

Informe

Caso de Estudio: Transformación Digital de "Perfulandia SPA"

**Grupo 1**

Gonzalo Navarrete

Carla Prado

Fernando Zárate

Índice

[**0. Introducción 3**](#_5j1tahb0cdmb)

[**1. Análisis de Requerimientos 3**](#_a8ukwp8yjrsi)

[1.1. Requisitos Funcionales 3](#_inr8434vac9q)

[1.2. Requisitos No Funcionales 4](#_ft5kmshw0r)

[1.3. Entrevistas con Perfiles de Usuarios (Método Épicas de usuario) 5](#_wru5a85ykgh9)

[**2. Análisis del Sistema Actual 6**](#_ij8jaderhzrs)

[2.1. Descripción del Sistema Monolítico Existente 6](#_zcktlktd1wh1)

[2.2. Principales Causas de Fallos y Sobrecargas 6](#_3npwkzak33c)

[2.3. Consecuencias 7](#_3wx290ld7b8g)

[**3. Diseño de la Nueva Arquitectura 8**](#_2oafr55bi3rh)

[3.1 Propuesta de arquitectura nueva basada en Microservicios 8](#_6ycbqcq63hm1)

[3.2. Ventajas de la propuesta basada en microservicios 10](#_lul1f64owuik)

[3.3. Comparación con otras arquitecturas 10](#_6yz88mshqk45)

[3.4. Herramientas y tecnología a utilizar 11](#_mgewice6fcqg)

[3.5. Consideraciones Éticas 13](#_7yn9l4tyewcs)

[3.6. Diagramas de Casos de Uso 16](#_syoky7ahrkeo)

[3.7. Diagrama de clases 18](#_uwy27n23v15v)

[3.8. Diagrama de despliegue 19](#_hrkgalhmc4am)

[**4. Planificación de la Migración 19**](#_aayriwbp6r27)

[4.1. Análisis de metodologías 19](#_8a85bptsc7fb)

[4.2 Propuesta de Migración ágil 20](#_lzio3blq5ett)

[4.3. Matriz de riesgos 27](#_zfkqt0d02kmg)

[**6. Conclusión 27**](#_orew81og4715)

# 0. Introducción

Perfulandia SPA es una empresa chilena en expansión, reconocida por su oferta de productos de calidad a precios accesibles. Sin embargo, su rápido crecimiento ha evidenciado las limitaciones de su actual sistema monolítico, afectando su rendimiento y disponibilidad. Para sostener su escalabilidad y mejorar la eficiencia operativa, se ha propuesto la adopción de una arquitectura basada en microservicios, utilizando MySQL como motor de base de datos. Este estudio tiene como objetivo analizar, diseñar e implementar una solución tecnológica que permita a la empresa optimizar su infraestructura y garantizar una experiencia fluida para sus clientes y empleados.

# 1. Análisis de Requerimientos

## 1.1. Requisitos Funcionales

1. **Gestión de personas y sucursales (User persona: Administrador del Sistema)**

Funciones asociadas:

* Crear cuentas nuevas o registros para empleados y asignarles un identificador único, nombre, correo electrónico y rol dentro del sistema (ej. gerente, vendedor, logística).
* Actualizar información existente, como cambios de rol o datos personales, garantizando que los registros sean consistentes.
* Desactivar cuentas temporalmente en caso de ausencias prolongadas o bajas, manteniendo un historial de actividad.
* Eliminar cuentas de forma permanente cuando un empleado deja la empresa.
* Crear nuevas sucursales y asignarles un identificador único, nombre, dirección y gerente.
* Actualizar información de sucursales, como cambios de gerente o ubicación.
* Eliminar sucursales (solo si no están activas o tienen operaciones pendientes).
* Asociar empleados a una o varias sucursales.
* Sistema de respaldo o restauración de datos.
* Asignar permisos específicos a cada usuario según su rol (RBAC), como acceso a módulos de inventario, ventas o reportes.
* Modificar permisos en tiempo real (por ejemplo, otorgar acceso temporal a un módulo durante una tarea específica).
* Revocar permisos de forma inmediata en caso de cambios de responsabilidad o incidentes de seguridad.

1. **Gestión de inventario (User persona: Gerente de Sucursal**)

Funciones asociadas:

* Sistema de supervisión de pedidos con generación de reportes.
* Configuración de sucursales
* Asignar permisos específicos a cada usuario según su rol, como acceso a módulos de inventario, ventas o reportes, utilizando un sistema de control de acceso basado en roles (RBAC).
* Modificar permisos en tiempo real, por ejemplo, otorgar acceso temporal a un módulo durante una tarea específica.
* Revocar permisos de forma inmediata en caso de cambios de responsabilidad o incidentes de seguridad.

1. **Registro y administración de ventas (User persona: Empleado de Ventas)**

Funciones asociadas:

* Guardar ventas en el sistema con datos como producto, cantidad y precio.
* Manejar devoluciones, actualizando el stock y dando reembolsos si toca.
* Consultar el inventario para ver qué productos hay disponibles.
* Emitir facturas después de vender, con opción de enviarlas por correo.
* Actualizar si una venta está hecha o cancelada, conectándolo al inventario.

1. **Gestión de envíos (User persona: Logística)**

Funciones asociadas:

* Organizar envíos, asignando pedidos a quien los lleva y siguiendo su avance.
* Planear rutas cortas y rápidas para entregar pedidos.
* Cambiar el estado de los pedidos (ej. enviado, entregado) y avisar a todos.
* Llevar control de proveedores, como quién trae qué y cuándo.
* Hacer reportes básicos de entregas, como tiempos o problemas.

