Informe de Proyecto

Perfulandia SPA

**Integrantes del equipo**: Cesar Henriquez, Xiomara Romero, Macarena Cruz

**Asignatura**: Desarrollo Full Stack

**Docente**: Victor Rosendo

**Fecha de entrega**: [Fecha]

## **Definición del Problema**

### **Descripción del sistema monolítico actual**

Perfulandia SPA es una empresa chilena en constante expansión, lo que ha llevado a la apertura de nuevas sucursales en distintas ciudades del país. Actualmente, la empresa opera con un sistema monolítico, es decir, una única aplicación donde todos los módulos de gestión, como ventas, inventarios, logística y clientes, están integrados en un solo bloque de software.

Este tipo de arquitectura, aunque funcional en los inicios de la empresa, ha demostrado ser un obstáculo conforme ha crecido la demanda y la complejidad operativa. La falta de modularidad y la fuerte interdependencia entre los componentes del sistema han generado problemas de rendimiento, dificultando su escalabilidad y mantenimiento.

### **Identificación del problema principal**

El principal inconveniente del sistema monolítico es su dificultad para escalar de manera eficiente y soportar cargas crecientes de transacciones y procesamiento de datos. A medida que la empresa ha expandido su operación, la arquitectura monolítica ha mostrado limitaciones evidentes, tales como:

* **Cuellos de botella:** El aumento en el volumen de usuarios y transacciones afecta la velocidad de respuesta del sistema, causando latencias y tiempos de inactividad.
* **Dificultad de mantenimiento:** Cualquier actualización o corrección en el código requiere pruebas extensas para garantizar que no afecte a otros módulos de la aplicación.
* **Falta de flexibilidad:** La incorporación de nuevas funcionalidades implica modificar una gran parte del código existente, lo que ralentiza el desarrollo e implementación de mejoras.

### **Impacto del problema en el negocio**

Los problemas mencionados generan repercusiones significativas en la operación y crecimiento de la empresa, tales como:

* **Interrupción de operaciones:** Los fallos en el sistema pueden detener las ventas, la gestión de inventarios y el procesamiento de pedidos, afectando la rentabilidad y la satisfacción del cliente.
* **Dificultad para escalar:** El crecimiento de la empresa y la apertura de nuevas sucursales requieren un sistema que pueda adaptarse rápidamente a mayores volúmenes de transacciones y nuevos requerimientos.
* **Altos costos de mantenimiento:** Las actualizaciones del sistema demandan un tiempo prolongado de desarrollo y pruebas, lo que incrementa los costos operativos y reduce la capacidad de innovación tecnológica de la empresa.

### **Objetivo General**

Implementar una arquitectura basada en microservicios para reemplazar el sistema monolítico actual, con el propósito de mejorar la escalabilidad, el rendimiento y la disponibilidad de la plataforma de Perfulandia SPA.

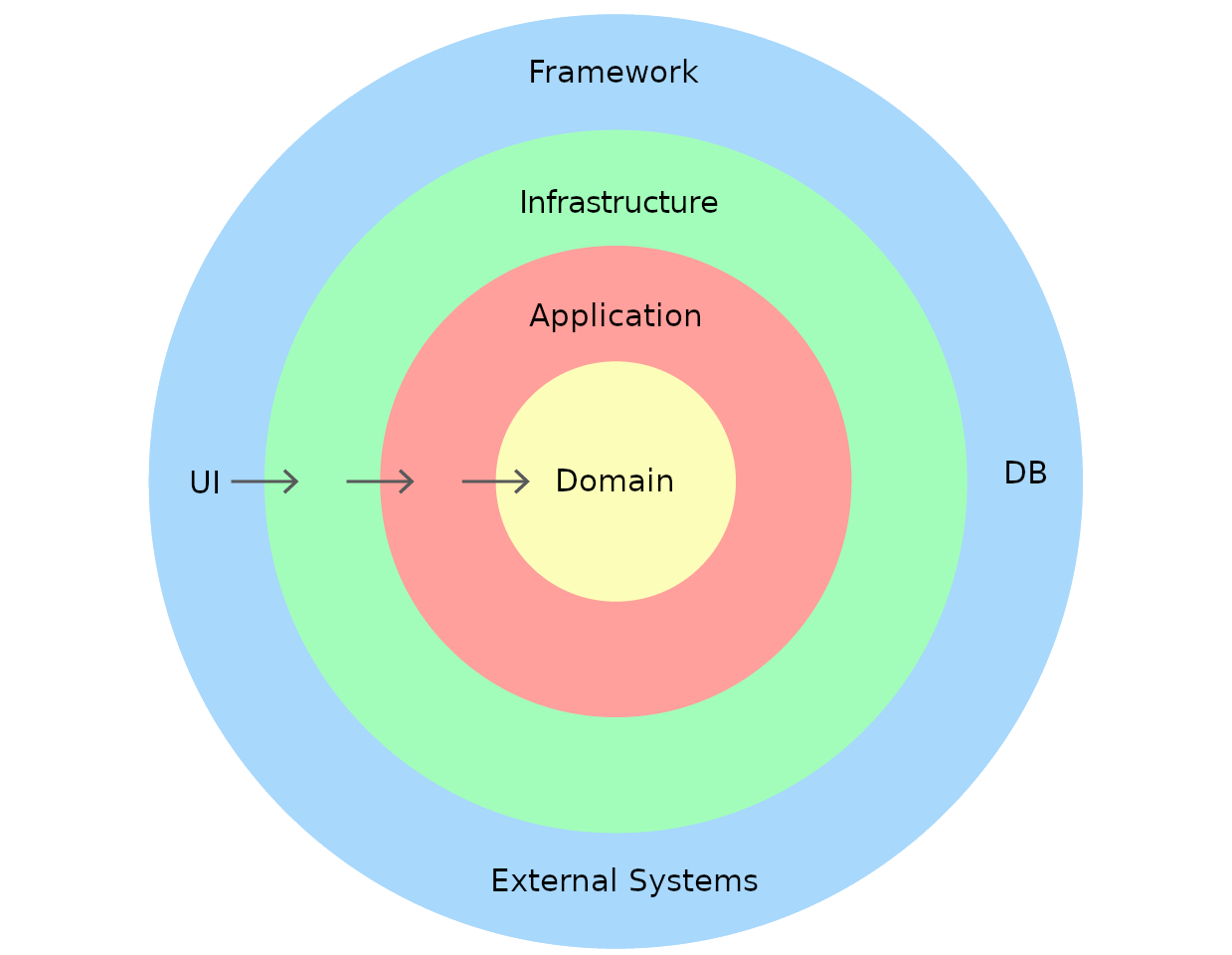
### **Objetivos Específicos**

1. Realizar un análisis detallado del sistema monolítico para identificar sus principales puntos débiles.
2. Diseñar una estrategia de microservicios alineada con los requerimientos del negocio.
3. Seleccionar tecnologías adecuadas para la implementación, despliegue y monitoreo de la nueva infraestructura.
4. Planificar y ejecutar la migración de manera progresiva, asegurando la continuidad del servicio.
5. Implementar mecanismos de seguridad, monitoreo y control de calidad para garantizar el correcto funcionamiento del nuevo sistema.

