rendimiento.
- **Escalabilidad de Microservicios**: Cada microservicio relacionado con el inventario debe poder escalar de manera independiente a través de Kubernetes y Docker, ajustando el número de instancias según la carga.
- **Optimización de Consultas para Reportes**: Las consultas a la base de datos para generar reportes deben optimizarse con Índices y consultas agregadas para minimizar el tiempo de respuesta.

R3: Empleado de Ventas

- **Alta Disponibilidad y Transacciones Simultáneas**: Java y Spring Boot deben ser configurados para manejar múltiples transacciones simultáneamente con cola de mensajes (por ejemplo, RabbitMQ o Kafka) para asegurar la alta disponibilidad.

- **Escalabilidad en Solicitudes de Devoluciones**: El sistema debe ser capaz de manejar un número creciente de solicitudes de devoluciones sin afectar el rendimiento general del sistema.
- **Disponibilidad 24/7**: La funcionalidad debe estar disponible las 24 horas, 7 días a la semana, sin interrupciones.
- **Transacciones Seguras de Facturación**: Las facturas deben ser enviadas a través de canales seguros (por ejemplo, usando **TLS/SSL**) para proteger la información financiera de los clientes.
- **Autenticación Fuerte para Facturación**: Implementar autenticación fuerte (OAuth o MFA) para garantizar que solo usuarios autenticados puedan generar y enviar facturas.

R4: Logística

- **Gestión Escalable de Envíos**: El sistema de gestión de envíos debe poder manejar múltiples envíos simultáneamente utilizando microservicios escalables a través de Kubernetes.
- **Actualizaciones en Tiempo Real**: Las actualizaciones de estado de los envíos deben reflejarse en tiempo real, con latencia inferior a 500 ms.
- **Optimización de Rutas de Entrega**: Usar algoritmos eficientes de optimización de rutas (como Google OR-Tools o algoritmos de Ruta más corta).

R5: Acciones de los Clientes vía Web

- **Lazy Loading de Productos**: Utilizar técnicas de lazy loading para cargar los productos solo cuando sea necesario, mejorando la experiencia del usuario.
- **Transacciones Seguras con HTTPS**: Implementar HTTPS (SSL/TLS) para asegurar las transacciones y datos personales.
- **Cumplimiento de Estándares PCI DSS**: El proceso de pago debe ser compatible con los estándares PCI DSS.
- **Soporte 24/7 Escalable**: El sistema de soporte debe funcionar correctamente las 24 horas del día, los 7 días de la semana, y ser escalable para soportar un alto volumen de solicitudes simultáneas.
- **Alta Disponibilidad del Sitio Web**: El sitio web debe estar disponible el 99.9% del tiempo, con un tiempo máximo de inactividad de 8.76 horas al año. Esto incluye la redundancia en servidores, balanceo de carga y mecanismos de recuperación ante fallos.
- **Redirección Automática ante Fallos**: En caso de que un servidor o componente del sistema falle, el sistema debe ser capaz de redirigir el tráfico de manera automática a otros servidores para mantener la disponibilidad del sitio sin interrumpir la experiencia del usuario.

- **Autenticación Robusta:** Implementar autenticación robusta (por ejemplo, con OAuth2, autenticación de dos factores) para la creación de cuentas e inicio de sesión. Se debe validar que los usuarios solo puedan acceder a las funcionalidades para las que están autorizados (privilegios de administrador, gerente, cliente, etc.).
- **Interfaz de Usuario Intuitiva**: La interfaz de usuario debe ser intuitiva y fácil de navegar, con una estructura clara que permita a los usuarios realizar acciones como crear cuentas, buscar productos, realizar pedidos, etc., con pocos clics.
- **Soporte Multilingüe y Multimoneda**: El sitio debe soportar múltiples idiomas y monedas para clientes de diferentes regiones, con la capacidad de cambiar entre diferentes configuraciones regionales.
- **Actualizaciones sin Interrupción del Servicio**: Las actualizaciones del sistema, como nuevas versiones o parches de seguridad, deben poder realizarse sin afectar la disponibilidad del servicio o la experiencia del usuario.
- **Software responsive**: El sitio web debe ser totalmente funcional en dispositivos móviles, incluyendo teléfonos inteligentes y tabletas, y adaptarse correctamente a diferentes tamaños de pantalla mediante diseño responsivo.

A continuación se explican las tecnologías usadas en los requerimientos no funcionales para que sea posible la migración de una arquitectura monolítica a una arquitectura de microservicios:

ORM (Object-Relational Mapping):

Es una técnica de programación que permite interactuar con bases de datos utilizando objetos en lugar de consultas SQL directas.

Hibernate es uno de los frameworks más populares para hacer ORM en Java, facilitando la conversión de datos entre objetos Java y tablas de bases de datos, lo que mejora la mantenibilidad y seguridad al evitar inyecciones SQL.

