Transformación digital de Perfulandia SPA

ÍNDICE

Parte 1: Introducción, definición del problema y análisis del sistema actual

El objetivo de este informe es analizar y proponer una solución basada en arquitectura de microservicios con MySQL, que permita a Perfulandia SPA mejorar su operación y soportar su continuo crecimiento.

Perfulandia SPA es una empresa chilena emergente, que se ha destacado por ofrecer productos de alta calidad a precios competitivos. Inicialmente, la empresa comenzó con una sucursal en el Barrio Meiggs en Santiago, pero su éxito en ventas, tanto al por menor como al por mayor, ha llevado a la apertura de nuevas sucursales en Concepción y Viña del Mar. La empresa ahora planea continuar su expansión debido a su crecimiento exponencial y el aumento de nuevos clientes a nivel nacional. Sin embargo, este **rápido crecimiento** ha revelado las limitaciones de su actual sistema de software monolítico. El sistema ha comenzado a fallar, presentando **problemas de rendimiento** y **disponibilidad** que ponen en riesgo las **operaciones diarias** y la **satisfacción del cliente**.

Actualmente, es probable que el sistema monolítico de Perfulandia SPA presente los siguientes problemas:

- **Dificultad para escalar**: Con la apertura de nuevas sucursales, la carga de trabajo ha aumentado y el sistema no responde eficientemente.
- **Baja disponibilidad**: Un fallo en un módulo puede afectar a toda la plataforma, generando interrupciones en las operaciones.
- **Cuellos de botella en la base de datos**: Un solo servidor de base de datos MySQL maneja toda la información del sistema. A medida que la empresa crece, las consultas se vuelven más lentas y hacen que el sistema no funcione como se le espera.
- **Errores que afectan todo el sistema:** Considerando todo lo anterior, al ser un sistema monolítico, si hay un error en un módulo, toda la aplicación puede fallar para todos los usuarios.
- **Tiempo de desarrollo y mantenimiento alto**: Cualquier cambio en el sistema afecta a todo el código, lo que hace que agregar nuevas funciones sea complicado y arriesgado.

Para solucionar estos problemas, se propone migrar a una arquitectura de **microservicios** que permita un desarrollo modular y escalable.

Para la implementación de los microservicios se utilizará:

- **Backend**: Flask (Python) o Spring Boot (Java), por su facilidad de uso y compatibilidad con microservicios.
- **Base de datos**: MySQL, por su estabilidad y soporte transaccional.
- **API Gateway**: Kong, para gestionar el enrutamiento de servicios.
- **Autenticación**: JWT (JSON Web Token), para seguridad de usuarios.
- **Contenedores**: Docker, para facilitar el despliegue.

Estas herramientas han sido seleccionadas porque cuentan con documentación accesible y permiten una implementación gradual.

Parte 2: Análisis de requerimientos

Requisitos funcionales

Administrador del sistema

- **Gestionar usuarios:** Crear, actualizar, desactivar y eliminar cuentas de usuarios del sistema según su rol.
- **Configurar permisos:** Asignar y modificar permisos de acceso según el rol de cada usuario en la empresa.
- **Monitorización del sistema:** Visualizar el estado del sistema en un panel de control y recibir alertas sobre fallos de rendimiento o disponibilidad.
- **Respaldar y restaurar datos:** Realizar copias de seguridad periódicas en servidores externos y restaurar datos en caso de pérdida o fallo.

Gerente de sucursal

- **Gestionar inventario:** Agregar, actualizar y eliminar productos del inventario, ajustando las cantidades de stock manualmente o mediante integración con un sistema automatizado.
- **Generar reportes:** Crear reportes de ventas, inventario y rendimiento, almacenados en la plataforma para consulta y análisis por la administración.
- **Gestionar sucursales:** Administrar detalles de cada 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.

