Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Arquitectura basada en Microservicios
Implementación de un sistema cloud políglota para una franquicia de tiendas de ropa

Autor:
Jorge Aguilera Gómez

Tutor:
Dr. Miguel A. Laguna

Resumen

El propósito de este Trabajo de Fin de Grado es realizar un proyecto de ingeniería de software y su correspondiente implementación de una aplicación empresarial que gestione las diferentes partes del negocio de una franquicia de tiendas de ropa. Se necesita una aplicación que ofrezca una alta escalabilidad, una alta disponibilidad y que sea resistente a fallos.

Con esta finalidad, se ha diseñado e implementado una aplicación cloud políglota basada en una arquitectura de microservicios, siguiendo las buenas prácticas que exige este tipo de arquitectura.

Abstract

The purpose of this end-of-degree project is to make a Software Engineering project and its corresponding implementation of a enterprise app that manages the different parts of a clothing store franchise business. An app that offers high scalability, high availabiliy and resiliency is required.

With this finality, it has been designed and implemented a microservice-based architecture cloud polyglot app, following good practices that the architecture type required.

AGRADECIMIENTOS

Quiero agradecer a mis padres por apoyarme en los momentos más difíciles.

Jorge Aguilera Gómez
Valladolid, 2021

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

CAPÍTULO 1

INTRODUCCIÓN

1.1. CONTEXTO

La franquicia de ropa ficticia *Clothing My World* está siendo un éxito en el mercado de la moda en los últimos meses. Socios e inversores prevén un incremento en el negocio enorme, por lo que deciden invertir en un software que permita gestionar la totalidad del negocio de forma eficiente y que pueda escalar al mismo ritmo que lo hará el negocio. Este software tendrá un papel fundamental en el funcionamiento del negocio y se estima que la cantidad de clientes y empleados será muy elevada, así que se necesitará una aplicación con una alta disponibilidad y resistente a fallos, que evite grandes pérdidas económicas por caídas del sistema.

Se desea que el software ofrezca servicios de tienda online, gestión de productos, stocks, tiendas, almacenes, ventas físicas, ventas online, repartos, proveedores...

Con el fin de satisfacer las necesidades requeridas por el negocio, se ha diseñado e implementado una aplicación cloud (en la nube) políglota basada en una arquitectura de microservicios. Este tipo de arquitectura ha sido bastante utilizada con éxito en los últimos años para diseñar aplicaciones con este tipo de requisitos, debido a que ofrece un sistema resistente, con una alta disponibilidad y con una gran escalabilidad.

1.2. MOTIVACIÓN

Mi interés por el diseño e implementación de las arquitecturas de microservicios surgió durante mis prácticas de empresa extracurriculares. En esta etapa de becario trabajé como desarrollador en varios de los microservicios de una aplicación que estaba basada en esta arquitectura.

La curiosidad por conocer los conceptos más avanzados de esta arquitectura me empujó a investigar de forma autodidacta con el objetivo de ampliar mis conocimientos.

A medida que leía artículos y libros sobre las arquitecturas de microservicios mi interés crecía. Fue así como me surgió la idea de que mi TFG sería una buena oportunidad para poner en práctica y consolidar todos estos conocimientos estudiados sobre las arquitecturas de microservicios. La motivación por aprender sobre este tipo de arquitectura me llevó a tomar la decisión de hacer una propuesta de TFG sobre esta temática.

Proponer un TFG es algo bastante más duro de lo que pensé en un principio, sobre todo cuando buena parte de lo que va a tratar tu trabajo es nuevo para ti. Definir el alcance sin meterse en cosas demasiado complejas, dejarse llevar por la curiosidad pero no demasiado, abortar decisiones que pueden llevar demasiado tiempo o incluso no ser capaz de llevarlas acabo por exigir demasiado nivel.

La parte positiva de realizar un TFG propuesto por mí ha sido la facilidad para echar muchísimas horas de insistencia hasta que las cosas salían, a la vez que me hallaba envuelto en un enorme interés.

