**Segunda iteración Hyunseda**



Ingeniería de Software II

Desarrollar un sistema de gestión de ventas en línea que permita a los usuarios realizar pedidos de productos, administrar su carrito de compras y gestionar el catálogo de productos de la tienda. El objetivo es proporcionar una experiencia de compra fácil y conveniente para los clientes y contribuir al crecimiento del negocio en línea.

Presentado por:

[Maria Camila Hoyos Gomez](mailto:mariahoyos@unicauca.edu.co)

[Jose Guillermo Segura Casas](mailto:josesegura@unicauca.edu.co)

[Eider Yesid Obando](mailto:eyobando@unicauca.edu.co)

[Yicell Alejandra Troyano Urbano](mailto:ytroyano@unicauca.edu.co)

Profesor:

Wilson Libardo Pantoja Yepez

Universidad del Cauca

Facultad de Electrónica y Telecomunicaciones

Ingeniería de Sistemas

Popayán, Mayo de 2024

**Contenido**

[**1. Planteamiento del problema**](#_heading=h.1fob9te) 1

[1.1. Justificación del Proyecto 1](#_heading=h.3znysh7)

1.2. Objetivos del Proyecto [1](#_heading=h.2lwamvv)

[1.2.1. Objetivos específicos](#_heading=h.3dy6vkm) 1

[**2. Requisitos funcionales**](#_heading=h.2s8eyo1)2

[2.1. Historias Épicas](#_heading=h.17dp8vu) 2

[2.2. Historias de usuario](#_heading=h.3rdcrjn) 2

[2.3.Prototipos](#_heading=h.yq3wc5mtycp3) 2

[**3. Requisitos no funcionales**](#_heading=h.lnxbz9) 3

**4**[**. Arquitectura del sistema**](#_heading=h.35nkun2) 4

[4.1. Arquitectura en Capas](#_heading=h.1ksv4uv) 4

[4.1.1. Capa de Presentación](#_heading=h.44sinio) 4

[4.1.2. Capa de Negocio](#_heading=h.2jxsxqh) 4

[4.1.3. Capa de Acceso a Datos](#_heading=h.z337ya) 4

[4.2. Patrón de Diseño MVC](#_heading=h.3j2qqm3) 4

[4.2.1. Modelo](#_heading=h.1y810tw) 4

[4.2.2. Vista](#_heading=h.4i7ojhp) 4

[4.2.3.Controlador](#_heading=h.2xcytpi) 4

[4.3. Arquitectura de Microservicios](#_heading=h.z67a1764kojw) 4

[4.3.1. Microservicios Implementados](#_heading=h.1qtmd6z3tfs) 4

[4.3.2. Comunicación entre Microservicios](#_heading=h.z58o1dbhqjwg) 4

[4.3.3. Beneficios de la Arquitectura de Microservicios](#_heading=h.qzf5jp1nerqe) 4

[4.3.4. Implementación con Spring Boot](#_heading=h.gxqy3vpimd1z) 4

[4.3.4.1. Creación de Microservicios](#_heading=h.14cunt4t1tii) 4

[4.3.4.2. Comunicación entre Microservicios con Feign](#_heading=h.a0hplou1yr2v) 4

[4.3.4.3. Documentación de APIs con Swagger](#_heading=h.tqtqlr8swy6s) 4

[4.4. Principios SOLID](#_heading=h.hai3l4254elq) 4

[4.5. Estilos de Arquitectura](#_heading=h.xj6t5pepzk8t) 4

[4.6. Patrones de Diseño](#_heading=h.23rpjh6tb8ya) 4

[**5. Metodología**](#_heading=h.2p2csry)5

[5.1. Herramientas de desarrollo](#_heading=h.3o7alnk) 5

[5.2. Diseño del Sistema](#_heading=h.23ckvvd) 5

[5.3. Detalles de implementación](#_heading=h.ihv636) 5

**6**[**. Modelo C4**](#_heading=h.1hmsyys)6

6[.1. Diagrama de contexto](#_heading=h.41mghml) 6

6[.2 Diagrama de Contenedores](#_heading=h.vx1227) 6

6[.3 Diagrama de Componentes](#_heading=h.1v1yuxt) 6

**7**[**. Diagrama de Clases**](#_heading=h.2u6wntf)7

7[.1.](#_heading=h.ihv636) Diagrama de Clases Dominio 5

**8**[**.Repositorio de GitHub**](#_heading=h.nmf14n)9

**Tabla de ilustraciones**

[Ilustración 1. Prototipo del Sistema. Elaboración propia.](#bookmark=id.111kx3o) 2

[Ilustración 2. Prototipo del Sistema. Elaboración propia.](#bookmark=id.111kx3o) 2

[Ilustración 3. Diagrama de contexto. Elaboración propia.](#bookmark=id.111kx3o) 6

[Ilustración 4. Diagrama de contenedores. Elaboración propia.](#bookmark=id.111kx3o) 6