Empleado de ventas

- **Registrar ventas:** Procesar transacciones de venta en el sistema, aplicando descuentos y ofertas automáticamente o de forma manual.
- **Atender devoluciones y reclamaciones:** Procesar devoluciones de productos y gestionar reclamaciones de clientes en el sistema.
- **Consultar inventario:** Verificar disponibilidad de productos en tiempo real, accediendo al sistema centralizado de inventario.
- **Generar facturas:** Emitir facturas electrónicas y enviarlas automáticamente por correo electrónico a los clientes.

Logística

- **Gestionar envíos:** Crear, actualizar y rastrear envíos de productos desde la bodega hasta las sucursales y clientes.
- **Optimizar rutas de entrega:** Utilizar un sistema de planificación para optimizar rutas de entrega de pedidos locales y regionales.
- **Actualizar estado de pedidos:** Modificar el estado de los pedidos en el sistema desde el procesamiento hasta la entrega final.
- **Gestionar :** Mantener y actualizar la información de proveedores, realizar pedidos de reabastecimiento y gestionar la recepción de mercancías.

Acciones de los clientes vía Web

- **Crear cuenta:** Registrarse en la plataforma web proporcionando nombre, dirección de correo electrónico y dirección de envío.
- **Iniciar sesión:** Acceder a la cuenta utilizando credenciales de usuario.
- **Navegar y buscar productos:** Explorar el catálogo de productos mediante filtros y barra de búsqueda.
- **Agregar productos al carrito:** Seleccionar productos y añadirlos al carrito de compras.
- **Realizar pedidos:** Completar la compra ingresando detalles de pago y seleccionando opciones de envío.
- **Consultar historial de pedidos:** Ver compras anteriores y el estado actual de 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 y calificaciones:** Evaluar productos comprados y dejar comentarios en el sitio web.
- **Aplicar cupones y descuentos:** Ingresar códigos promocionales durante la compra para obtener descuentos.

Requisitos no funcionales

- **Escalabilidad:** El sistema debe soportar el crecimiento de la empresa sin pérdida de rendimiento.
- **Seguridad:** Implementación de cifrado en datos sensibles y autenticación segura.
- **Disponibilidad:** Garantizar un 99.9% de tiempo activo mediante servidores redundantes.
- **Usabilidad:** Interfaz intuitiva para facilitar la gestión interna y la experiencia del cliente.
- **Compatibilidad:** Acceso desde dispositivos móviles y navegadores web modernos.
- **Mantenimiento:** Facilitar actualizaciones y correcciones sin afectar la operatividad.

Estos requisitos aseguran que el sistema cubrirá todas las necesidades operativas de Perfulandia SPA.

Entrevistas a diferentes perfiles de usuario

1. Entrevista al Administrador del Sistema
Contexto: Reunión virtual vía Teams (15 min).
Participante: Juan Pérez, Administrador TI (3 años en Perfulandia).

Preguntas y Respuestas:

- P: "¿Cuál es el principal dolor de cabeza al gestionar usuarios?"
- R: -Cuando hay que dar permisos nuevos, debo modificar código en el monolito. Pierdo 2 horas por cada cambio y a veces rompe otros módulos. "
- P: "¿Cómo manejas las caídas del sistema?"
- R: -Nos enteramos cuando los usuarios llaman indignados. No hay alertas automáticos. Necesitamos un dashboard con métricas en vivo!"

Hallazgos Clave:
• Requiere gestión de permisos sin tocar código.
• Urge monitorización proactiva (Ej.: Grafana).

2. Entrevista al Gerente de Sucursal (Viña del Mar)
Contexto: Visita presencial a sucursal (20 min).
Participante: María González, Gerente Regional.
Preguntas y Respuestas:

- **P:** -¿Cómo decides reabastecer inventario?
- **R:** 'Revisar reportes impresos„. que están desactualizados! Ayer vendimos 30 unidades de un producto que el sistema decía tener 'en stock', pero no había.
- **P:** -¿Qué reporte te gustaría tener?
- **R:** -Uno que combine ventas por hora y stock actual, con alertas si algo se agota.

Hallazgos Clave:

- El inventario en tiempo real es crítico.
- Reportes deben ser automáticos y personalizables.