## **2. Estrategia de Microservicios**

### **Tipo de estrategia usada**

Se adoptará una **estrategia basada en dominios (Domain-Driven Design)**, dividiendo el sistema en microservicios según las funciones del negocio, tales como: gestión de inventarios, ventas, logística, clientes y pedidos.



### **Justificación técnica y de diseño de la estrategia elegida**

La estrategia basada en dominios permite que cada microservicio sea autónomo y se pueda escalar de forma independiente. Además, cada servicio puede ser desarrollado, desplegado y mantenido por equipos específicos sin afectar otras partes del sistema, lo que facilita la gestión de la infraestructura a medida que la empresa crece.

### **Beneficios de adoptar esta estrategia**

* **Escalabilidad**: Permite escalar de forma independiente cada microservicio según la demanda de recursos, lo que mejora la eficiencia.
* **Mantenimiento más fácil**: Los microservicios son más fáciles de modificar y actualizar sin que impacten al resto del sistema.
* **Resiliencia**: Un fallo en un microservicio no afecta a toda la aplicación, lo que incrementa la disponibilidad del sistema.

## **3. Herramientas Utilizadas**

### **Lenguaje de Programación y Framework: Spring Boot (Java)**

Spring Boot se utiliza para el desarrollo de microservicios debido a su capacidad para crear aplicaciones autónomas, su integración nativa con bases de datos y su compatibilidad con arquitecturas basadas en RESTful.

**Justificación:** Spring Boot fue elegido como el framework principal para el desarrollo de microservicios por varias razones:

* **Simplicidad y rapidez en el desarrollo**: Permite crear aplicaciones autónomas sin necesidad de configuraciones complejas, reduciendo significativamente el tiempo de desarrollo.
* **Arquitectura RESTful**: Proporciona una integración sencilla para la construcción de API REST, facilitando la comunicación entre microservicios.
* **Compatibilidad con bases de datos**: Incluye soporte nativo para PostgreSQL y otros motores de bases de datos, lo que agiliza la gestión de datos.
* **Escalabilidad y modularidad**: Facilita la implementación de una arquitectura de microservicios al permitir que cada servicio se desarrolle y despliegue de manera independiente.
* **Integración con herramientas empresariales**: Spring Boot es ampliamente utilizado en la industria y se integra fácilmente con herramientas como Spring Cloud, que facilita la gestión de configuración, descubrimiento de servicios y balanceo de carga.

### **Orquestación de Microservicios: Kubernetes**

Kubernetes facilita la gestión y orquestación de microservicios en contenedores, garantizando que cada uno se ejecute de manera independiente pero coordinada.

**Justificación:** Kubernetes es la plataforma elegida para la orquestación de microservicios porque permite administrar la infraestructura de manera eficiente:

* **Gestión automatizada de contenedores**: Kubernetes facilita la asignación de recursos a cada microservicio, optimizando el uso de CPU y memoria.
* **Alta disponibilidad**: Detecta fallos en los microservicios y los reinicia automáticamente para garantizar la continuidad del servicio.
* **Balanceo de carga y escalabilidad**: Permite ajustar dinámicamente la cantidad de instancias de un microservicio según la demanda, evitando sobrecargas en el sistema.
* **Despliegue y actualización sin interrupciones**: Kubernetes facilita la implementación de nuevas versiones de los microservicios sin afectar la operatividad del sistema.
* **Portabilidad**: Permite ejecutar microservicios en distintos entornos (nube, servidores on-premise o híbridos) sin necesidad de modificar la infraestructura.

### **Contenerización: Docker**

Docker permite ejecutar cada microservicio en contenedores aislados, lo que facilita la migración, el despliegue y el mantenimiento del sistema.

**Justificación:** Docker se usa para empaquetar y desplegar cada microservicio en contenedores independientes, lo que aporta múltiples beneficios:

* **Aislamiento**: Cada microservicio se ejecuta en su propio entorno, sin interferencias con otros servicios.
* **Consistencia**: Garantiza que las aplicaciones se ejecuten de la misma manera en todos los entornos (desarrollo, prueba y producción).
* **Facilidad de migración y despliegue**: Permite mover microservicios entre servidores o la nube sin cambios en la configuración.
* **Integración con Kubernetes**: Docker se complementa perfectamente con Kubernetes para la gestión de los contenedores.
* **Optimización de recursos**: Reduce el consumo de recursos en comparación con máquinas virtuales, lo que mejora la eficiencia del sistema.

### **Base de Datos: PostgreSQL**

PostgreSQL se elige por ser una base de datos relacional robusta, que maneja grandes volúmenes de datos con alta integridad, lo cual es crucial para las operaciones transaccionales de Perfulandia SPA.

**Justificación:** PostgreSQL se seleccionó como el sistema de gestión de bases de datos por las siguientes razones:

* **Integridad y fiabilidad**: PostgreSQL cumple con el estándar ACID, lo que garantiza la integridad de los datos en operaciones transaccionales.
* **Escalabilidad**: Es capaz de manejar grandes volúmenes de datos y puede escalar horizontalmente si la demanda crece.
* **Soporte para JSON y NoSQL**: Además de datos estructurados, PostgreSQL permite manejar documentos JSON, lo que brinda flexibilidad en el almacenamiento de datos.
* **Seguridad**: Ofrece mecanismos avanzados de autenticación, cifrado y control de acceso.
* **Código abierto y sin costos de licencia**: A diferencia de otras bases de datos comerciales, PostgreSQL no requiere licencias costosas, reduciendo los costos operativos.

### **Comunicación entre Microservicios: Apache Kafka**

Apache Kafka se utiliza como sistema de mensajería para permitir la comunicación asíncrona entre los microservicios, asegurando que cada uno pueda procesar eventos de forma independiente.