1. **UX (Clientes vía web)**

Funciones asociadas:

* Registrarse con correo y contraseña, y entrar al sistema.
* Buscar productos fácilmente, con filtros como precio o tipo.
* Armar un carrito, agregar o quitar cosas antes de comprar.
* Hacer pedidos y pagar con tarjeta o en efectivo al recibir.
* Ver compras pasadas y descargar facturas.
* Pedir ayuda por chat o mensaje si algo sale mal.
* Dejar opiniones de productos después de comprar.
* Usar códigos de descuento al pagar.

## 1.2. Requisitos No Funcionales

1. **Escalabilidad**: El sistema debe soportar el crecimiento exponencial de usuarios y transacciones.

* El sistema debe funcionar bien aunque haya más usuarios o ventas.
* Aguantar días ocupados sin ponerse lento.

1. **Disponibilidad**: Garantizar un tiempo de actividad del 99.9% para evitar interrupciones en las operaciones.

* Hacer arreglos cuando pocos lo usan, avisando antes.

1. **Rendimiento**: Respuesta rápida (<2 segundos) en consultas de inventario y procesamiento de pedidos.

* Responder rápido (<2 segundos) al buscar stock o hacer pedidos.
* Manejar muchas ventas al mismo tiempo sin trabarse.

1. **Seguridad**: Protección de datos sensibles (clientes, transacciones) mediante cifrado y autenticación robusta.

* Proteger datos importantes (como nombres o pagos) con claves seguras.
* Pedir contraseñas fuertes y revisar quién entra al sistema.

1. **Mantenibilidad**: Facilitar actualizaciones y correcciones sin afectar el sistema completo.

* Hacer cambios o arreglos sin romper todo.
* Tener guías claras para que otros puedan trabajar en el sistema.

## 1.3. Entrevistas con Perfiles de Usuarios (Método Épicas de usuario)

1. **Administrador del Sistema**

* "Como administrador del sistema, quiero un panel de monitoreo proactivo que me permita supervisar el estado de todo el sistema en tiempo real, para prevenir fallos y reaccionar rápidamente ante incidencias."
* "Como administrador del sistema, quiero un sistema de auditoría automatizado que registre todas las acciones de usuarios y cambios en el sistema, para garantizar trazabilidad y cumplir con normativas de seguridad."

1. **Gerente de Sucursal**

* "Como gerente de sucursal, quiero un módulo de análisis predictivo que me ayude a anticipar la demanda de productos y optimizar el inventario, para evitar faltantes o excesos en stock."
* "Como gerente de sucursal, quiero configurar alertas personalizadas sobre niveles bajos de inventario o retrasos en pedidos, para actuar antes de que afecten las ventas."

1. **Empleado de Ventas**

* "Como empleado de ventas, quiero un sistema que me permita procesar ventas offline y sincronizarlas automáticamente cuando vuelva la conexión, para no perder transacciones en caso de fallos de red."

1. **Logística**

* "Como responsable de logística, quiero un sistema que optimice automáticamente las rutas de entrega basándose en tráfico y distancia, para reducir tiempos y costos de envío."
* "Como responsable de logística, quiero un tablero de seguimiento en tiempo real que muestre el estado de todos los envíos activos, para coordinar mejor con transportistas y clientes."

1. **Cliente**

* "Como cliente, quiero una experiencia de compra amigable, personalizable con recomendaciones basadas en mis compras anteriores, para encontrar productos que me interesen más rápido."
* "Como cliente, quiero cancelar o modificar mi pedido fácilmente antes de que sea enviado, para tener más control sobre mis compras sin depender de soporte."

# 2. Análisis del Sistema Actual

## 2.1. Descripción del Sistema Monolítico Existente

El sistema actual de Perfulandia SPA es una aplicación monolítica que integra todas las funcionalidades (ventas, inventario, logística, etc.) en un solo código base. Además, parece ser que funciona en un servidor centralizado que soporta las tres sucursales y la plataforma web.

## 2.2. Principales Causas de Fallos y Sobrecargas

1. **Acoplamiento Fuerte/Falta de Tolerancia a Fallos (Single Point of Failure)/Falta de modularidad**: Un fallo en un módulo o base de datos afecta a todo el sistema.
2. **Dependencias Internas Complejas:** Los módulos (ventas, inventario, logística) están tan interconectados que un cambio o error en una librería o dependencia compartida puede generar fallos en cadena, dificultando la identificación y resolución del problema.
3. **Escalabilidad Limitada/Concurrencia no controlada**: El servidor único no puede manejar el aumento de transacciones en horas peak y aumentos de demanda.
4. **Base de Datos Centralizada**: Consultas simultáneas de sucursales y clientes generan cuellos de botella.
5. **Gestión Ineficiente de Recursos:** El sistema monolítico no permite asignar recursos (CPU, memoria, etc.) de forma independiente a cada módulo. Por ejemplo, un pico en las consultas de inventario podría consumir todos los recursos disponibles, dejando sin capacidad a las ventas o la logística.
6. **Tiempo de Respuesta Lento:** al procesar todas las solicitudes en un solo flujo, puede sufrir retrasos significativos cuando hay un volumen alto de usuarios.
7. **Pruebas y Depuración Complejas:** Al estar todo en un solo código base, probar una funcionalidad específica requiere probar el sistema completo, lo que aumenta el riesgo de introducir errores no detectados y retrasa la corrección de fallos.
8. **Sobrecarga por Procesos Secundarios:** Tareas como generación de reportes, backups de la base de datos o sincronización de inventario compiten por los mismos recursos que las operaciones principales, lo que puede degradar el rendimiento en momentos críticos.
9. **Incapacidad para Implementar Actualizaciones Parciales.**
10. **Limitaciones o brechas en la Integración con Nuevas Tecnologías.**

## 2.3. Consecuencias

1. Retrasos en el procesamiento de ventas y pedidos.
2. Competencia por recursos entre usuarios concurrentes, aumentando el riesgo de bloqueos o datos corruptos.
3. Pérdida de clientes por Interrupciones en la plataforma Web, y consecuente pérdida de dinero.
4. Dificultad para expandirse a nuevas sucursales sin comprometer el rendimiento.
5. Costos operativos ocultos y pérdida de productividad por tiempos de inactividad por actualizaciones o fallos, lo que obliga a los empleados a recurrir a procesos manuales o en papel, causando inconsistencias en el inventario. Error humano amplificado.
6. Riesgo de obsolescencia tecnológica, afectando ventaja competitiva.
7. Insatisfacción de los empleados.
8. Pérdida de datos potencial, afectando la integridad de la información crítica.