3. Entrevista a Empleado de Ventas (Santiago)
Contexto: Llamada telefónica (10 min).
Participante: Carlos Muñoz, Vendedor senior**.**
Preguntas y Respuestas
• **P:** Describe el proceso de una venta con el sistema actual
• **R:** Si el cliente pide 3 productos, debo buscarlos uno por uno.
A veces el sistema se traba y pierdo la venta.
• **P:** ¿Qué mejorarías?
• **R:** Un buscador rápido que filtre por categoría y muestre fotos del producto. Y que no se caiga en horario punta.

Hallazgos Clave:

- Buscador de productos optimizado (con imágenes y filtros).
- Sistema debe soportar alta concurrencia.

Parte 3: Diseño de la nueva arquitectura

Estrategia de microservicios

Se propone dividir el sistema en varios microservicios independientes, cada uno con una funcionalidad específica:

Usuarios: Registro, autenticación y gestión de clientes.
Inventario: Control de productos y stock.
Ventas: Procesamiento de transacciones y facturación.
Logística: Gestión de envíos y seguimiento de pedidos.

Estos microservicios se comunicarán vía API REST inicialmente, con la posibilidad de implementar colas de mensajes en el futuro.
Se elige MySQL como motor de base de datos por su:
Soporte a transacciones ACID, asegurando la consistencia de los datos.
Alto rendimiento en consultas transaccionales.
Amplia documentación y soporte, facilitando su uso por parte del equipo de estudiantes.

Diagramas

1. Diagrama de contexto

Propósito: Mostrar el sistema como una "caja negra" y su interacción con actores externos.

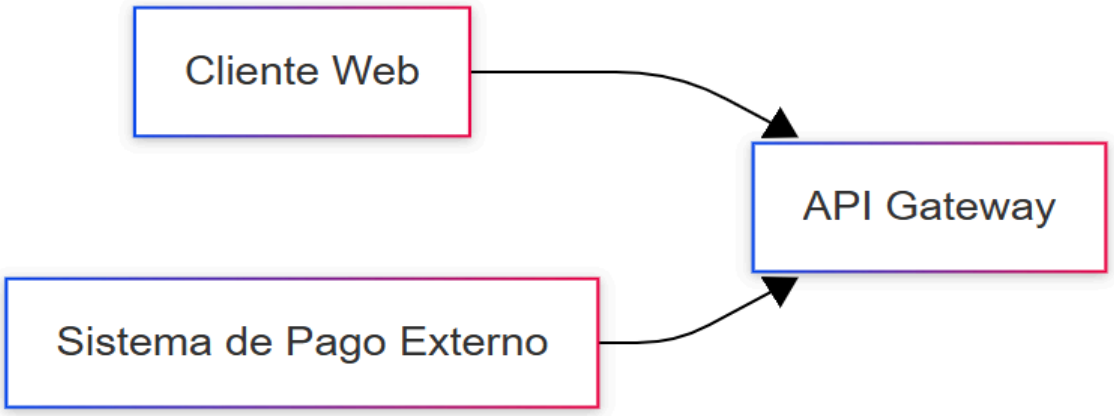
Qué incluir.

• Actores principales: Clientes web, empleados, proveedores.

• Sistemas externos: Pasarelas de pago, servicios de logística.

• Flujos clave: "Cliente realiza pedido", "Proveedor recibe orden".

Ejemplo visual:



2. Diagrama de Componentes

Propósito: Detallar los microservicios y sus relaciones.

Qué incluir:

• Cada microservicio como un componente (ej: Inventario, Ventas).

• Protocolos de comunicación (REST, Kafka, gRPC).

• Bases de datos asociadas a cada servicio

Ejemplo:

3. Diagrama de Despliegue

Propósito: Mostrar la infraestructura en la nube y contenedores.

Qué incluir.