Spring Boot Actuator:

Un conjunto de herramientas dentro de Spring Boot que facilita la monitorización y administración de aplicaciones. Proporciona métricas, detalles de salud, información de la aplicación, entre otros, para asegurarse

de que el sistema está funcionando correctamente. Esto ayuda a monitorear el estado del sistema.

Prometheus y Grafana:

Prometheus: Es una herramienta de monitoreo y alerta de código abierto que recolecta métricas de diversas aplicaciones y servicios.

Grafana: Es una herramienta de visualización de métricas, que se utiliza comúnmente junto con Prometheus para crear paneles de control (dashboards) y visualizaciones en tiempo real de las métricas del sistema.

Spring Batch: Es un framework de Spring para procesar grandes volúmenes de datos, especialmente útil en la ejecución de trabajos por lotes (como respaldos de base de datos).

Utilizado para ejecutar tareas programadas, como realizar copias de seguridad (backups) de la base de datos o procesar grandes cantidades de datos de manera eficiente.

Docker y Kubernetes:

Docker: Es una plataforma que permite empaquetar una aplicación junto con todas sus dependencias (bibliotecas, configuración, etc.) dentro de un contenedor. Esto facilita la implementación y ejecución de aplicaciones de manera aislada y consistente, sin importar el entorno.

Kubernetes: Es un sistema de orquestación de contenedores que gestiona la implementación, escalado y operación de aplicaciones contenerizadas (como aquellas que utilizan Docker). Se usa para asegurar que las aplicaciones funcionen de manera eficiente en un entorno distribuido.

RabbitMQ / Kafka:

RabbitMQ: Es un broker de mensajes que permite la comunicación entre sistemas de manera asíncrona y desacoplada. Se utiliza para gestionar colas de mensajes y asegurar que los servicios se comuniquen entre sí sin necesidad de estar directamente conectados.

Kafka: Similar a RabbitMQ, Kafka es un sistema de mensajería distribuido, pero con un enfoque en la alta escalabilidad y rendimiento para manejar flujos de datos en tiempo real.

Memcached:

Es una solución de caching distribuido que almacena datos en la memoria para reducir la carga de las bases de datos y mejorar la velocidad de acceso.

Lazy Loading:

Es un patrón de diseño que retrasa la carga de un objeto hasta el momento en que se necesita por primera vez. Se usa para optimizar el rendimiento al evitar cargar datos o componentes que no se van a utilizar inmediatamente.

En el frontend, puede usarse para cargar productos en un catálogo solo cuando el usuario hace scroll, en lugar de cargar todos los productos de una vez.

Indexación de Base de Datos:

Crear índices en las bases de datos mejora la velocidad de las consultas. Un índice es una estructura de datos que mejora la rapidez en la búsqueda de registros en una base de datos, similar al índice de un libro.

Acelera las consultas, como las búsquedas de productos en el catálogo, cuando el sistema tiene muchos datos.

PCI DSS (Payment Card Industry Data Security Standard):

Es un conjunto de normas de seguridad para proteger la información de tarjetas de crédito. Es fundamental para aplicaciones que procesan pagos en línea.

Asegura que las aplicaciones de comercio electrónico sean seguras durante el procesamiento de pagos.

3.3.1 Requisitos de rendimiento

- Las operaciones de crear, actualizar y eliminar usuarios deben completarse en menos de 1 segundo bajo carga normal.
- Los procesos de restauración de datos deben completarse en menos de 5 minutos en caso de un fallo crítico.
- Los reportes deben generarse en menos de 3 segundos, incluso para grandes volúmenes de datos.
- Implementar un sistema de caching para reportes generados anteriormente, reduciendo la carga de la base de datos en cada solicitud.
- Proteger la funcionalidad con JWT y Spring Security para garantizar que solo los usuarios autorizados puedan autorizar y gestionar pedidos.
- El registro de ventas debe ser procesado en menos de 2 segundos incluso durante picos de alta carga, para no afectar la experiencia de usuario.
- La optimización de rutas debe completarse en menos de 3 segundos, incluso con rutas complejas y múltiples destinos.

- La base de datos debe ser optimizada para consultas rápidas y debe ser posible actualizar el estado de los pedidos en menos de 1 segundo.
- La búsqueda de productos debe ser eficiente, con tiempos de respuesta inferiores a 1 segundo. Implementar indexación y caching para los filtros y resultados frecuentes.
- El proceso de compra debe completarse en menos de 5 segundos, incluso bajo condiciones de alta demanda, mediante optimización de formularios y procesamiento asíncrono.
- Las consultas deben ser registradas y procesadas en **menos de 2 segundos**, proporcionando una respuesta inmediata al cliente.

3.3.2 Seguridad

Spring Security: Es un marco de seguridad para aplicaciones Java que proporciona autenticación y autorización robustas. Es utilizado para proteger aplicaciones contra ataques comunes como Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF) y ataques de inyección SQL.