# 3. Diseño de la Nueva Arquitectura

## 3.1 Propuesta de arquitectura nueva basada en Microservicios

1. **Core (Autenticación, Autorización de usuarios, gestión de sucursales)**
   * Tipo: Microservicio de seguridad y control de acceso.
   * Funcionalidades:
     + Gestión de autenticación de usuarios.
     + Control de acceso basado en roles (RBAC).
     + Configuraciones generales del sistema.

* Ejemplo**:** Netflix utiliza un microservicio de seguridad basado en su sistema "Zuul" (un API Gateway) combinado con servicios de autenticación como parte de su plataforma en la nube AWS.

1. **Comercio (Inventario y Productos)**
   * Tipo: Microservicio de gestión de productos.
   * Funcionalidades:
     + Manejo de catálogo de productos.
     + Control de inventario y stock en tiempo real.
     + Integración con ventas para actualizar inventario.

* Ejemplo:Amazon migró su sistema monolítico a microservicios en la década de 2000, y uno de sus servicios clave es la gestión de inventario y productos (desarrollado por ellos mismos). Multi-Channel Fulfillment (MCF).

1. **Pedidos (Ventas y Facturación)**
   * Tipo: Microservicio transaccional.
   * Funcionalidades:
     + Registro de ventas y pedidos.
     + Manejo de facturas y pagos.
     + Historial de compras de clientes.

* Ejemplo:Uber utiliza un microservicio transaccional para gestionar pedidos (viajes) y facturación. Está integrado dentro de lo que Uber llama su "Trip Execution Engine". Este sistema fue desarrollado internamente por los equipos de ingeniería de Uber.

1. **Logística (Gestión de Envíos)**
   * Tipo: Microservicio de operación y despacho.
   * Funcionalidades:
     + Gestión y seguimiento de envíos.
     + Planificación de rutas de entrega.
     + Coordinación con transportistas.

* Ejemplo:FedEx ha adoptado microservicios para optimizar su cadena logística y seguimiento de envíos, originalmente basada en mainframes. Forma parte de su plataforma "FedEx Ship Manager" y fue desarrollado por su propio equipo de IT.

1. **Cliente (Gestión de clientes, Interfaz y Experiencia de Usuario, autenticación separada)**
   * Tipo: Microservicio de interacción con usuarios.
   * Funcionalidades:
     + Manejo de cuentas de clientes.
     + Registro y autenticación de usuarios.
     + Opciones de compra y revisiones de productos.
   * Ejemplo: Integrado en lo que Spotify llama su "Client Services", hay un microservicio que gestiona las aplicaciones móviles, de escritorio y web. Fue desarrollado por sus equipos de ingeniería en Suecia y EE.UU.

## 3.2. Ventajas de la propuesta basada en microservicios

1. Reducción del acoplamiento fuerte.
2. Escalabilidad mejorada.
3. Eliminación de cuellos de botella en la base de datos.
4. Despliegue continuo y sin downtime.
5. Tolerancia a fallos mejorada.
6. Gestión más eficiente de recursos.
7. Menor complejidad de desarrollo, actualizaciones y mantenimiento comparado a sobreescribir la arquitectura monolítica existente.
8. Comunicación simplificada: menos problemas de coordinación y sincronización entre servicios.
9. Menor sobrecarga en infraestructura: uso eficiente de servidores y contenedores.
10. Procesamiento más rápido en horas peak.
11. Reducción de costos operativos ocultos.
12. Mitigación del riesgo de obsolescencia.
13. Migración Sin Interrupciones.

## 3.3. Comparación con otras arquitecturas

**SOA (Arquitectura Orientada a Servicios)**: SOA, aunque modular, suele depender de un bus de servicios centralizado (ESB), lo que crea un punto único de fallo y complejidad adicional. Para Perfulandia, esto sería excesivo frente a la simplicidad de las APIs REST de microservicios.

**Serverless**: Aunque serverless ofrece escalabilidad automática, no es ideal para procesos complejos y stateful como la gestión de inventario o envíos, que requieren consistencia y control local. Además, los costos pueden dispararse con el volumen de transacciones que Perfulandia espera.

**Enfoque híbrido monolítico-microservicios**: Mantener partes del monolito mientras se migran otras podría ser una transición temporal, pero genera deuda técnica y dificulta la coherencia a largo plazo, algo que Perfulandia necesita evitar para soportar su expansión.