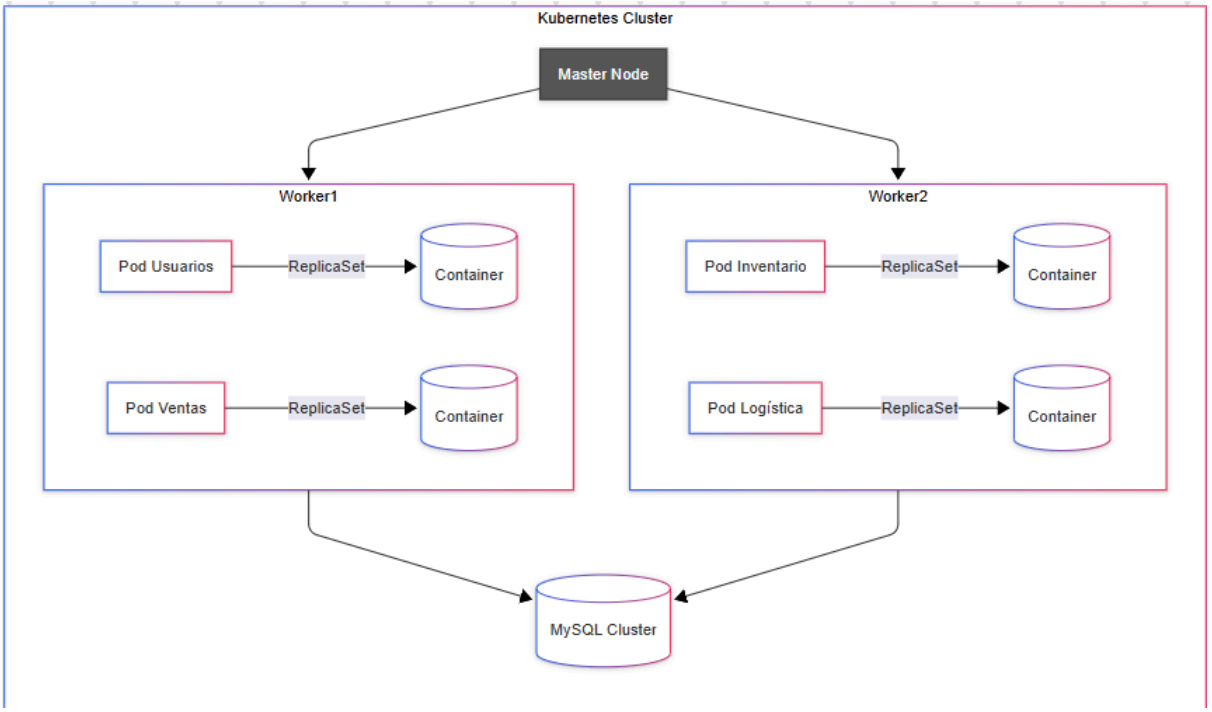
• Nodos Kubernetes (pods, deployments).

• Servicios en la nube (AWS RDS EC2).

• Balanceadores de carga y API Gateway.

Herramienta: Lucidchart (iconos de AWS/Azure).

Ejemplo:



4. Diagrama de Secuencia

Propósito: Explicar flujos específicos (ej: proceso de venta).