[Ilustración 5. Diagrama de Componentes. Elaboración propia.](#bookmark=id.111kx3o) 6

[Ilustración 6. Diagrama de Clases. Elaboración propia.](#bookmark=id.111kx3o) 7

[Ilustración 7. Diagrama de Clases Dominio .Elaboración propia.](#bookmark=id.111kx3o) 7

**Índice de tablas**

[Tabla 1. Historias Épicas. Elaboración propia. 2](#bookmark=id.3l18frh)

[Tabla 2. Historias de Usuario. Elaboración propia. 2](#bookmark=id.3l18frh)

Introducción

El presente documento aborda el desarrollo de un sistema de gestión de ventas en línea para la empresa Hyun Seda, diseñado para facilitar a los usuarios la realización de pedidos de productos, la administración de su carrito de compras y la gestión del catálogo de productos de una tienda virtual. La empresa Hyun Seda es una compañía consolidada en el mercado de productos artesanales que reconoce la importancia de fortalecer su presencia en el mercado digital. El objetivo principal de este proyecto es proporcionar una experiencia de compra fácil y conveniente para los clientes de Hyun Seda, contribuyendo así al crecimiento del negocio en línea de la empresa.

Capítulo 1

# Planteamiento del problema

Hyun Seda, una empresa consolidada en el mercado de productos artesanales, reconoce la importancia de fortalecer su presencia en el mercado digital. Actualmente, aunque cuenta con una página web existente que permite a los clientes realizar compras, la plataforma carece de ciertas funcionalidades clave y de la modernidad necesaria para satisfacer plenamente las expectativas de los clientes y competir eficazmente en el mercado en línea.

El objetivo de este proyecto es mejorar la plataforma de ventas en línea de Hyun Seda para ofrecer una experiencia de compra más atractiva, conveniente y completa. Esto implica la optimización de las existentes, como la mejora del proceso de compra, la gestión más eficiente del inventario, un seguimiento más detallado de los pedidos y una generación de informes más robusta.

En resumen, el proyecto tiene como objetivo actualizar y optimizar la plataforma de ventas en línea de Hyun Seda para satisfacer las necesidades y expectativas cambiantes de los clientes, así como para mejorar su competitividad en el mercado digital

## Justificación del Proyecto

Ante esta necesidad de adaptación y mejora, es imperativo desarrollar un sistema de gestión de ventas en línea que se ajuste a las particularidades y requerimientos específicos de Hyun Seda. La implementación de una plataforma digital no solo les permitirá ampliar su alcance geográfico y captar nuevos clientes en el mercado en línea, sino que también les brindará la oportunidad de optimizar sus procesos internos y mejorar la eficiencia operativa de la empresa. Además, ofrecer una experiencia de compra satisfactoria y segura a través de internet contribuirá a fortalecer la relación con los clientes existentes y atraer a nuevos consumidores interesados en los productos de Hyun Seda.

## Objetivos del Proyecto

Desarrollar un sistema de gestión de ventas en línea personalizado para Hyun Seda, que les permita expandir su presencia en el mercado en línea y mejorar la experiencia de compra de sus clientes.

#### Objetivos Específicos

* Diseñar e implementar una interfaz de usuario intuitiva y atractiva que refleje la identidad de la marca y facilite la navegación de los usuarios.
* Desarrollar funcionalidades para la gestión integral de pedidos, incluyendo la recepción, procesamiento y seguimiento de los mismos.
* Integrar un catálogo digitalizado de productos que permita a los clientes explorar el inventario de manera fácil y rápida, con opciones de búsqueda y filtrado avanzadas.
* Mejorar la disponibilidad y rendimiento del sistema, asegurando tiempos de respuesta rápidos y un funcionamiento ininterrumpido durante las horas de mayor demanda.

**Capítulo 2**

# 2. Requisitos funcionales

Un requisito funcional es una declaración de cómo debe comportarse un sistema. Define lo que el sistema debe hacer para satisfacer las necesidades o expectativas del usuario. Los requisitos funcionales se pueden considerar como características que el usuario detecta. Son diferentes de los requisitos no funcionales, que definen cómo debe funcionar internamente el sistema