**Almacenamiento en la Nube:**

Amazon S3 (Simple Storage Service) es un servicio de almacenamiento en la nube que permite gestionar grandes volúmenes de datos de forma escalable, segura y accesible. Se utiliza para almacenar archivos estáticos como imágenes, documentos, registros de transacciones, copias de seguridad, entre otros.

#### **Justificación:**

1. **Escalabilidad y Alta Disponibilidad:**

* Se adapta automáticamente a la cantidad de datos almacenados sin necesidad de aprovisionamiento manual.
* Replicación de datos en múltiples zonas de disponibilidad para garantizar la continuidad del servicio.

1. **Seguridad y Cumplimiento:**

* Cifrado de datos en reposo y en tránsito.
* Políticas de control de acceso detalladas mediante AWS IAM (Identity and Access Management).
* Cumple con normativas como **GDPR, HIPAA, ISO 27001**, entre otras.

1. **Eficiencia y Costos:**
   * Pago por uso según la cantidad de datos almacenados y transferidos.
   * Opciones de almacenamiento en distintas clases para optimizar costos (S3 Standard, S3 Glacier, etc.).
2. **Durabilidad y Resistencia a Fallos:**
   * Garantiza **99.999999999%** (11 nueves) de durabilidad de los datos.
   * Recuperación automática ante fallos en hardware o infraestructura.

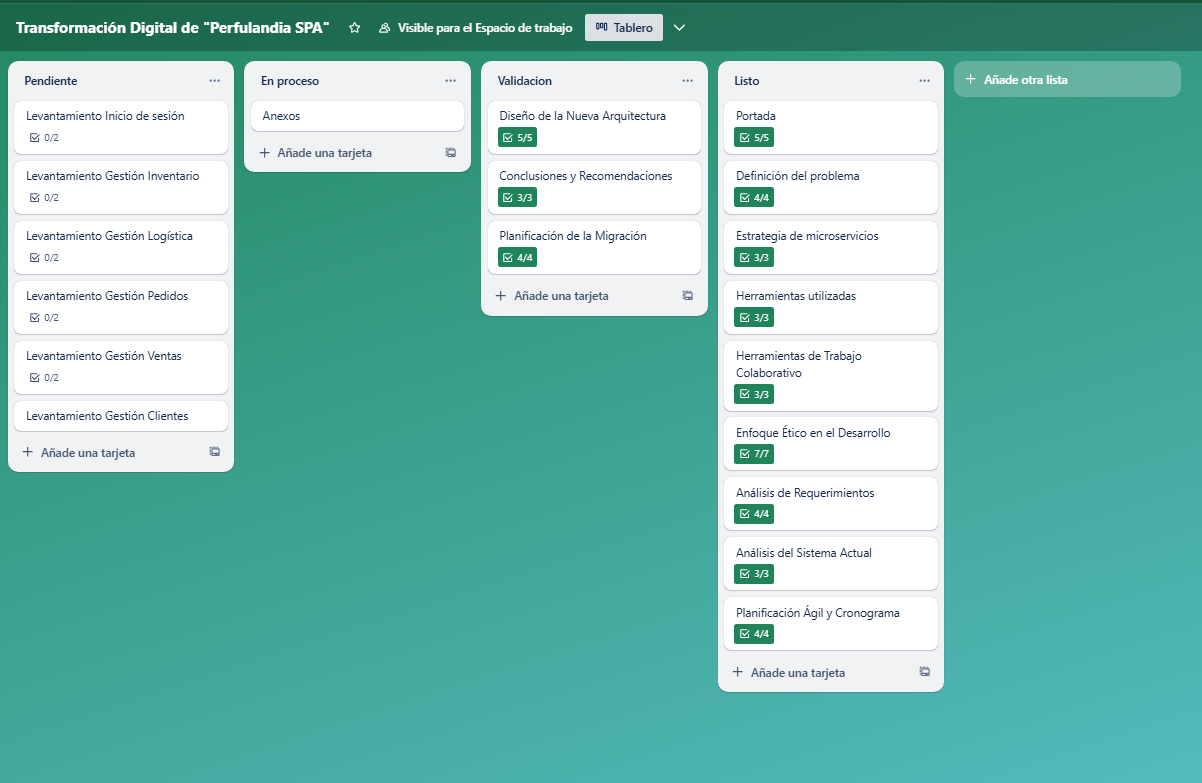
## **4. Herramientas de Trabajo Colaborativo**

### **Herramientas utilizadas:**

* **Discord**: Para la comunicación en tiempo real, permitiendo una coordinación efectiva entre los miembros del equipo.
* **Trello**: Para la planificación y seguimiento de tareas de manera visual.
* **GitHub**: Para el control de versiones y la integración continua.