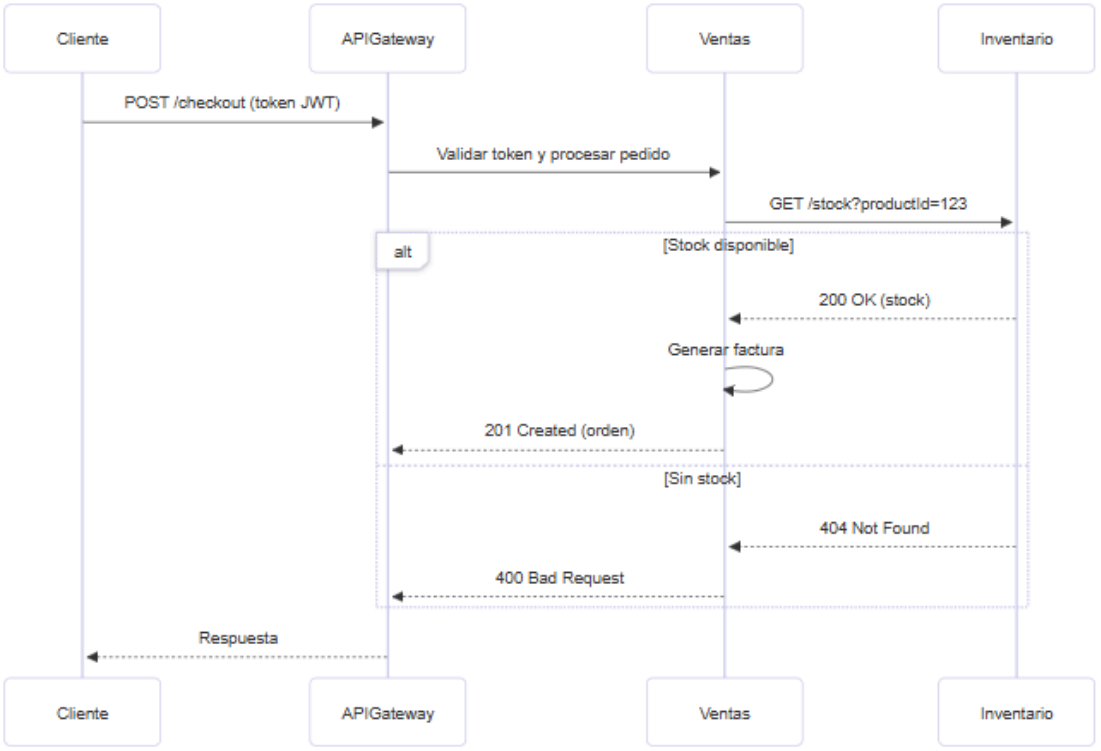
Qué incluir:

• Pasos cronológicos entre componentes.

• Mensajes/eventos (ej: "GET /stock", "Pedido Creado").

• Tiempos de respuesta (opcional).

Ejemplo:



5. Diagrama de Casos de Uso

Propósito: Relacionar actores con funcionalidades.