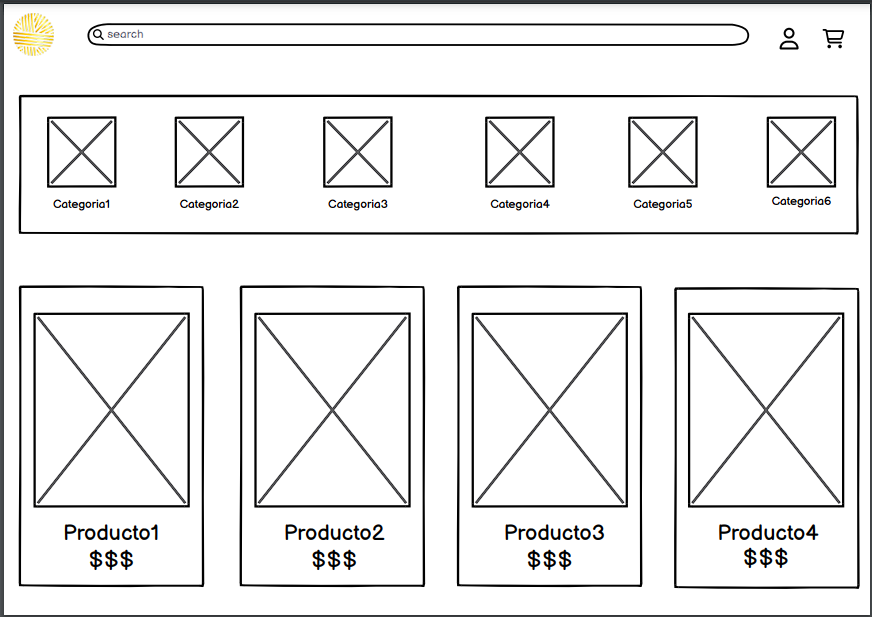
## 2.1. Historias Épicas

[**HistoriasEpicas**](https://docs.google.com/spreadsheets/u/0/d/1Zn9bsSx7eAeilC15REzf2Q9giALNRyLaxSAkej18kLw/edit)

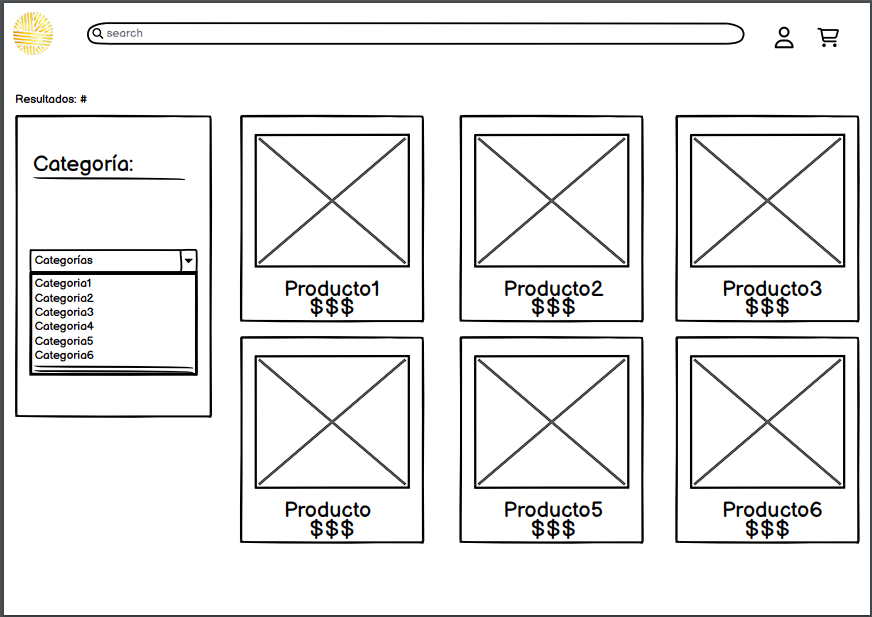
## 2.2. Historias de usuario

[**HistoriasUsuario**](https://docs.google.com/spreadsheets/u/0/d/1oLz2MHG8f-P4wrxAKdQlHT7bh6MhzyOPtvrheS23mPQ/edit)

## 2.3. Prototipos:



[Ilustración 1. Prototipo sistema. Elaboración propia.](#bookmark=id.111kx3o)



[Ilustración 2. Prototipo sistema. Elaboración propia.](#bookmark=id.111kx3o)

Capítulo 3

# 3. Requisitos no funcionales

**Rendimiento:**

* Escenario: Durante un evento de ventas especiales, como el Black Friday, se registra un pico de tráfico en el sitio web.
* Estímulo: Aumento repentino en el número de usuarios y solicitudes de compra.
* Respuesta: El sistema debe mantener tiempos de respuesta rápidos y estables, con una capacidad para manejar al menos el doble de la carga habitual sin degradación del rendimiento.

**Seguridad:**

* Escenario: Un hacker intenta acceder a la base de datos del sistema para robar información de clientes.
* Estímulo: Intento de acceso no autorizado a través de técnicas de piratería.
* Respuesta: El sistema debe resistir intentos de intrusión mediante medidas como el cifrado de datos, la autenticación de dos factores y la detección de intrusiones. Debe registrar y notificar cualquier intento de acceso no autorizado para su análisis posterior.

**Disponibilidad:**

* Escenario: Durante el horario comercial, un cliente intenta acceder al sitio web para realizar una compra.
* Estímulo: Acceso al sitio web a través de un navegador web estándar.
* Respuesta: El sitio web debe estar disponible y accesible las 24 horas del día, los 7 días de la semana, con un tiempo de inactividad planificado mínimo para mantenimiento programado. El tiempo de respuesta para cargar la página principal no debe exceder los 3 segundos en condiciones normales de tráfico.