OAuth2 / JWT: OAuth2: Es un protocolo de autorización que permite que una aplicación pueda acceder a recursos de otra aplicación sin tener que compartir contraseñas. Es comúnmente usado para delegar acceso entre servicios.

JWT (JSON Web Token): Es un formato compacto y autónomo para transmitir información de forma segura entre las partes como un objeto JSON. Generalmente se utiliza para la autenticación y autorización en sistemas distribuidos.

BCrypt: Un algoritmo de hashing utilizado para almacenar contraseñas de manera segura. Se utiliza en Spring Security para proteger las contraseñas de los usuarios, aplicando un proceso que incluye un "sal" (salting) y múltiples rondas de hashing, lo que hace muy difícil descifrar las contraseñas.

3.3.3 Fiabilidad

Esperamos que gracias a la implementación de microservicios, el sistema falle poco o no falle, en caso de fallas, se responderá de manera inmediata y veloz para dar una pronta solución.

3.3.4 Disponibilidad

Esperamos que el software esté disponible el 99% del tiempo, para que los usuarios puedan interactuar con él de manera eficiente y segura.

3.3.5 Mantenibilidad

Para la mantenibilidad haremos uso de backups automáticos, lo que facilitará el mantenimiento.

Gracias a esto y a que el software contará con microservicios según áreas, lograremos una dinámica mantenibilidad.

3.3.6 Portabilidad

Para que el software sea portable utilizaremos tecnologías multiplataforma como Java y Docker.

-Uso de bases de datos estándar y ORMs para facilitar la migración.

-Compatibilidad con sistemas operativos Windows, Linux y macOS.

4. Plan de migración de MicroServicios

4.1. Descomposición por dominio

Problema a resolver:

El monolito actual tiene módulos acoplados (ej: ventas, inventario y CRM), lo que genera cuellos de botella al escalar.

Solución:

Se aplicó DDD para identificar Bounded Contexts (límites de responsabilidad):

Sales-Service: Procesamiento de transacciones (POS).

Inventory-Service: Gestión de stock en tiempo real.

Customer-Service: Datos de clientes y promociones.

Justificación :

Bajo acoplamiento: Cada servicio puede evolucionar y escalar independientemente.

Alta cohesión: Las funcionalidades se agrupan por dominio de negocio (ej: no mezclar lógica de ventas con inventario).

4.2. Comunicación asíncrona con Event Driven Architecture

Problema a resolver:

El monolito actual tiene módulos acoplados (ej: ventas, inventario y CRM), lo que genera cuellos de botella al escalar.

Solución:

Se aplicó DDD para identificar Bounded Contexts (límites de responsabilidad):

Sales-Service: Procesamiento de transacciones (POS).

Inventory-Service: Gestión de stock en tiempo real.

Customer-Service: Datos de clientes y promociones.

Justificación :

Bajo acoplamiento: Cada servicio puede evolucionar y escalar independientemente.

Alta cohesión: Las funcionalidades se agrupan por dominio de negocio (ej: no mezclar lógica de ventas con inventario).

4.3. Api Gateway y Seguridad centralizada

Problema a resolver:

El monolito tiene autenticación redundante en cada módulo y dificulta el manejo de clientes móviles/pos físicos.

Solución:

API Gateway (Kong/Spring Cloud Gateway):

Enrutamiento único para todas las peticiones.

Autenticación centralizada con JWT + OAuth2.0.

Justificación :

Simplificación: Evita duplicar lógica de seguridad en cada servicio.

Protección: Istio (Service Mesh) añade encriptación mTLS entre servicios.

4.4. Bases de datos por servicio

Problema a resolver:

La base de datos única del monolito genera contention (ej: bloqueos al actualizar inventario y registrar ventas).

Solución:

Sales-Service: PostgreSQL (transacciones ACID).

Customer-Service: MongoDB (flexibilidad con datos no estructurados).

Justificación :

Performance: Cada servicio elige la DB que mejor se adapta a sus necesidades.

Aislamiento: Un fallo en inventario no afecta las ventas.

4.5. Implementación gradual

Problema a resolver:

No se puede reemplazar el monolito de golpe sin afectar operaciones.

Solución:

Migrar solo el POS a sales-service (módulo más crítico).

Luego, inventario (inventory-service) y CRM (customer-service).

Justificación técnica:

Menor riesgo: Validación progresiva sin paralizar el negocio.

Aprendizaje: El equipo se adapta gradualmente a la nueva arquitectura.

4.6. Automatización y DevOps

Problema a resolver:

El despliegue manual del monolito es lento y propenso a errores.

Solución:

Kubernetes: Orquestación de contenedores (escalado automático por sucursal).

Prometheus + Grafana: Monitoreo de métricas en tiempo real.

Jenkins/GitLab CI: Pipelines independientes por servicio.

Justificación técnica:

Despliegues rápidos: Actualizaciones sin downtime.

Auto-reparación: Kubernetes reinicia servicios fallidos.