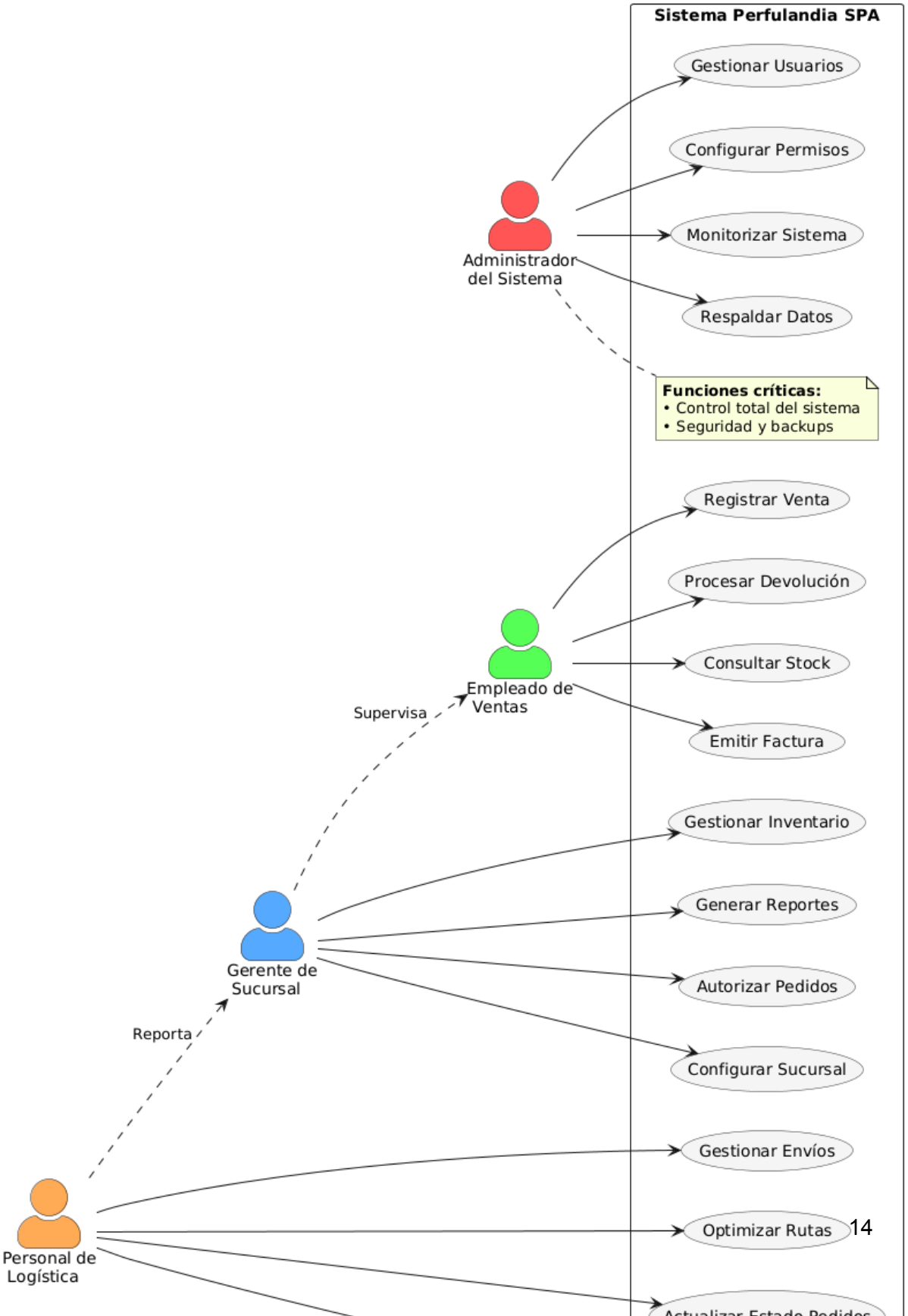
Qué incluir:

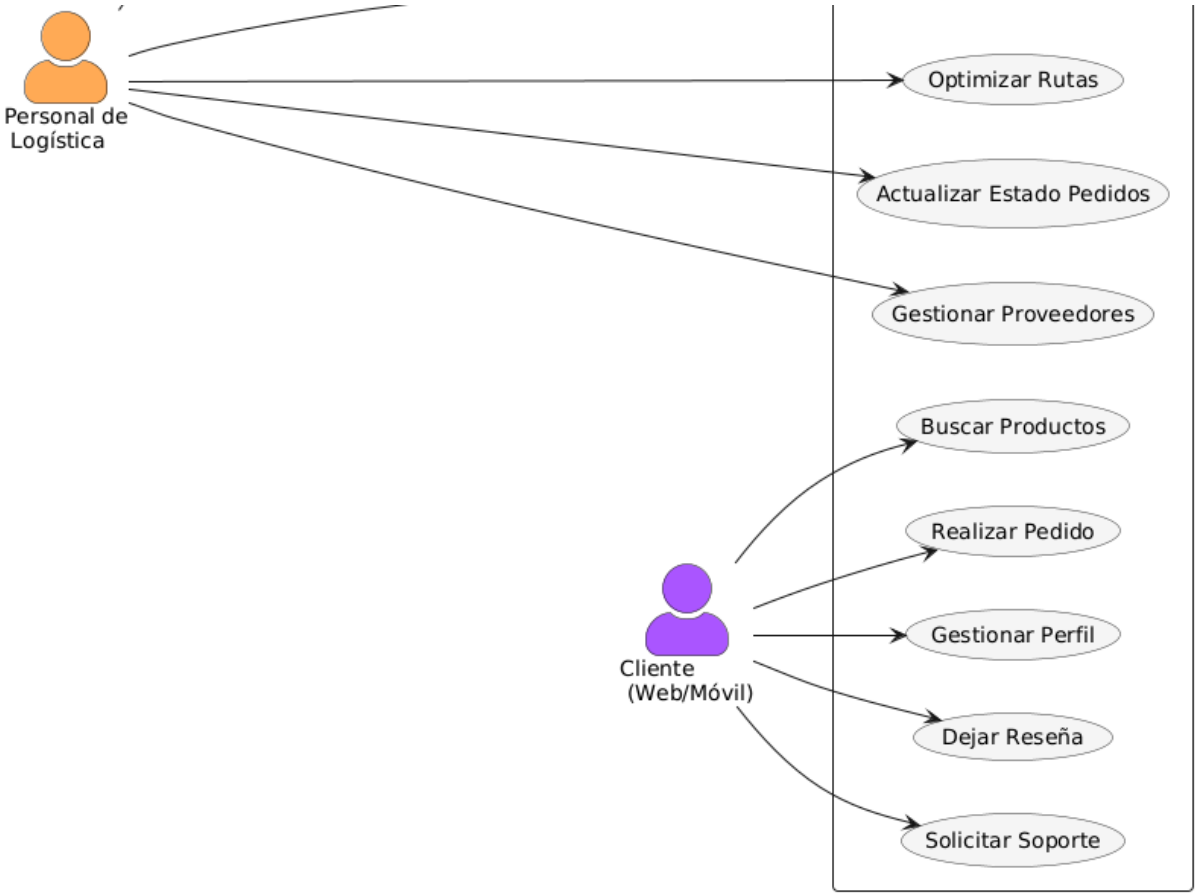
• Actores (Gerente, Cliente, etc.).