## 3.4. Herramientas y tecnología a utilizar

1. **Spring Boot**: Framework para desarrollar microservicios en Java, con soporte para REST y MySQL.
   * **Funcionalidad**: Simplifica la creación de servicios independientes y su integración con MySQL.
   * **Contribución**: Agiliza el desarrollo backend y asegura escalabilidad.
   * **Comparación**: Simplifica el desarrollo de microservicios con configuraciones automáticas y soporte robusto. Comparado con Node.js (más ligero pero menos estructurado) o Django (Python, ideal para prototipos pero menos escalable en Java-ecosistemas).
2. **MySQL**: Motor de base de datos relacional.
   * **Funcionalidad**: Almacena datos estructurados (usuarios, inventario, pedidos).
   * **Contribución**: Soporta transacciones ACID y alta concurrencia.
   * **Comparación**: Base de datos relacional confiable, de código abierto, lo que reduce la curva de aprendizaje. Frente a PostgreSQL (más avanzado en funciones pero más complejo) o MongoDB (no relacional, menos ideal para datos estructurados como inventario).
3. **Canva/Drive/Docs**: Herramienta de trabajo colectivo.
   * **Funcionalidad**: Organiza diagramas (análisis, diseño, implementación).
   * **Contribución**: Mejora la planificación y comunicación del equipo.
4. **Contenedores, Orquestación, Mensajería. (Ej Docker, Kubernetes, RabbitMQ)**
   * **Contenedores**: Permiten empaquetar aplicaciones junto con sus dependencias, garantizando que se ejecuten de manera consistente en cualquier entorno.
   * **Orquestación**: Manejan la distribución de carga, reinicio automático de servicios fallidos y balanceo de tráfico. Permite escalar servicios según la demanda.
   * **Mensajería**: Facilita la comunicación asíncrona entre microservicios mediante mensajes encolados. Mejora la resiliencia y disponibilidad del sistema.
   * **Comparaciones:**

* Docker: Garantiza ejecución consistente al empaquetar aplicaciones y dependencias. Alternativas como Podman (sin daemon, más ligero) o LXC (menos popular) no tienen la adopción ni el ecosistema de Docker, clave para despliegues rápidos y portátiles.
* Kubernetes: Líder en orquestación, permite escalar servicios automáticamente según demanda. Comparado con Docker Swarm (más simple pero menos potente) o Nomad (flexible pero con menos soporte), Kubernetes ofrece escalabilidad, resiliencia y herramientas avanzadas para microservicios complejos.
* RabbitMQ: Sistema de mensajería fiable para comunicación asíncrona entre microservicios. Frente a Kafka (mejor para flujos masivos de datos pero más complejo) o Redis (rápido pero limitado a colas básicas), RabbitMQ es fácil de implementar y adecuado para las necesidades de logística y pedidos de Perfulandia.

1. **Frontend (React.js)**

* **Funcionalidad**: Proporciona un framework basado en componentes para construir interfaces de usuario interactivas y dinámicas, integrando HTML para la estructura y CSS para un diseño adaptable y visualmente coherente con la identidad de Perfulandia.
* **Contribución**: Permite desarrollar una plataforma web rápida y modular que mejora la interacción de los clientes con los microservicios (como búsquedas de productos, gestión de pedidos y seguimiento de envíos), facilitando actualizaciones frecuentes sin interrumpir el backend.
* **Comparación**: React.js: Framework para interfaces de usuario dinámicas y rápidas, ideal para la experiencia web de clientes de Perfulandia (búsquedas, carritos, pedidos). Su enfoque basado en componentes y su amplio ecosistema facilitan el desarrollo y mantenimiento. Comparado con Angular (más completo pero con mayor curva de aprendizaje y overhead) o Vue.js (ligero y simple, pero con menos adopción empresarial).

1. **Seguimiento (Grafana y Prometheus):** Monitoreo en tiempo real para evaluar rendimiento y detectar anomalías.
   * **Funcionalidad:** Prometheus recolecta métricas de rendimiento (latencia, uso de CPU, errores) de los microservicios, mientras que Grafana las visualiza en dashboards interactivos.
   * **Contribución:** Permite monitoreo proactivo para evitar downtime, optimizar recursos y garantizar la disponibilidad del sistema, alineándose con las necesidades de escalabilidad y resiliencia de Perfulandia.
   * **Comparación:** Prometheus es ideal para recolectar métricas en tiempo real con alta precisión y escalabilidad, superando a herramientas como Nagios (más tradicional, menos flexible para microservicios) o Zabbix (robusto pero con mayor complejidad de configuración). Grafana, por su parte, destaca en visualización frente a Kibana (enfocado en logs más que métricas) o Datadog (potente pero costoso y propietario), ofreciendo una solución de código abierto y personalizable que se integra perfectamente con Prometheus.

### **Control de versiones (GitHub)**

* **Funcionalidad:** Plataforma de control de versiones basada en Git, que permite alojar, revisar y colaborar en código fuente.
* **Contribución:** Facilita el trabajo colaborativo, la integración continua (CI), la trazabilidad de cambios y la gestión de ramas para el desarrollo paralelo de microservicios.

## 3.5. Consideraciones Éticas

1. **Privacidad y Seguridad de Datos**

* **Cifrado y Protección:**
  + Implementar cifrado AES-256 para datos sensibles (como información de clientes, transacciones y detalles de inventario) tanto en tránsito (usando TLS) como en reposo.
  + Proteger la privacidad de los clientes es una obligación moral y legal, especialmente en un contexto donde las filtraciones de datos pueden dañar la confianza y causar perjuicios económicos o personales.
  + Cumplimiento Normativo: Asegurar conformidad con regulaciones como la LGPD (Ley General de Protección de Datos) en Brasil, GDPR en Europa, o equivalentes locales, dependiendo del mercado de Perfulandia. Esto incluye obtener consentimiento explícito para el uso de datos y permitir a los usuarios acceder, modificar o eliminar su información.
  + Riesgo: Si un microservicio (como "Cliente") falla en su seguridad, podría exponer datos sensibles. Por ello, cada servicio debe tener auditorías de seguridad independientes.
* **Minimización de Datos:**
  + Recolectar solo la información estrictamente necesaria para cada microservicio (por ejemplo, "Pedidos" no necesita datos personales más allá de lo esencial para facturación).
  + Reducir el riesgo de mal uso o sobreexposición de datos personales, respetando el principio de "menos es más" en la gestión de información.