1.3. FINALIDAD, ALCANCE Y OBJETIVOS

El objetivo de este proyecto es diseñar e implementar una aplicación cloud políglota basada en una arquitectura de microservicios para dar una solución tecnológica a las distintas partes de negocio de una franquicia de tiendas de ropa. Una aplicación basada en arquitectura de microservicios ofrece una alta disponibilidad, resistencia a fallos y escalabilidad.

Para conseguir este objetivo, se realizará un estudio de todos los patrones de diseño recomendados para las diferentes situaciones que se pueden dar en una arquitectura de este tipo. Parte de estos patrones de diseño, serán aplicados en el diseño e implementación de la aplicación de la que trata el proyecto presentado en esta memoria.

Esta aplicación tendrá un diseño back-end políglota en la que una parte de los microservicios estará desarrollada en ASP.Net Core y la otra parte en Spring Boot. El despliegue se realizará en contenedores docker en el proveedor de servicios de computación en la nube Google Cloud mediante la herramienta Google Kubernetes Engine (GKE).

Todo esto estará contenido dentro del proyecto de ingeniería de software que se presenta en esta memoria.

1.4. ESTRUCTURA DEL DOCUMENTO

Este documento ha sido estructurado de la siguiente forma:

1. Un primer capítulo introductorio en el que se presentan las motivaciones y los objetivos.

2. Un segundo capítulo en el que se presenta el plan de proyecto para conseguir los objetivos, el plan de control y el seguimiento del proyecto.

3. Un tercer capítulo en el que se presenta la parte de análisis y elicitación de los requisitos que debe de cumplir el sistema

4. Un cuarto capítulo en el que se presenta el diseño de la arquitectura del sistema y se detallan los patrones de diseño utilizados.

5. Un quinto capítulo en el que se presenta el diseño de cada uno de los microservicios que componen el sistema. Se detalla para cada microservicio, el diseño de su arquitectura, el diseño de su base de datos y el diseño de su API REST.

6. Un sexto capítulo en el que se presenta la implementación del la aplicación y se destacan las partes de la implementación más características de la arquitectura.

7. Un séptimo capítulo en el que se presentan las pruebas de los end-points y las pruebas de integración que se han realizado para verificar que se han alcanzado los objetivos.

8. Un octavo capítulo en el que se presenta el despliegue del sistema y los pasos necesarios para instalar la aplicación, tanto en un entorno de desarrollo cómo en un entorno de producción.

9. Un noveno capítulo en el que se presentan las conclusiones obtenidas al finalizar el proyecto y el trabajo futuro.

10. Un Anexo A que contiene el manual de usuario de la aplicación.

1.5. REPOSITORIO DEL PROYECTO

Se ha utilizado GitHub como gestor de versiones del código de la aplicación. El código esta dividido en varios repositorios, estos repositorios han sido agrupados en una organización GitHub, a la que se accede en el siguiente enlace:

https://github.com/ClothingStoreFranchise

CAPÍTULO 2

PLAN DE PROYECTO

2.1. METODOLOGÍA

Para desarrollar el proyecto se ha seguido la metodología de desarrollo de software Proceso Unificado de Desarrollo (RUP). Se ha considerado que esta metodología es la más conveniente para realizar el proyecto presentado en esta memoria, al tener un ciclo de vida iterativo, incremental y centrado a la arquitectura, además de adaptarse bien al calendario de la universidad.