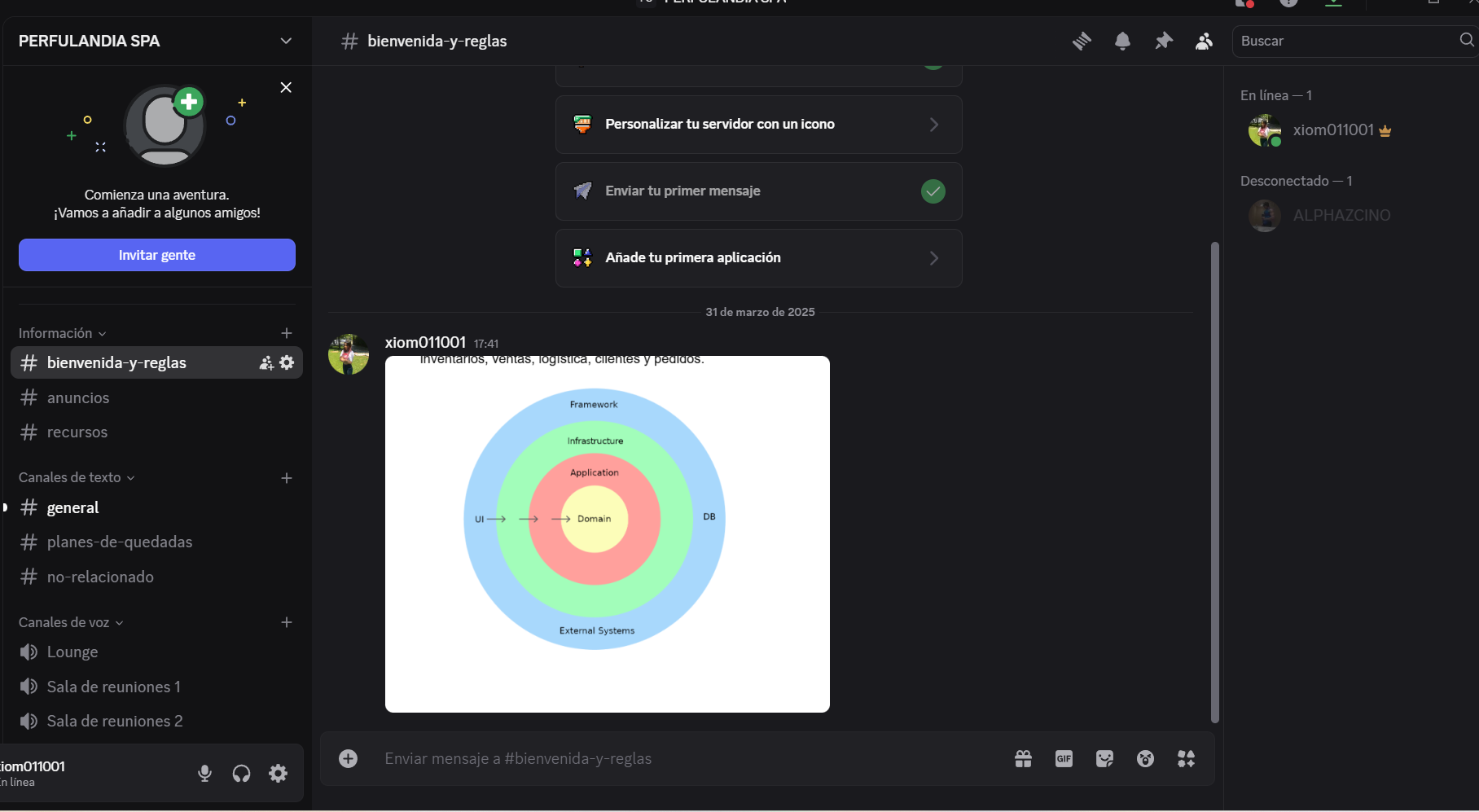
### **Ejemplos de cómo se organizó el trabajo:**

* **Discord**: Se crearon canales para discutir sobre aspectos técnicos, reuniones de seguimiento y problemas puntuales de la migración.
* **Trello**: Las tareas del proyecto fueron divididas en sprints y asignadas a los miembros del equipo. Cada tarea se movía a través de diferentes estados (Por hacer, En progreso, Hecho).
* **GitHub**: Utilizamos repositorios para gestionar el código y facilitar la colaboración mediante pull requests y revisiones de código.

****

**Imagen 1:** Tablero de Trello.

Enlace: [Enlace trello](https://trello.com/invite/b/67e05f41fb73f243811aa40a/ATTI203ca42a6b357d7adb828f35d91c4e207EF9CC3B/transformacion-digital-de-perfulandia-spa)



**Imagen 2:** Canal de discord del equipo de trabajo.

Enlace: [Enlace discord](https://discord.gg/e9JjeeQP)

## **5. Enfoque Ético en el Desarrollo**

### **Desafíos éticos y soluciones adoptadas:**

### **1. Privacidad de los Datos**

#### **Desafío**

El manejo de información sensible de clientes, como datos personales, historiales de compra y preferencias, implica riesgos relacionados con la privacidad. En Chile, la **Ley de Protección de la Vida Privada (Ley N° 19.628)** regula el uso de datos personales, exigiendo que las empresas implementen medidas para su resguardo y evitar accesos no autorizados.

#### **Soluciones Adoptadas**

* **Cifrado de datos:** Se implementa **encriptación** para los datos en reposo y en tránsito, garantizando que la información esté protegida en todo momento.
* **Cumplimiento de la Ley N° 19.628:** Se garantiza que los clientes puedan **acceder, rectificar o eliminar** sus datos cuando lo soliciten.

### **2. Seguridad**

#### **Desafío**

La seguridad en una arquitectura de microservicios es crítica, ya que múltiples puntos de entrada pueden ser explotados por atacantes. La protección de datos debe considerar ataques como **inyección SQL, ataques de phishing y acceso no autorizado**.

#### **Soluciones Adoptadas**

* **Autenticación y autorización robustas:** Se implementan estándares como **OAuth 2.0**  para gestionar accesos de manera segura.
* **Control de acceso basado en roles (RBAC):** Se garantiza que los usuarios solo accedan a la información necesaria para sus funciones.
* **Pruebas de seguridad continuas:** Se realizan **pruebas de penetración (pentesting)** y auditorías de código para identificar vulnerabilidades antes de que sean explotadas.
* Firewalls y sistemas de detección de intrusos (IDS/IPS) para mitigar ataques externos.

### **3. Responsabilidad en el Despliegue**

#### **Desafío**

Migrar de un sistema monolítico a una arquitectura de microservicios implica cambios operativos que pueden generar **fallas en el servicio**, afectar la experiencia del usuario y provocar pérdidas económicas si no se gestiona adecuadamente.

#### **Soluciones Adoptadas**

* **Despliegue gradual:** comenzando con pruebas en entornos de desarrollo y staging
* Planes de contingencia, permitiendo revertir cambios en caso de fallos críticos.
* Capacitación del equipo técnico, asegurando que los desarrolladores y administradores comprendan el nuevo entorno.

### **4. Impacto en los Puestos de Trabajo**

#### **Desafío**

La implementación de nuevas tecnologías puede generar incertidumbre en los empleados y afectar la moral del equipo si no se manejan adecuadamente los cambios.

#### **Soluciones Adoptadas**

* Programas de formación y capacitación para facilitar la adaptación a la nueva plataforma.
* Automatización responsable, asegurando que las nuevas herramientas mejoren la productividad sin desplazar a los trabajadores.
* Soporte técnico y asistencia, garantizando una transición fluida para los usuarios del sistema.

## **5. Cumplimiento Normativo en Chile**

Para garantizar el cumplimiento de las normativas vigentes en Chile, se tomaron en cuenta las siguientes regulaciones:

* Ley de Protección de Datos Personales (Ley 19.628): Se aseguraron medidas para proteger la privacidad y el uso adecuado de los datos de los clientes.
* Normativas del Servicio de Impuestos Internos (SII): Se verificó el cumplimiento en la gestión de datos fiscales y transacciones electrónicas.