1. **Transparencia y Responsabilidad en el Despliegue**

* **Monitoreo Proactivo:**
  + Se recomienda usar herramientas como para monitorear el rendimiento de los microservicios en tiempo real, detectando fallos antes de que afecten a los usuarios.
  + Evitar interrupciones inesperadas que perjudiquen a clientes (retrasos en pedidos) o empleados (incapacidad de procesar ventas).
* **Comunicación Clara:**
  + Informar a los usuarios sobre actualizaciones o mantenimientos programados mediante el microservicio de "Notificaciones".
  + La transparencia fomenta la confianza y evita frustraciones, alineándose con el deber de mantener a los stakeholders informados.

1. **Impacto en el Empleo y Bienestar del Personal**

* **Capacitación del Personal:**
  + Ofrecer formación en el uso del nuevo sistema (por ejemplo, cómo gestionar inventario o pedidos a través de las nuevas interfaces) antes y durante la transición.
  + Evitar despidos o marginación de empleados que no dominen la tecnología, promoviendo la inclusión y el desarrollo profesional. Esto es crucial en un contexto donde la automatización podría percibirse como una amenaza.
  + Ejemplo: En lugar de reemplazar a los empleados que usaban procesos manuales, capacitarlos para operar el microservicio de "Logística" mejora su empleabilidad.
* **Reducción de Estrés Operativo:**
  + Al eliminar procesos manuales durante fallos (como ajustes de inventario en papel), los empleados enfrentan menos presión y frustración.
  + Mejorar las condiciones laborales es una responsabilidad ética que también aumenta la productividad y la satisfacción del equipo.

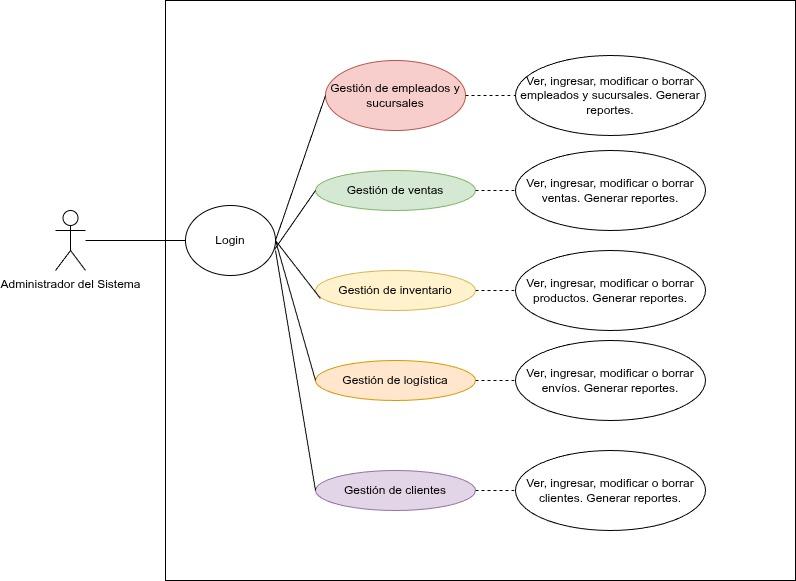
1. **Accesibilidad**

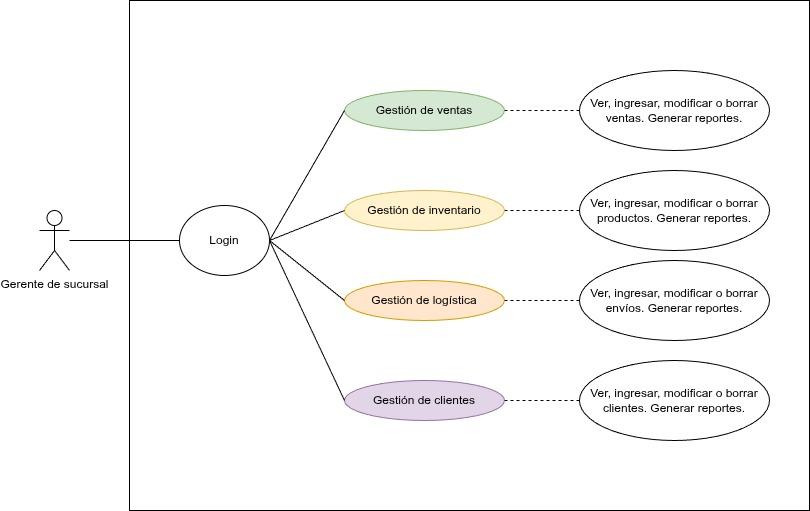
* **Diseño Inclusivo del Frontend:**
  + Asegurar que la interfaz (HTML/CSS) cumpla con estándares de accesibilidad como WCAG 2.1, incluyendo soporte para lectores de pantalla y navegación por teclado.
  + Garantizar que personas con discapacidades puedan usar la plataforma web de Perfulandia.

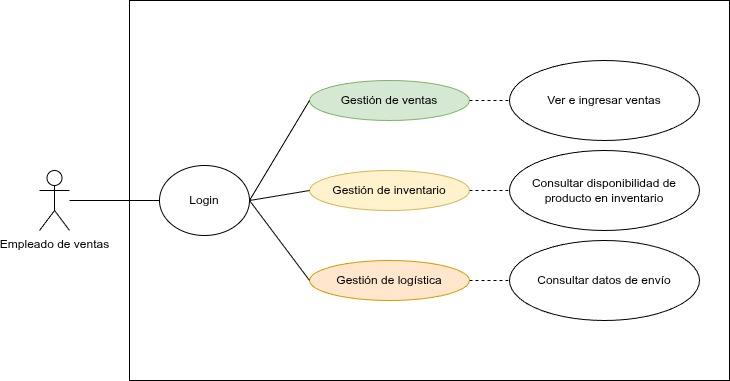
1. **Equidad en el Servicio:**
   * Priorizar que los microservicios (como "Pedidos" o "Logística") funcionen igual de bien en todas las sucursales, evitando que las áreas rurales o menos rentables tengan un servicio inferior.
2. **Uso Ético de los Datos Analíticos**