**Escalabilidad:**

* Escenario: Hyun Seda lanza una nueva línea de productos que genera un aumento en el tráfico del sitio web.
* Estímulo: Incremento en el número de usuarios y solicitudes de compra.
* Respuesta: El sistema debe poder escalar horizontalmente para manejar el aumento en la carga sin afectar el rendimiento. Se deben implementar tecnologías y prácticas que permitan agregar recursos adicionales según sea necesario.

**Usabilidad:**

* Escenario: Un cliente navega por el sitio web en busca de un producto específico.
* Estímulo: Interacción con la interfaz de usuario del sitio web.
* Respuesta: La interfaz de usuario debe ser intuitiva y fácil de usar, con un diseño limpio y una navegación clara. Se deben minimizar los clics y los tiempos de carga para mejorar la experiencia del usuario.

**Mantenibilidad**

* Escenario: Un desarrollador necesita corregir un error en el sistema.
* Estímulo: Identificación de un problema durante pruebas de calidad o revisiones de código.
* Respuesta: El código del sistema debe estar bien estructurado, documentado y modularizado para facilitar la identificación y corrección de errores. Se deben seguir prácticas de desarrollo ágil y utilizar herramientas de gestión de versiones para mantener un código limpio y organizado.

**Compatibilidad:**

* Escenario: Un cliente intenta acceder al sitio web desde su dispositivo móvil.
* Estímulo: Acceso al sitio web a través de un navegador móvil en un dispositivo Android.
* Respuesta: El sitio web debe ser compatible con una amplia gama de dispositivos y navegadores, incluidos dispositivos móviles, tabletas y computadoras de escritorio. Debe ajustarse automáticamente al tamaño de pantalla y mantener una funcionalidad completa en todos los dispositivos compatibles.

**Legal y normativo:**

* Escenario: Un cliente realiza una compra y proporciona información personal y financiera.
* Estímulo: Ingreso de datos personales y de pago durante el proceso de compra.

- Respuesta: El sistema debe cumplir con todas las leyes y regulaciones aplicables en cuanto a privacidad de datos, protección del consumidor y seguridad de la información. Debe seguir estándares de seguridad de la industria y utilizar prácticas recomendadas para proteger la información del cliente y garantizar transacciones seguras.

**Escalabilidad**

* Uno de los principales beneficios de adoptar una arquitectura de microservicios es la capacidad de escalar los diferentes componentes del sistema de forma independiente según las necesidades de carga. En el sistema de gestión de ventas en línea de Hyun Seda, se implementaron estrategias para permitir la escalabilidad horizontal y vertical de los microservicios.

**Escalabilidad Horizontal**

* La escalabilidad horizontal implica agregar más instancias de un microservicio para manejar un mayor volumen de solicitudes. Esto se logró mediante el uso de Docker y Kubernetes:
* Cada microservicio se empaqueta como una imagen de Docker, lo que facilitó su implementación como contenedores ligeros e independientes.
* Kubernetes se utilizó para orquestar y gestionar los contenedores de los microservicios, permitiendo escalar horizontalmente cada microservicio de forma dinámica según la demanda.
* Se configuraron reglas de escalado automático en Kubernetes, que monitorean métricas como el uso de CPU y memoria, y ajustaban automáticamente el número de réplicas de cada microservicio según fuera necesario.

**Escalabilidad Vertical**

* Además de la escalabilidad horizontal, también se implementó la escalabilidad vertical, que implica aumentar los recursos (CPU, memoria, etc.) asignados a cada instancia de un microservicio.
* Se utilizaron instancias de computación escalables en la nube, lo que permitió ajustar fácilmente los recursos asignados a cada microservicio según sus necesidades.
* Se monitorea continuamente el rendimiento y la utilización de recursos de cada microservicio, y se ajustaron los recursos verticalmente cuando se detectaron cuellos de botella o saturación de recursos.

**Equilibrio de Carga**

* Para distribuir eficientemente el tráfico entrante entre las múltiples instancias de cada microservicio, se implementó un equilibrador de carga:
* Se utilizó un equilibrador de carga basado en software, como NGINX o HAProxy, que se encargó de enrutar las solicitudes entrantes a las instancias de microservicios disponibles.
* El equilibrador de carga aplicó algoritmos de balanceo de carga, como round-robin o least-connections, para distribuir equitativamente el tráfico entre las instancias.
* Se configuraron comprobaciones de estado para monitorear la salud de cada instancia de microservicio y retirar automáticamente las instancias no saludables del grupo de balanceo de carga.

Capítulo 4

# 4. Arquitectura del sistema

El sistema de gestión de ventas en línea para Hyun Seda se ha diseñado siguiendo una arquitectura en capas, el patrón Modelo-Vista-Controlador (MVC) y la arquitectura de microservicios para garantizar la modularidad, la separación de responsabilidades y la escalabilidad del sistema.