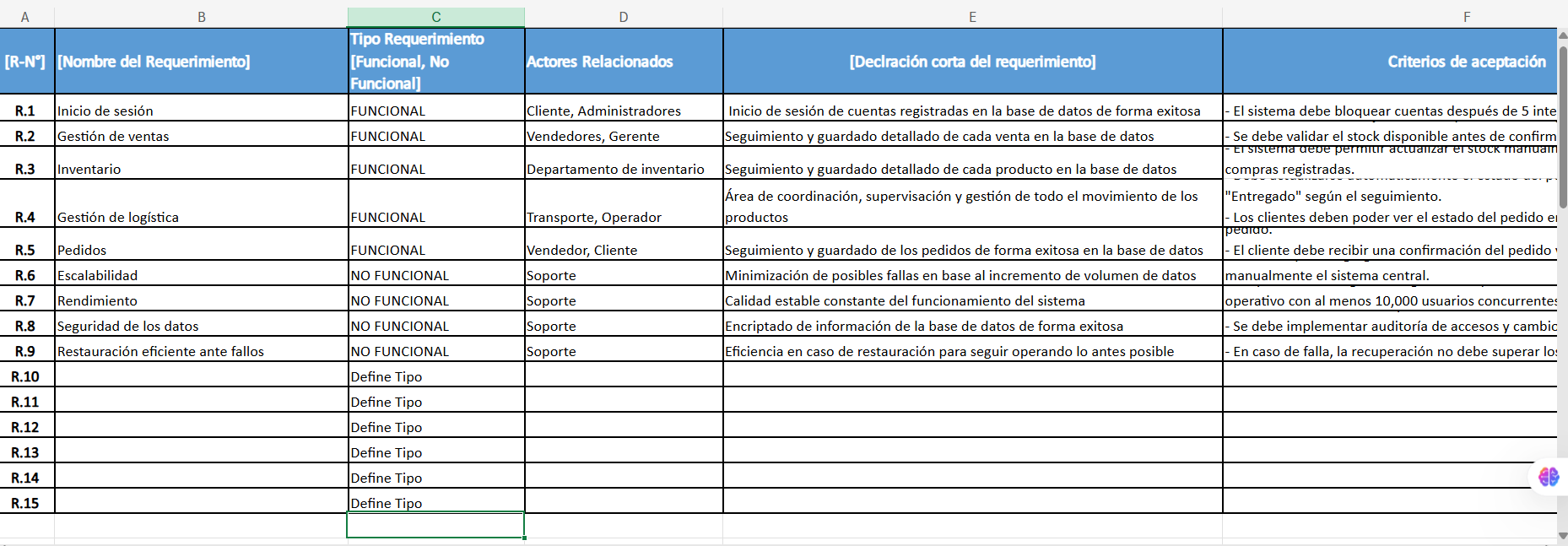
## **6. Análisis de Requerimientos**

### **Requisitos funcionales:**

* Gestión de ventas, inventarios, logística, pedidos, inicio de sesión y clientes.

### **Requisitos no funcionales:**

* Escalabilidad y rendimiento.
* Alta disponibilidad y recuperación ante fallos.
* Seguridad de los datos de clientes.
* Monitoreo y control en tiempo real del estado de los microservicios.



Enlace requisitos funcionales: [Requisitos funcionales](https://1drv.ms/x/c/6d9fb4cd48f30c72/EQj3XiMgXd1KmQCve8NferUBA1t7jKLUXXJ3VYM4AHPDLg?e=ejFbGT)

### **Entrevistas y Encuestas:**

* **Administrador**: Necesita un sistema que facilite la gestión de usuarios y la configuración de permisos.
* **Vendedor**: Requiere acceso rápido a datos de inventario y ventas.
* **Cliente**: Necesita una plataforma sencilla para realizar compras, consultar historial y realizar pagos de manera segura.

## **7. Análisis del Sistema Actual**

#### **Arquitectura General del Sistema Monolítico**

El sistema monolítico actual de Perfulandia SPA ha sido desarrollado como una única aplicación en la que todos los módulos (ventas, inventario, logística, clientes, facturación, etc.) están interconectados dentro del mismo código base. Esto significa que todas las funcionalidades comparten los mismos recursos, bases de datos y lógica de negocio.

Si bien esta arquitectura es común en los inicios de una empresa, con el tiempo se vuelve difícil de gestionar debido a su tamaño y complejidad. En una aplicación monolítica:

* Todas las funcionalidades dependen unas de otras.
* Cada cambio o actualización debe implementarse en toda la aplicación.
* Se requiere desplegar todo el sistema cada vez que se introduce una nueva funcionalidad, lo que incrementa el riesgo de errores.

#### **Puntos Débiles Detectados**

Después de un análisis detallado del sistema monolítico, se identificaron los siguientes problemas clave:

##### **1. Fallas frecuentes y afectación global**

* Debido a que todos los módulos están interconectados, un error en una parte del sistema puede afectar a toda la aplicación.
* Un fallo en el módulo de ventas, por ejemplo, puede impedir el acceso a la gestión de inventarios o bloquear la generación de facturas.
* En horarios de alta demanda, como temporadas de ofertas o festividades, un error menor puede interrumpir toda la operación, afectando la experiencia del usuario y las ventas.

##### **Dificultad para actualizar y agregar nuevas funcionalidades**

* Cualquier modificación requiere modificar el código de toda la aplicación, lo que hace que el proceso de desarrollo sea lento y propenso a errores.
* Implementar una nueva funcionalidad, como un sistema de recomendaciones personalizadas para los clientes, puede afectar módulos existentes que no están directamente relacionados.
* Se necesita un equipo de desarrollo especializado en la aplicación completa, lo que hace difícil la delegación de tareas a diferentes equipos.

##### **Cuellos de botella en el rendimiento y escalabilidad**

* La aplicación está diseñada para ejecutarse como una sola unidad, lo que limita la posibilidad de escalar componentes individuales según la demanda.
* Si el módulo de ventas recibe muchas solicitudes simultáneas, todo el sistema se ralentiza porque comparte los mismos recursos con los demás módulos.
* La base de datos centralizada genera tiempos de respuesta lentos cuando el número de usuarios aumenta significativamente.

##### **4. Alta dependencia del código y dificultad en la integración con nuevas tecnologías**

* La tecnología usada en el sistema monolítico no es flexible para adoptar nuevas herramientas o lenguajes de programación.
* Si se quiere integrar inteligencia artificial o analítica avanzada, se deben realizar cambios profundos en todo el código.
* La dependencia de un único servidor y estructura rígida impide la adopción de modelos de infraestructura en la nube de manera eficiente.