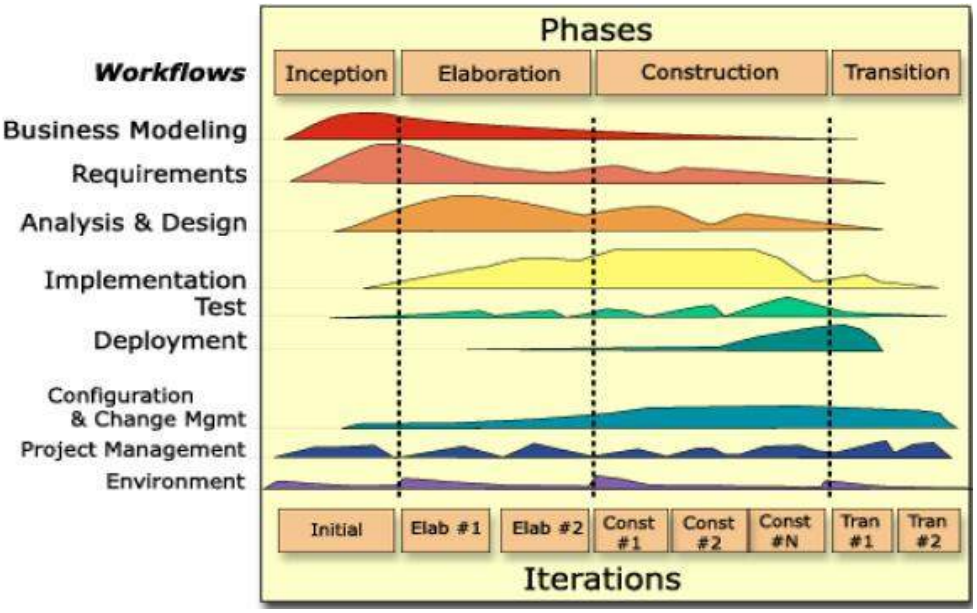

Figura 2.1: Fases del RUP. Tomada de https://www.ecured.cu/Proceso_unificado_de_desarrollo.

2.2. PLAN DE ITERACIONES

2.2.1. Fase de Inicio

Tabla 2.1: Plan de iteraciones: Fase de inicio - Iteración 1

Nombre de la tarea	Duración estimada (horas)	Comienzo estimado	Final estimado
FASE DE INICIO - ITERACIÓN 1	**189**	**02/03/2020**	**01/05/2020**
Comienzo de la iteración 1	0	02/03/2020	02/03/2020
Planteamiento del proyecto	30	02/03/2020	17/04/2020
Adquirir documentación y contrastar información	20	02/03/2020	27/03/2020
Estudio de las arquitecturas de microservicios	80	02/03/2020	04/05/2020
Estudio tecnologías implicadas	40	02/03/2020	01/05/2020
Plan de fases	4	01/04/2020	01/04/2020
Plan del proyecto	8	20/04/2020	27/04/2020
Plan de riesgos	6	27/04/2020	01/05/2020
Plan de iteración siguiente	1	01/05/2020	01/05/2020
Fin de la fase de inicio	0	01/05/2020	01/05/2020

Tabla 2.2: Plan de iteraciones: Fase de inicio - Iteración 2

Nombre de la tarea	Duración estimada (horas)	Comienzo estimado	Final estimado
FASE DE INICIO - ITERACIÓN 2	**34**	**01/05/2020**	**22/05/2020**
Comienzo de la iteración 2	0	01/05/2020	01/05/2020
Análisis de requisitos iniciales	6	01/05/2020	06/05/2020
Estudio entorno cloud	12	06/05/2020	12/05/2020
Preparar entorno de desarrollo	12	06/05/2020	22/05/2020
Documento costes	1	22/05/2020	22/05/2020
Documento seguimiento	2	22/05/2020	22/05/2020
Plan de iteración siguiente	1	22/05/2020	22/05/2020
Fin de la fase de inicio	0	22/05/2020	22/05/2020

2.2.2. Fase de Elaboración

Tabla 2.3: Plan de iteraciones: Fase de elaboración - Iteración 1

Nombre de la tarea	Duración estimada (horas)	Comienzo estimado	Final estimado
FASE DE ELABORACIÓN - ITERACIÓN 1	**46**	**22/05/2020**	**26/06/2020**
Comienzo de la iteración 1	0	22/05/2020	22/05/2020
Elicitación de requisitos	8	22/05/2020	25/05/2020
Diagrama de casos de uso	10	25/05/2020	29/05/2020
Descripción de casos de uso	12	29/05/2020	05/06/2020
Modelo de dominio en análisis	15	01/06/2020	12/06/2020
Plan de iteración siguiente	1	12/06/2020	12/06/2020