## 4.1. Arquitectura en Capas

El proyecto se desarrolló utilizando una arquitectura en capas que permite una clara separación de responsabilidades y facilita la modularidad, el mantenimiento y la evolución del sistema. A continuación se detalla cómo se utilizaron estas capas:

### 4.1.1. Capa de presentación

* La capa de presentación se implementó utilizando el framework de desarrollo de aplicaciones web Spring MVC en Java.
* En esta capa, se crearon vistas dinámicas para la interfaz de usuario del sistema.
* Spring MVC se encargó de manejar las solicitudes HTTP, enrutando las peticiones a los controladores adecuados y actualizando la vista según los datos proporcionados por el modelo.

### 4.1.2. Capa de Negocio

* La capa de negocio se implementa utilizando clases y componentes Java que definen la lógica de negocio del sistema.
* Se crearon modelos de dominio, como Productos, Pedidos y Clientes, junto con las reglas de negocio asociadas a estas entidades.
* Se utilizaron servicios de negocio para coordinar las operaciones complejas, como la gestión de inventario y la validación de pedidos.

### 4.1.3. Capa de Acceso a Datos

* La capa de acceso a datos se implementó utilizando el framework Spring Data JPA, que simplifica el acceso a la base de datos y reduce la cantidad de código boilerplate.
* Se configuró una base de datos SQLite en memoria para el desarrollo y pruebas del sistema.

## 4.2. Patrón de Diseño MVC

El proyecto siguió el patrón de diseño Modelo-Vista-Controlador (MVC) para separar las responsabilidades de presentación, lógica de negocio y acceso a datos.

### 4.2.1. Modelo

* El modelo consistió en las clases de dominio Java que representaban las entidades del sistema, como Productos, Pedidos y Clientes.
* Se implementaron métodos de negocio dentro de estas clases para realizar operaciones relacionadas con cada entidad.

### 4.2.2. Vista

* La vista se creó utilizando el framework JavaFX para el desarrollo de interfaces de usuario modernas y enriquecidas.
* Se diseñaron y desarrollaron ventanas, paneles, botones y otros componentes de interfaz de usuario mediante el lenguaje JavaFX y sus herramientas de diseño visual.
* JavaFX permitió crear interfaces de usuario atractivas y altamente interactivas, con efectos visuales, animaciones y transiciones suaves.
* Además, JavaFX facilitó la integración de multimedia, como imágenes lo que enriqueció la experiencia del usuario en el sistema de gestión de ventas en línea.

### 4.2.3. Controlador

Los controladores de Spring MVC se encargaron de recibir las solicitudes del usuario, invocar los métodos apropiados en la capa de negocio y actualizar la vista con los resultados.

## 4.3. Arquitectura de Microservicios

Además de la arquitectura en capas, el patrón Modelo-Vista-Controlador (MVC) y la arquitectura microkernel descritas anteriormente, el sistema de gestión de ventas en línea de Hyun Seda también adopta una arquitectura de microservicios. Esto permite una mayor modularidad, escalabilidad y flexibilidad en el desarrollo y mantenimiento del sistema.

### 4.3.1 Microservicios Implementados

El sistema se compone de los siguientes microservicios, cada uno enfocado en un dominio específico del negocio:

1. Microservicio Usuario: Encargado de la gestión de usuarios y roles. Contiene las entidades User y Role.
2. Microservicio Cliente: Maneja la información de los clientes, como nombre, apellido, dirección, etc. Contiene la entidad Client.
3. Microservicio Producto: Gestiona el catálogo de productos, incluyendo detalles como id, nombre, stock y categoría asociada. Contiene las entidades Product y Category.
4. Microservicio Pedido: Responsable del proceso de realización de pedidos. Contiene las entidades Order e Ítem, donde Ítem representa los productos individuales dentro de un pedido.

### 4.3.2 Comunicación entre Microservicios

Estos microservicios se comunican entre sí a través de interfaces bien definidas, utilizando protocolos de comunicación como HTTP/REST o mensajería asíncrona. Por ejemplo, el microservicio Pedido interactúa con los microservicios Cliente y Producto para obtener la información necesaria al procesar un nuevo pedido.

### 4.3.3 Beneficios de la Arquitectura de Microservicios

La adopción de una arquitectura de microservicios en el sistema de gestión de ventas en línea de Hyun Seda ofrece varios beneficios:

* Modularidad: Cada microservicio se enfoca en una funcionalidad específica, lo que facilita el desarrollo, pruebas y mantenimiento de cada componente de forma independiente.
* Escalabilidad: Los microservicios pueden escalarse de forma individual según las necesidades de carga, sin afectar a otros componentes del sistema.
* Flexibilidad: Debido a su naturaleza desacoplada, los microservicios permiten la incorporación de nuevas tecnologías y la actualización de componentes individuales sin interrumpir el funcionamiento del sistema completo.
* Resiliencia: Si un microservicio falla, el resto del sistema puede continuar funcionando, lo que mejora la tolerancia a fallos y la disponibilidad general del sistema.