Debido a estos problemas, se ha determinado que el sistema monolítico ya no es sostenible para el crecimiento de Perfulandia SPA. La migración a una arquitectura de microservicios permitirá solucionar estos inconvenientes, ya que cada funcionalidad se podrá gestionar de forma independiente, mejorando la escalabilidad, el rendimiento y la flexibilidad del sistema.

## **8. Diseño de la Nueva Arquitectura**

### **Descripción general de la arquitectura basada en microservicios:**

La nueva arquitectura estará basada en microservicios independientes, cada uno encargado de una parte específica del negocio (ventas, inventario, logística, etc.).

### **Diagramas:**

* **Diagrama de Actores de Alto Nivel**: Los diferentes actores (usuarios, clientes, administradores) interactúan con los microservicios específicos.

| Funcionalidad | Clientes | Administradores | Vendedores | Logística | Soporte Técnico |
| --- | --- | --- | --- | --- | --- |
| Gestión de ventas | X | X | X |  |  |
| Gestión de pedidos | X | X | X | X |  |
| Gestión de inventarios |  | X | X | X |  |
| Gestión de Logística |  | X |  | X |  |
| Autenticación y seguridad | X | X | X | X | X |
| Gestión de Clientes | X | X | X |  |  |
| Monitoreo y Control |  | X |  |  | X |

Tabla 1: Actores de alto Nivel.

**Explicación de la Tabla**

**Clientes**: Pueden realizar compras, gestionar pedidos y autenticarse en el sistema.

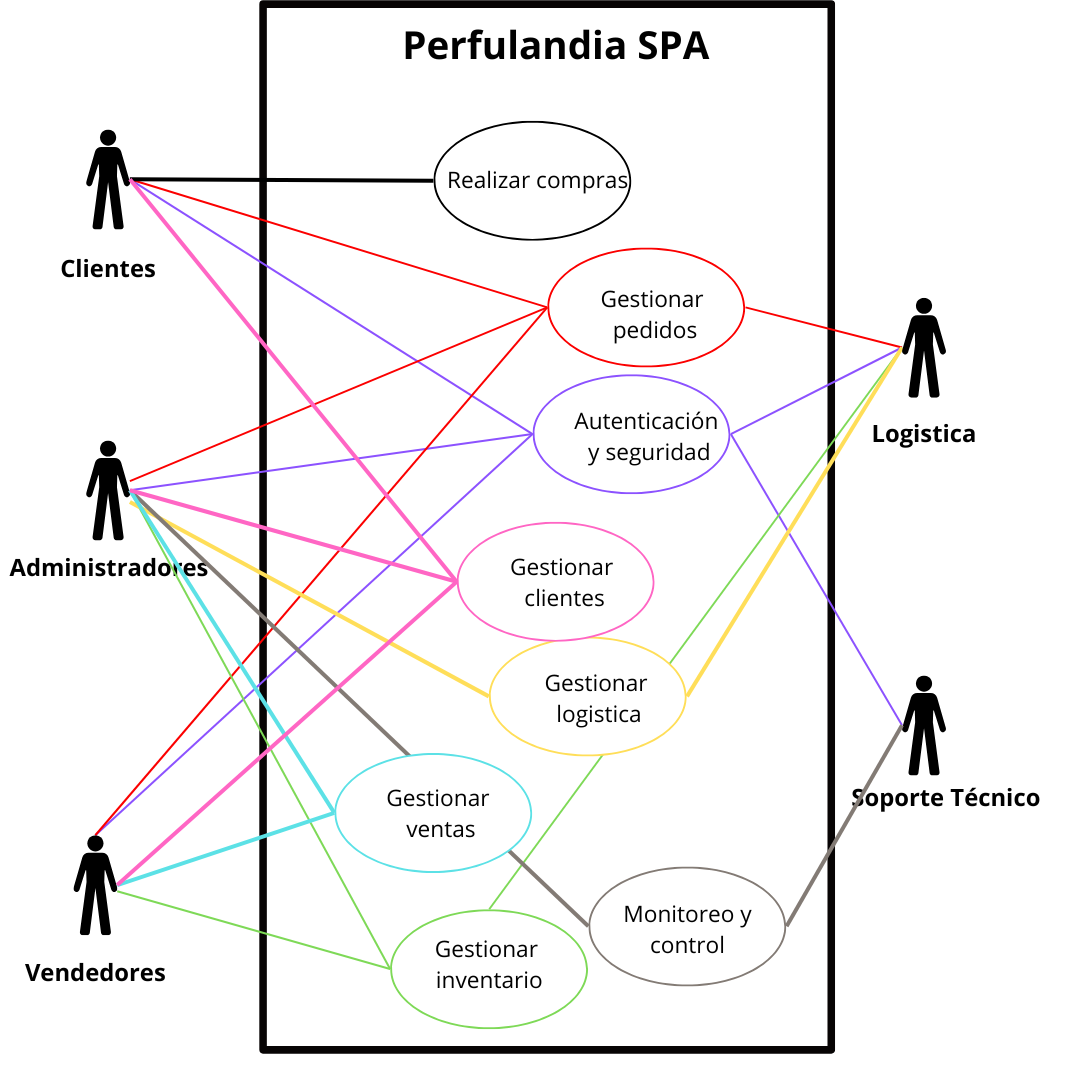
**Administradores**: Tienen acceso total para gestionar ventas, inventarios, clientes y logística, además de monitorear los microservicios y autenticarse en el sistema.

**Vendedores**: Gestionan ventas, pedidos e inventarios, pero no tienen acceso a logística ni monitoreo de microservicios.

**Logística**: Se encarga del seguimiento de pedidos y la gestión del inventario.

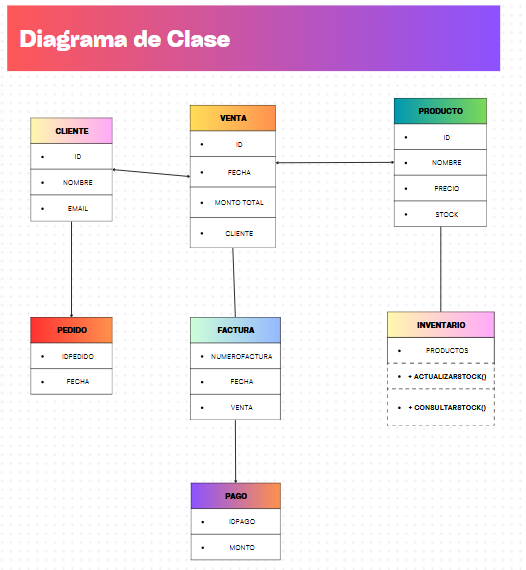
**Soporte Técnico**: Garantiza el monitoreo de microservicios y la seguridad del sistema.