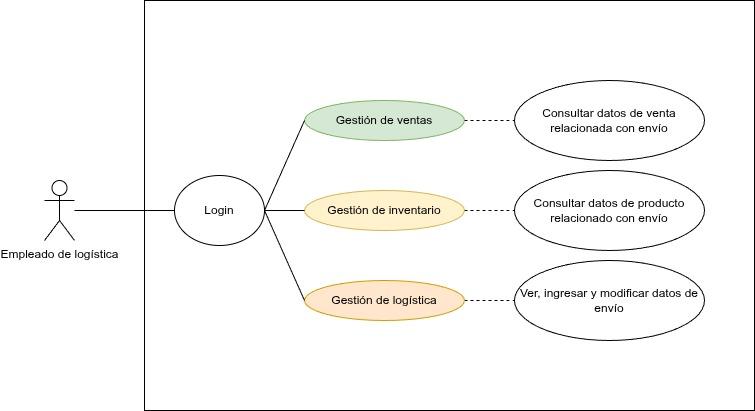
* **Reportes Responsables:**
  + El microservicio de "Reportes" debe evitar el uso de datos para prácticas manipuladoras (como aumentar precios basados en el historial de un cliente sin su conocimiento).
  + Respetar la autonomía del cliente y usar análisis solo para mejorar el servicio, no para explotarlo.
* **Spring Boot y MySQL:** Facilitan auditorías de seguridad y cumplimiento (como LGPD) al soportar configuraciones robustas de cifrado y transacciones seguras.
* **Contenedores y Kubernetes:** Reducen el impacto ambiental al optimizar recursos y permiten despliegues resilientes que minimizan interrupciones.
* **Frontend (HTML/CSS):** Su diseño accesible asegura inclusión, mientras que las APIs REST permiten transparencia en la comunicación con los usuarios.