### 4.3.4 Implementación con Spring Boot

Para la implementación de la arquitectura de microservicios en el sistema de gestión de ventas en línea de Hyun Seda, se utilizó el framework Spring Boot como tecnología principal. Spring Boot facilita el desarrollo, prueba y despliegue de aplicaciones basadas en Spring, lo que lo convierte en una excelente opción para construir microservicios.

#### 4.3.4.1 Creación de Microservicios

Cada microservicio se construyó como una aplicación Spring Boot independiente, siguiendo los siguientes pasos:

* Generar la estructura base del proyecto utilizando Spring Initializr.
* Definir las dependencias necesarias, como Spring Web, Spring Data JPA, etc.
* Crear las entidades de dominio y los repositorios JPA para interactuar con la base de datos.
* Implementar los controladores REST que expongan las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para las entidades correspondientes.

#### 4.3.4.2 Comunicación entre Microservicios con Feign

Para permitir la comunicación entre los microservicios, se utilizó la biblioteca Feign, que es una herramienta de cliente HTTP declarativa desarrollada por Netflix. Feign se integra de manera sencilla con Spring Boot y permite definir interfaces de cliente que actúan como proxies de los servicios remotos.

En el código de cada microservicio, se crearon interfaces Feign que definían los métodos de los servicios que se desea consumir. Estas interfaces se anotaron con @FeignClient y se especificó el nombre del microservicio al que se accedería. Luego, estas interfaces se inyectaron en los componentes correspondientes, lo que permitió invocar los métodos de otros microservicios como si fueran locales.

#### 4.3.4.3 Documentación de APIs con Swagger

Para documentar las APIs REST de los microservicios, se utilizó la herramienta Swagger. Swagger genera documentación interactiva a partir de los comentarios en el código fuente, lo que elimina la necesidad de escribir documentación manualmente.

Se agregaron las dependencias necesarias de Swagger en cada proyecto de microservicio y se configuró mediante la anotación @EnableSwagger2. Luego, se documentaron las APIs utilizando anotaciones específicas de Swagger, como @Api, @ApiOperation y @ApiParam, para describir el propósito de los controladores, métodos y parámetros.

Una vez en ejecución, se pudo acceder a la interfaz de usuario de Swagger visitando la ruta /swagger-ui.html en cada microservicio, lo que permitió explorar y probar las APIs de manera interactiva.

## 4.4. Principios SOLID

En el diseño y desarrollo del sistema de gestión de ventas en línea de Hyun Seda, se aplican los principios SOLID (Responsabilidad Única, Abierto/Cerrado, Sustitución de Liskov, Segregación de Interfaces y Principio de Inversión de Dependencias) para promover una arquitectura modular, escalable y mantenible. Estos principios fueron especialmente relevantes en la implementación de la arquitectura de microservicios:

* Responsabilidad Única : Cada microservicio se enfocó en una responsabilidad específica del negocio, como la gestión de usuarios, productos o pedidos, lo que facilitó su desarrollo, pruebas y mantenimiento.
* Abierto/Cerrado : Los microservicios se diseñan para ser abiertos a extensiones pero cerrados a modificaciones, lo que permitió agregar nuevas funcionalidades sin alterar el código existente.
* Sustitución de Liskov : Las interfaces y abstracciones utilizadas en los microservicios se diseñan de manera que las implementaciones derivadas podrían sustituirse sin afectar el comportamiento del sistema.
* Segregación de Interfaces : Los microservicios expusieron interfaces específicas y coherentes, evitando depender de interfaces monolíticas y favoreciendo la reutilización de código.
* Inversión de Dependencias : Los microservicios dependían de abstracciones, no de implementaciones concretas, lo que facilitó la inyección de dependencias y el acoplamiento flexible entre los componentes.

## 4.5. Estilos de Arquitectura

El sistema de gestión de ventas en línea de Hyun Seda adoptó una arquitectura de microservicios y una arquitectura hexagonal. Estos estilos arquitectónicos se eligieron por las siguientes razones:

* Microservicios : Este estilo permite dividir la funcionalidad del sistema en servicios independientes y cohesivos, lo que favoreció la modularidad, escalabilidad, flexibilidad y resiliencia del sistema. Además, facilitó el cumplimiento de requisitos no funcionales como el rendimiento, la seguridad y la disponibilidad, al poder escalar y actualizar los microservicios de forma individual.
* Arquitectura Hexagonal : Este estilo arquitectónico promovió la separación de la lógica de negocio de la tecnología utilizada, lo que facilitó la modificabilidad y mantenibilidad del sistema a largo plazo. Además, al aislar el dominio del sistema, se cumplieron requisitos funcionales relacionados con la gestión de productos, pedidos y usuarios de manera más eficiente.