* **Diagrama de Casos de Uso**: Define los casos de uso y cómo cada servicio interactúa con los demás.



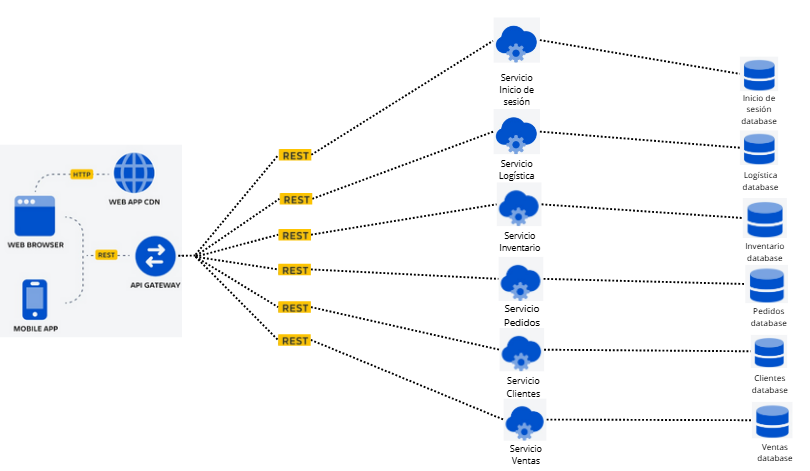
**Imagen 3:** Diagrama de Casos de uso.

* **Diagrama de Clases y Despliegue**: Representación de las clases dentro de los microservicios y su despliegue en contenedores.



**Imagen 4:** Diagrama de clases y despliegue

* **Diagrama de arquitectura de microservicios:** Representación visual que muestra cómo se organiza y comunica un sistema basado en microservicios.



**Imagen 5:** Diagrama de arquitectura de microservicios.

## **9. Planificación de la Migración**

La migración del sistema monolítico a una arquitectura basada en microservicios se llevará a cabo en varias fases para garantizar una transición segura y eficiente sin afectar la operatividad de la empresa.

### **Fases de Migración**

La migración se realizará en tres fases principales:

1. **Migración de Microservicios Clave** Se comenzará con los microservicios más críticos para el negocio, como:

* **Microservicio de Ventas**: Responsable de gestionar las transacciones comerciales, generar facturas y actualizar el historial de compras de los clientes.
* **Microservicio de Inventarios**: Controlará el stock de productos en tiempo real y enviará alertas cuando los niveles sean bajos.

1. Durante esta fase, se desarrollarán estos microservicios por separado y se probará su compatibilidad con el sistema monolítico actual.
2. **Pruebas de Integración y Rendimiento** Una vez implementados los primeros microservicios, se llevarán a cabo pruebas de integración para asegurarse de que funcionan correctamente junto con el sistema existente.

* **Pruebas de carga**: Se simulará un alto volumen de transacciones para evaluar la capacidad de respuesta del nuevo sistema.
* **Pruebas de comunicación**: Se validará que los microservicios puedan intercambiar información correctamente utilizando Apache Kafka.

1. **Despliegue Gradual con Monitoreo**

* Se habilitarán los microservicios en entornos controlados (entornos de prueba y staging).
* Se implementará un monitoreo en tiempo real para detectar posibles fallos antes de lanzar completamente el nuevo sistema.
* Una vez comprobada la estabilidad, se procederá con la migración de otros módulos (logística, clientes y pedidos) hasta completar la transformación total.

### **Consideraciones de Compatibilidad**

Para evitar interrupciones en el servicio durante la migración, se seguirán las siguientes estrategias:

* **Estrategia de doble ejecución (Strangler Pattern)**: Durante la transición, el sistema monolítico y los nuevos microservicios operarán en paralelo. A medida que los microservicios se validen, se irán desactivando progresivamente las funcionalidades del sistema antiguo.
* **Adaptadores y APIs**: Se desarrollarán APIs intermediarias para permitir que los nuevos microservicios se comuniquen con el sistema legacy mientras dure la migración.
* **Capacitación del equipo**: Se proporcionarán entrenamientos a los empleados para que puedan adaptarse a las nuevas herramientas sin afectar la productividad.

Con este enfoque, se garantizará una migración controlada, evitando riesgos y minimizando el impacto en la empresa.

**Identificación de riesgos y su plan de mitigación**

#### **Riesgo 1: Fallos en la comunicación entre microservicios**

#### **Posibles causas:**

* Configuración incorrecta de **Apache Kafka**.
* Errores en la serialización/deserialización de mensajes.
* Latencia alta o pérdida de mensajes en la mensajería.

**Plan de mitigación:**

* **Pruebas de comunicación tempranas**: Se realizarán pruebas de mensajería antes de la migración total..
* **Implementación de reintentos**: Uso de estrategias de reintento y almacenamiento en caso de fallas.

#### **Riesgo 2: Pérdida de datos en la migración**

**Posibles causas:**

* Errores en la transformación de datos entre el sistema monolítico y los microservicios.
* Inconsistencia en los datos entre bases de datos antiguas y nuevas.

**Plan de mitigación:**

* **Uso de Apache Kafka para sincronización**: Kafka actuará como intermediario para garantizar la transferencia ordenada de datos.
* **Backup previo**: Se realizarán copias de seguridad antes de migrar.
* **Pruebas de consistencia de datos**: Se validará la integridad de la información migrada con scripts de comparación.

#### **Riesgo 3: Problemas de rendimiento en los nuevos microservicios**

**Posibles causas:**

* Microservicios mal optimizados que generan latencias elevadas.
* Exceso de tráfico en Kafka causando cuellos de botella.

**Plan de mitigación:**

* **Pruebas de carga con Apache JMeter**: Se simularán picos de tráfico para identificar puntos débiles.
* **Optimización de consultas a bases de datos**: Uso de índices y caching para mejorar tiempos de respuesta.
* **Autoescalado con Kubernetes**: Implementación de escalado automático en caso de alta demanda.

#### **Riesgo 4: Interrupción del servicio durante la transición**

**Posibles causas:**

* Dependencias no resueltas entre el sistema monolítico y los nuevos microservicios.
* Fallos en la estrategia de doble ejecución (**Strangler Pattern**).