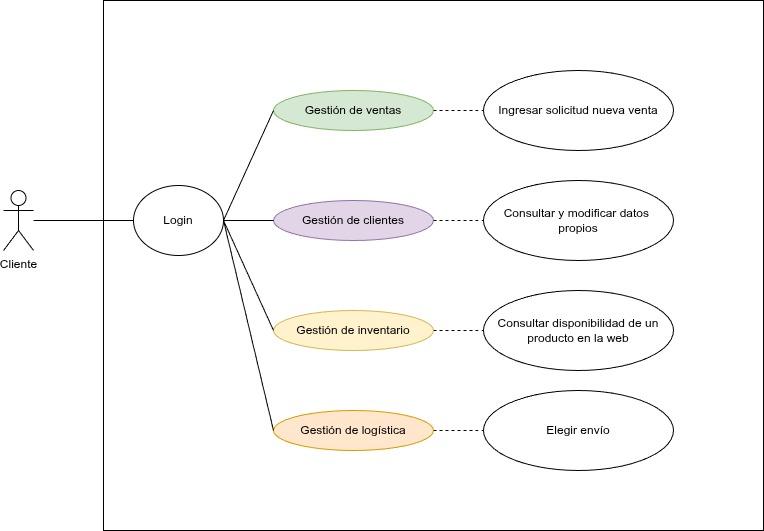
## 3.6. Diagramas de Casos de Uso

****

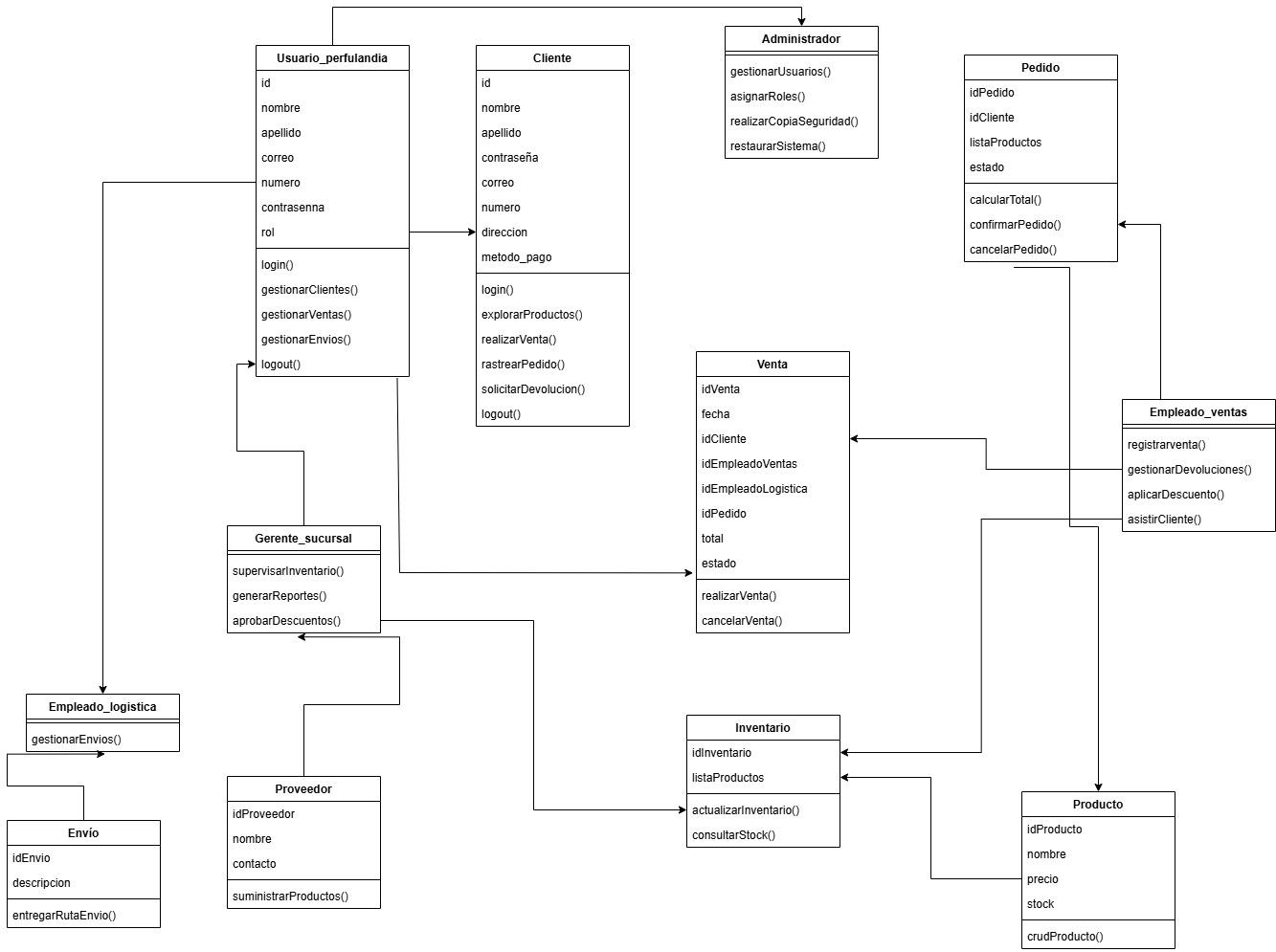
****

****

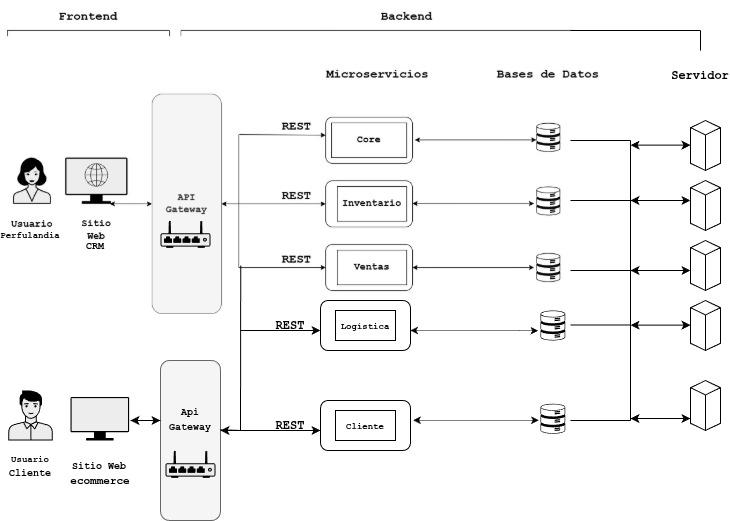
****

****

## 3.7. Diagrama de clases



## 3.8. Diagrama de despliegue



# 4. Planificación de la Migración

**Plan de migración método Strangler Fig para Perfulandia SPA:**

**Fase 1: Preparación y Monitoreo Inicial**

Duración estimada: 2–3 semanas

Acciones:

* Configurar infraestructura base (Kubernetes, Docker, RabbitMQ).
* Instalar y configurar Prometheus + Grafana.
* Auditar el sistema monolítico para identificar puntos de entrada (controladores REST, endpoints clave).
* Establecer un API Gateway que canalice todas las peticiones (p. ej., Zuul o Spring Cloud Gateway).
* **Se establece una capa de control que puede redirigir tráfico al monolito o a microservicios.**

**Fase 2: Extracción del Servicio Core (Usuarios y Sucursales)**

Duración estimada: 3–4 semanas

Motivo: Es una base de control de acceso, no dependiente de lógica transaccional compleja.

Acciones:

* Crear microservicio "Core" (usuarios, roles, autenticación, sucursales).
* Implementar RBAC y autenticación JWT.
* Redirigir endpoints de login/registro desde el API Gateway.
* Actualizar el monolito para usar el nuevo servicio vía REST o colas.
* **Gestión de usuarios y sucursales migrada; control de acceso centralizado.**

**Fase 3: Extracción del Servicio de Clientes**

Duración estimada: 3 semanas

Acciones:

* Crear microservicio "Clientes" con su propia base de datos.
* Manejar registro, login, perfiles y opiniones.
* Separar autenticación de clientes y empleados.
* Integrar frontend con este nuevo servicio.
* **Experiencia del cliente mejorada y desacoplada del backend de operaciones.**

**Fase 4: Migración de Inventario (Microservicio de Comercio)**

Duración estimada: 4 semanas

Acciones:

* Extraer lógica de productos e inventario.
* Asociar inventario a sucursales.
* Sincronizar stock con el microservicio "Ventas" y pedidos del monolito.
* Redirigir llamadas desde frontend/backend hacia este microservicio.
* **Stock gestionado de forma independiente, mejora rendimiento en horas peak.**

**Fase 5: Migración de Ventas y Facturación**

Duración estimada: 4 semanas

Acciones:

* Crear microservicio "Pedidos" (ventas, pagos, devoluciones).
* Conectar con Inventario y Clientes mediante eventos (RabbitMQ).
* Redirigir flujo de ventas desde API Gateway.
* Validar facturación y reembolsos.
* **Ventas desacopladas, sistema más resiliente y trazable.**

**Fase 6: Extracción de Logística**

Duración estimada: 4 semanas

Acciones:

* Extraer módulo de seguimiento de envíos y proveedores.
* Planificación de rutas y actualización de estado de pedidos.
* Integración con microservicios "Pedidos" y "Clientes".
* Probar con envíos simulados antes de cortar con el monolito.
* **Entregas gestionadas eficientemente y en tiempo real.**

**Fase 7: Monitoreo, Refactorización y Eliminación del Monolito**

Duración estimada: 2 semanas

Acciones:

* Evaluar logs, rendimiento y errores usando Prometheus/Grafana.
* Apagar módulos del monolito ya reemplazados.
* Refactorizar microservicios según lo aprendido.
* Documentar servicios, endpoints y flujos de comunicación.
* **Monolito eliminado, sistema 100% basado en microservicios.**

## 4.3. Matriz de riesgos

| **Riesgo** | **Descripción** | **Probabilidad** | **Impacto** | **Estrategias de Mitigación** |
| --- | --- | --- | --- | --- |
| **Resistencia al cambio del personal** | Los empleados pueden resistirse a adoptar el nuevo sistema por falta de familiaridad o miedo a la automatización. | Media | Alto | - Capacitación intensiva en Fase 1 (Sprint 2). - Comunicación transparente sobre beneficios (menos estrés operativo). - Involucrar a usuarios clave en pruebas y retroalimentación. |
| **Errores en la migración de datos** | Inconsistencias o pérdida de datos históricos al migrar del monolito a microservicios (ej. inventario, ventas). | Media | Alto | - Auditoría exhaustiva en Sprint 1. - Pruebas de consistencia en Fase 5 (Sprint 9). - Estrategia de respaldo y rollback documentada en Sprint 8. |
| **Fallas en la integración inicial** | Problemas de comunicación entre microservicios (ej. "Ventas" y "Logística") por configuraciones incorrectas de RabbitMQ o API Gateway. | Alta | Medio | - Pruebas de integración en Sprint 4. - Uso de mensajería asíncrona (RabbitMQ) bien configurada en Sprint 5. - Monitoreo con Prometheus desde Sprint 7. |
| **Sobrecarga en horas pico** | El sistema no escala adecuadamente bajo alta demanda (ej. días festivos), afectando rendimiento. | Media | Alto | - Configuración de Kubernetes para autoescalado en Sprint 10. - Pruebas de carga en Sprint 6. - Optimización basada en métricas de Grafana/Prometheus. |
| **Brechas de seguridad** | Exposición de datos sensibles (clientes, transacciones) por fallos en cifrado o autenticación. | Baja | Crítico | - Implementar cifrado AES-256 y TLS desde el inicio. - Auditorías de seguridad por microservicio en Fase 3. - Uso de JWT en "Clientes" (Sprint 3). |
| **Deuda técnica acumulada** | Iteraciones rápidas en metodología ágil generan código desorganizado o dependencias mal gestionadas. | Media | Medio | - Revisiones de código en cada sprint. - Implementación de CI/CD en Sprint 10. - Documentación clara en cada fase. |
| **Falta de experiencia del equipo** | El equipo no domina herramientas como Spring Boot, Kubernetes o Docker, retrasando el desarrollo. | Alta | Alto | - Capacitación en Sprint 2. - Prototipo dummy para práctica en Sprint 2. - Contratar consultores externos si es necesario. |
| **Interrupciones durante la convivencia** | Fallos al redirigir tráfico entre el monolito y microservicios mediante la API Gateway. | Media | Alto | - Implementación gradual en Fase 4 (Sprint 7). - Monitoreo activo con Prometheus. - Plan de rollback probado en Sprint 8. |
| **Costos imprevistos** | Subestimación de recursos (servidores, licencias, capacitación) eleva el presupuesto. | Media | Medio | - Estimación detallada en Sprint 1. - Uso de herramientas open-source (MySQL, RabbitMQ). - Revisión de costos en cada review. |
| **Retrasos en el cronograma** | Dependencias externas (proveedores, infraestructura) o problemas técnicos alargan los sprints. | Alta | Medio | - Buffer de tiempo en planificación (4 semanas por fase). - Priorización de microservicios críticos en Fase 2. - Daily Scrum para resolver bloqueos rápido. |
| **Insatisfacción del cliente** | Problemas en UX (ej. retrasos en pedidos) durante la transición afectan la experiencia del usuario. | Media | Alto | - Despliegue progresivo en staging (Sprint 6). - Microservicio "Notificaciones" para informar mantenimientos. - Pruebas con clientes simulados en Fase 3. |
| **Obsolescencia tecnológica** | Las herramientas elegidas (ej. RabbitMQ) quedan desactualizadas antes de completar la migración. | Baja | Medio | - Selección de tecnologías con soporte activo (Spring Boot, Kubernetes). - Evaluación continua en retrospectivas. - Flexibilidad para adoptar alternativas si es necesario. |

## 

# 6. Conclusión

La adopción de una arquitectura basada en microservicios representa una solución estratégica y efectiva para superar las limitaciones del sistema monolítico actual.

Esta propuesta mitiga los riesgos identificados, como fallos por acoplamiento fuerte, cuellos de botella en la base de datos y falta de escalabilidad, al tiempo que resuelve problemas operativos clave, incluyendo retrasos en el procesamiento de ventas, interrupciones en la plataforma web y dificultades para expandirse. La implementación de microservicios como "Core", "Pedidos", "Comercio" y "Logística", soportados por herramientas modernas como Spring Boot, MySQL, React.js, Docker, Kubernetes, RabbitMQ, Grafana y Prometheus, asegura un sistema escalable, resiliente y eficiente.

La planificación ágil en sprints permite una migración progresiva y adaptable, entregando valor desde las primeras fases y garantizando una transición fluida. En conclusión, esta arquitectura no solo resuelve los desafíos actuales, sino que posiciona a Perfulandia SPA como una empresa competitiva y preparada para un crecimiento sostenible en el futuro.