## 4.6. Patrones de Diseño

En el desarrollo del sistema, se aplicaron varios patrones de diseño tanto a nivel arquitectónico como a nivel de código, con el objetivo de favorecer la modificabilidad y facilitar la construcción de objetos complejos. El patrón más utilizado fue el Builder:

* Patrón Builder: Este patrón se utilizó ampliamente en la construcción de objetos complejos como Pedidos y Productos. Permitió separar la construcción de estos objetos de su representación, lo que facilitó la creación de diferentes representaciones del mismo objeto. Además, el patrón Builder promovió una mayor modularidad y reutilización del código, ya que los pasos de construcción se encapsularon en clases separadas (BuilderPedido, BuilderProducto, etc.). Esto simplificó la creación y modificación de objetos complejos, lo que resultó fundamental para cumplir con los requisitos funcionales relacionados con la gestión de pedidos y productos.

Otros patrones aplicados fueron:

* Patrones de Comportamiento: Se utilizaron patrones como State y Observer para manejar el estado y las notificaciones de los objetos de manera eficiente.
* Patrón Estructural: El patrón Facade se aplicó para proporcionar una interfaz unificada y simplificada para un conjunto de interfaces más complejas, lo que mejoró la usabilidad y la modularidad del sistema.
* Patrón Creacional: Además del Builder, se utilizó el patrón Factory para crear objetos de manera centralizada y encapsulada.

Capítulo 5

# 5. Metodología

## 5.1. Herramientas de desarrollo

Las principales herramientas utilizadas durante el desarrollo del proyecto fueron:

* Entorno de Desarrollo Integrado (IDE): Se utilizó IntelliJ IDEA como entorno de desarrollo principal debido a su robustez y amplio soporte para proyectos Java.
* Gestor de Dependencias: Apache Maven se utilizó para la gestión de dependencias y construcción del proyecto.
* Base de Datos: Se optó por SQLite como base de datos para simplificar el despliegue y la configuración del entorno de desarrollo.

## 5.2. Diseño del Sistema

* Controladores (Controllers): Responsables de manejar las solicitudes del usuario y coordinar las acciones correspondientes en el modelo y la vista.
* Servicios (Services): Contienen la lógica de negocio de la aplicación y coordinan las operaciones entre los controladores y los repositorios de datos.
* Repositorios (Repositories): Encargados de interactuar con la base de datos para realizar operaciones de lectura y escritura.

## 5.3. Detalles de implementación

Durante la implementación del sistema, se prestaron especial atención a los siguientes aspectos:

* Configuración de la capa de seguridad para proteger las rutas y recursos sensibles de la aplicación.
* Validación de datos de entrada para garantizar la integridad y consistencia de los datos almacenados en la base de datos.

Capítulo 6

# 6. Modelo C4

Es un marco diseñado para ilustrar las complejas estructuras e interacciones dentro de los sistemas de software. En su núcleo, el modelo C4 ayuda a los equipos a comprender su arquitectura de software a través de una serie de diagramas que representan diferentes niveles de abstracción.

## 6.1. Diagrama de contexto

## 

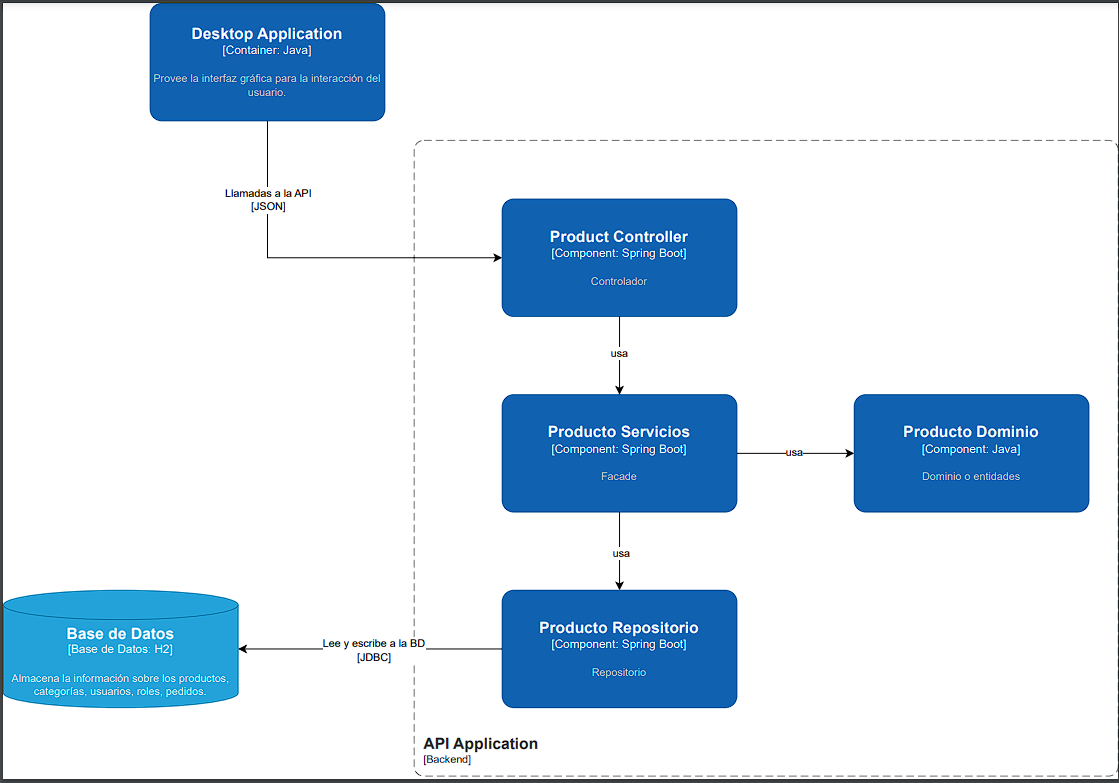
[Ilustración 3. Diagrama de contexto. Elaboración propia.](#bookmark=id.111kx3o)