**Plan de mitigación:**

* **Implementación en paralelo**: Se ejecutará el sistema monolítico junto con los microservicios para asegurar una transición sin cortes.
* **Despliegue progresivo en entornos controlados**: Se iniciará en **staging** antes de pasar a producción.
* **Rollback inmediato**: En caso de fallos críticos, se podrá volver a la versión anterior rápidamente.

#### **Riesgo 5: Falta de adaptación del equipo al nuevo sistema**

**Posibles causas:**

* Resistencia al cambio por parte del equipo.
* Falta de capacitación en nuevas tecnologías como Kafka y Kubernetes.

**Plan de mitigación:**

* Capacitaciones en Kafka, Kubernetes y monitoreo de microservicios.
* Sesiones prácticas con entornos sandbox antes de la migración.
* Documentación detallada sobre la nueva arquitectura.

## **10. Planificación Ágil y Cronograma**

### **Enfoque Ágil Usado: Scrum**

Se utilizó **Scrum** como metodología ágil debido a su flexibilidad y enfoque iterativo. El equipo trabajó en **sprints de dos semanas**, permitiendo la entrega continua de funcionalidades. Durante cada sprint se realizaron:

* **Reuniones diarias:** Para discutir avances, bloqueos y próximos pasos.
* **Revisiones de sprint**: Para evaluar los desarrollos completados y recibir retroalimentación.
* **Retrospectivas**: Para identificar áreas de mejora en la ejecución del proyecto.

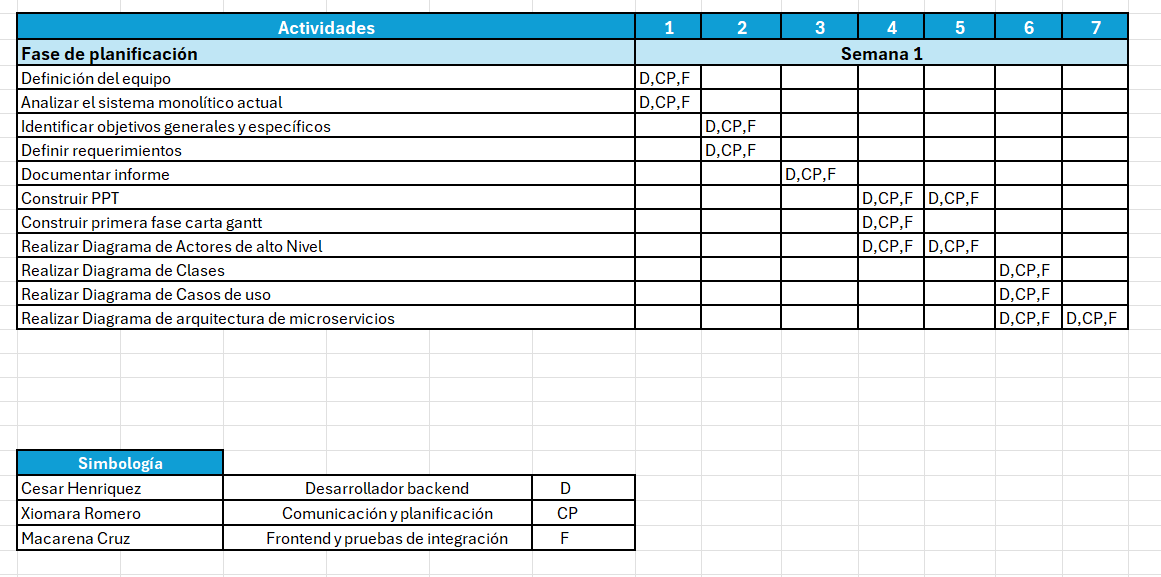
### **Roles del Equipo**

Cada miembro asume responsabilidades específicas dentro del marco de trabajo Scrum:

* **César Henríquez (Desarrollador Backend y Arquitecto de Microservicios)**: Responsable de diseñar y desarrollar la arquitectura de microservicios, asegurando una integración eficiente entre servicios.
* **Xiomara Romero (Scrum Master y Coordinadora de Tareas)**: Facilitó las reuniones, removió obstáculos y se aseguró de que el equipo siguiera los principios ágiles.
* **Macarena Cruz (Desarrolladora Frontend y Tester de Integración)**: Responsable de la interfaz de usuario y la validación de interacciones entre microservicios.

### **Carta Gantt:**

El proyecto se dividirá en 4 fases, con una duración total estimada de 3 meses.



## **11. Conclusiones y Recomendaciones**

### **Lecciones Aprendidas**

* **Pruebas continuas**: Se identificó la importancia de realizar pruebas constantes durante la migración para detectar errores de integración a tiempo y minimizar riesgos operativos. La ejecución de pruebas automatizadas y manuales ayudó a asegurar la estabilidad del nuevo sistema.
* **Capacitación del personal**: La adopción de la nueva arquitectura requirió sesiones de formación para los empleados, lo que permitió reducir la curva de aprendizaje y evitar disrupciones en la operación diaria.
* **Arquitectura modular y escalable**: La implementación de microservicios demostró ser una solución flexible y sostenible, permitiendo actualizaciones más ágiles sin comprometer la operatividad del negocio.

### **Potencial Escalabilidad**

El nuevo sistema basado en microservicios permitirá un crecimiento sostenible para Perfulandia SPA. Al dividir las funcionalidades en servicios independientes, la empresa podrá integrar nuevas herramientas y módulos sin afectar la estabilidad del sistema. Además, se podrán gestionar picos de demanda de manera eficiente mediante la asignación dinámica de recursos a los microservicios que lo requieran.

### **Posibles Mejoras Futuras**

* **Implementación de un sistema de monitoreo avanzado**: Se recomienda la integración de herramientas como Prometheus y Grafana para visualizar el rendimiento del sistema y detectar fallos en tiempo real.
* **Optimización de la interfaz de usuario**: La arquitectura basada en microservicios permitirá mejorar la experiencia del usuario a través de interfaces más dinámicas y adaptables, facilitando el acceso a funcionalidades clave.
* **Automatización de despliegues**: Incorporar herramientas de CI/CD (Integración y Despliegue Continuo) para mejorar la eficiencia en la entrega de nuevas versiones del sistema.

La migración a microservicios no solo ha resuelto los problemas de escalabilidad y rendimiento del sistema anterior, sino que también ha sentado las bases para la innovación y el crecimiento tecnológico de Perfulandia SPA en el futuro.