• Casos de uso (Gestionar Inventario, Realizar Pedido).

• Relaciones o <<extend>>.

Ejemplo:




Personal de Logística
Optimizar Rutas
Actualizar Estado Pedidos
Gestionar Proveedores
Buscar Productos
Realizar Pedido
Gestionar Perfil
Cliente (Web/Móvil)
Dejar Reseña
Solicitar Soporte

Paso 4: Planificación de la migración

Para migrar de manera efectiva el sistema monolítico de Perfulandia SPA a una arquitectura de microservicios, es necesario seguir un enfoque estructurado y minimizar riesgos. La estrategia de migración se basa en la metodología Strangler Fig Pattern, que permite migrar el sistema en fases sin afectar la operatividad del negocio.

Fases del plan de migración

Fase 1: Análisis y preparación

- Evaluación del sistema actual: Identificar los módulos del sistema monolítico, sus dependencias y cuellos de botella.
- Definición de los microservicios: Determinar qué componentes deben separarse primero basándose en la criticidad y frecuencia de uso.
- Elección de tecnologías: Confirmar herramientas como Spring Boot (Java), Node.js, MySQL, RabbitMQ/Kafka, Docker y Kubernetes.
- Planificación de la migración: Crear un roadmap detallado con hitos y asignación de recursos.

Fase 2: Implementación del API Gateway y

- Desplegar un API Gateway: Unificar el acceso a los microservicios y manejar la autenticación centralizada con OAuth 2.0 y JWT.
- Migrar el sistema de autenticación: Crear un microservicio independiente para el manejo de usuarios y roles.
- Pruebas de compatibilidad: Asegurar que el API Gateway enruta correctamente las solicitudes al sistema monolítico y al nuevo servicio de autenticación.

Fase 3: Migración del Servicio de Inventario y Ventas

- Extraer la lógica de inventario del monolito y crear un microservicio dedicado.
- Implementar el microservicio de ventas para procesar transacciones sin afectar la operación.
- Habilitar comunicación entre los nuevos servicios y el sistema antiguo a través del API Gateway.
- Monitorear el rendimiento y corregir errores antes de desactivar los módulos monolíticos.

Fase 4: Migración del Servicio de Pedidos y Logística

- Crear un microservicio de pedidos para gestionar órdenes en tiempo real.
- Implementar un microservicio de logística para coordinar envíos y optimizar rutas.
- Conectar estos microservicios con el de inventario y ventas mediante eventos asíncronos (RabbitMQ/Kafka).
- Realizar pruebas de carga y performance para garantizar una respuesta óptima.