## 6.2 Diagrama de Contenedores

## 

[Ilustración 4. Diagrama de contenedores. Elaboración propia.](#bookmark=id.111kx3o)

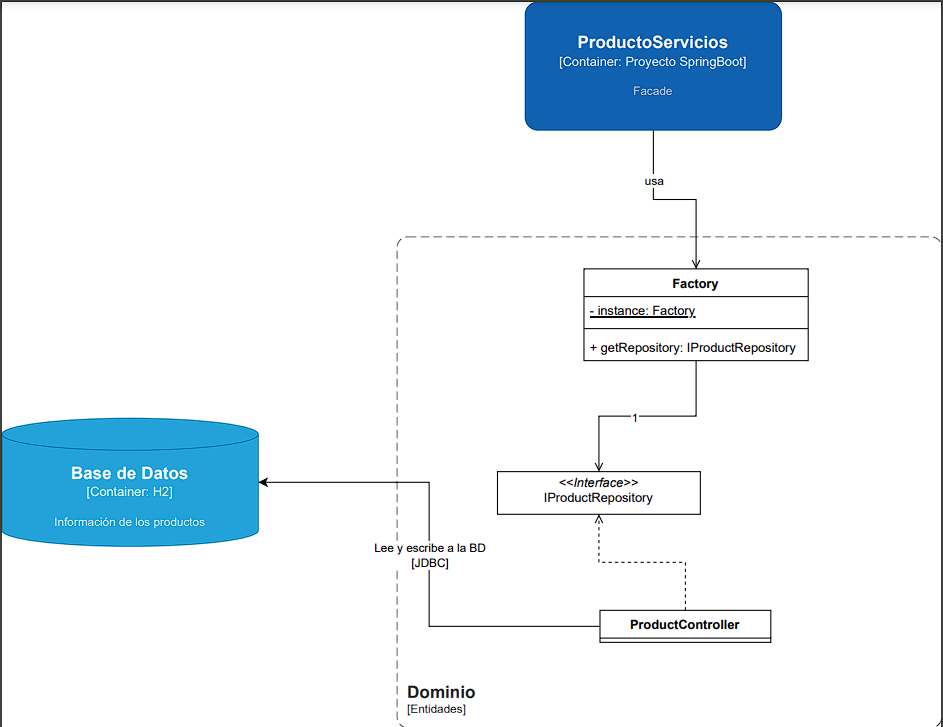
## 6.3 Diagrama de Componentes



[Ilustración 5. Diagrama de Componentes. Elaboración propia.](#bookmark=id.111kx3o)

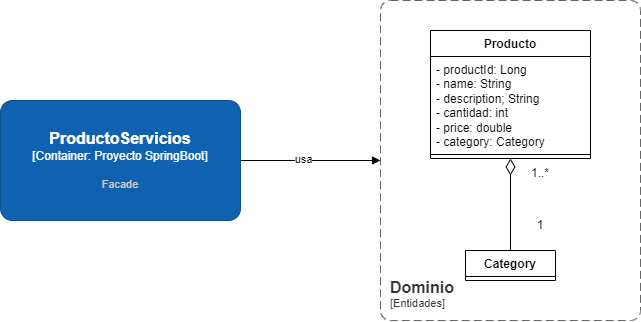
Capítulo 7

# 7. Diagrama de Clases



[Ilustración 6. Diagrama de Clases. Elaboración propia.](#bookmark=id.111kx3o)

## 7.1. Diagrama de Clases Dominio



[Ilustración 7. Diagrama de Clases. Elaboración propia.](#bookmark=id.111kx3o)

Capítulo 8

# 8.Repositorio de GitHub

<https://github.com/Yesid812/HyunSeda>

Referencias

[**https://visuresolutions.com/es/blog/functional-requirements/**](https://visuresolutions.com/es/blog/functional-requirements/)

<https://miro.com/es/diagrama/que-es-modelo-c4/>

El sistema abre un formulario rellenado

con los detalles de la publicación que

el administrador puede modificar.