Fase 5: Migración del Servicio de Reportes y Atención al Cliente

- Extraer la lógica de reportes y diseñar una solución que almacene métricas y tendencias en una base de datos optimizada.
- Crear el servicio de atención al cliente con gestión de tickets y chat en tiempo real.
- Integrar herramientas de monitoreo (Prometheus + Grafana) para medir el rendimiento de los nuevos servicios.

Fase 6: Desmantelamiento del monolito y optimización

- Desactivar módulos monolíticos reemplazados y migrar datos pendientes.
- Optimizar y escalar microservicios críticos según demanda.
- Capacitar a los usuarios y realizar pruebas finales antes del despliegue total.

Plan de mitigación de riesgos

Identificación de riesgos y plan de mitigación

Migrar un sistema monolítico a una arquitectura de microservicios implica diversos desafíos. A continuación, se identifican los principales riesgos asociados al proceso y se plantea un plan de mitigación para cada uno.

1. Riesgo: Pérdida de datos durante la migración:

La separación de la base de datos monolítica en múltiples bases específicas para cada microservicio puede generar inconsistencias o pérdida de información.

Plan de mitigación

- Realizar respaldos periódicos de la base de datos antes de cada fase de migración.
- Implementar migraciones progresivas con pruebas en entornos de staging antes del despliegue en producción.
- Usar herramientas de sincronización de datos como Debezium o AWS DMS.

2. Riesgo: Problemas de comunicación entre microservicios

La interacción entre microservicios puede generar latencias o fallos si no se configuran correctamente los mecanismos de comunicación.

Plan de mitigación:

- Usar un API Gateway para gestionar la comunicación y centralizar la autenticación.
- Implementar mensajería asíncrona con RabbitMQ o Kafka para minimizar acoplamientos.
- Establecer reintentos automáticos y circuit breakers (resiliencia) con herramientas como Hystrix o Resilience4j.

3. Riesgo: Sobrecarga y consumo excesivo de recursos

Descripción: Los microservicios mal diseñados pueden generar alta demanda en la infraestructura, causando lentitud o fallos en el sistema.

Plan de mitigación:

- Implementar escalado automático con Kubernetes para ajustar recursos dinámicamente.
- Monitorizar el rendimiento con herramientas como Prometheus y Grafana.
- Aplicar caching con Redis o Memcached para reducir consultas innecesarias a la base de datos.

4. Riesgo: Falta de coordinación en el equipo de desarrollo

La transición a microservicios requiere cambios en la metodología de trabajo y coordinación entre equipos.

Plan de mitigación:

- Adoptar metodologías ágiles como Scrum para gestionar el desarrollo por iteraciones.
- Usar herramientas de documentación como Swagger para estandarizar las APIs.
- Realizar sesiones de formación sobre arquitectura de microservicios y buenas prácticas.

5. Riesgo: Errores en la seguridad y autenticación

La transición de un sistema monolítico a múltiples servicios puede generar vulnerabilidades de seguridad si no se manejan correctamente los accesos y datos sensibles.

Plan de mitigación:

- Implementar autenticación centralizada con OAuth 2.0 y JWT.
- Aplicar cifrado de datos sensibles con TLS/SSL y limitar accesos mediante políticas de permisos estrictas.
- Realizar auditorías de seguridad y pruebas de penetración antes del despliegue final.

6. Riesgo: Dependencia del sistema monolítico durante la transición

La coexistencia del sistema antiguo con los nuevos microservicios puede generar inconsistencias o cuellos de botella en la migración.

Plan de mitigación:

- Utilizar la estrategia Strangler Fig Pattern para una transición gradual.
- Diseñar los nuevos microservicios de forma independiente, asegurando que cada módulo funcione sin depender del monolito.
- Usar Feature Flags para habilitar progresivamente los nuevos servicios y desactivar los antiguos sin interrupciones.