Tabla 2.4: Plan de iteraciones: Fase de elaboración - Iteración 2

Nombre de la tarea	Duración estimada (horas)	Comienzo estimado	Final estimado
FASE DE ELABORACIÓN - ITERACIÓN 2	**63**	**12/06/2020**	**09/07/2020**
Comienzo de la iteración 2	0	12/06/2020	12/06/2020
Revisión requisitos	3	12/06/2020	12/06/2020
Revisión casos de uso	4	15/06/2020	15/06/2020
Aplicar Domain Driven Desing	15	16/06/2020	22/06/2020
Diseño de las bases de datos	10	22/06/2020	24/06/2020
Implementación del modelo y del acceso a persistencia	30	24/06/2020	09/07/2020
Plan de iteración siguiente	1	09/07/2020	09/07/2020

Tabla 2.5: Plan de iteraciones: Fase de elaboración - Iteración 3

Nombre de la tarea	Duración estimada (horas)	Comienzo estimado	Final estimado
FASE DE ELABORACIÓN - ITERACIÓN 3	**131**	**09/07/2020**	**31/07/2020**
Comienzo de la iteración 3	0	09/07/2020	09/07/2020
Diseño de la API Gateway	15	09/07/2020	16/07/2020
Diseño de la autenticación del sistema	10	09/07/2020	16/07/2020
Implementación de la API Gateway	25	16/07/2020	22/07/2020
Diseño de la arquitectura de los microservicios	20	22/07/2020	24/07/2020
Implementación de la lógica de los microservicios	60	24/07/2020	31/07/2020
Plan de iteración siguiente	1	31/07/2020	31/07/2020

2.2.3. Fase de Construcción

Tabla 2.6: Plan de iteraciones: Fase de construcción - Iteración 1

Nombre de la tarea	Duración estimada (horas)	Comienzo estimado	Final estimado
FASE DE CONSTRUCCIÓN - ITERACIÓN 1	**221**	**31/07/2020**	**21/09/2020**
Comienzo de la iteración 1	0	31/07/2020	31/07/2020
Diseño de la API de los microservicios	20	31/07/2020	06/08/2020
Implementación de las APIs de los microservicios	50	06/08/2020	21/08/2020
Diseño de la comunicación entre los microservicios	40	21/08/2020	01/09/2020
Implementación de la comunicación entre microservicios	100	01/09/2020	21/09/2020
Pruebas de los elementos desarrollados en esta fase	10	15/09/2020	21/09/2020
Plan de iteración siguiente	1	21/09/2020	21/09/2020

Tabla 2.7: Plan de iteraciones: Fase de construcción - Iteración 2

Nombre de la tarea	Duración estimada (horas)	Comienzo estimado	Final estimado
FASE DE CONSTRUCCIÓN - ITERACIÓN 2	**131**	**21/09/2020**	**23/10/2020**
Comienzo de la iteración 2	0	21/09/2020	21/09/2020
Diseño de la arquitectura front end	15	21/09/2020	25/09/2020
Implementación del frontend	80	25/09/2020	23/10/2020
Completar la implementación de la lógica de negocio	15	10/10/2020	23/10/2020
Corrección de bugs graves	20	10/10/2020	23/10/2020
Plan de iteración siguiente	1	23/10/2020	23/10/2020

2.2.4. Fase de Transición

Tabla 2.8: Plan de iteraciones: Fase de transición - Iteración 1

Nombre de la tarea	Duración estimada (horas)	Comienzo estimado	Final estimado
FASE DE TRANSICIÓN - ITERACIÓN 1	**60**	**23/10/2020**	**27/11/2020**
Comienzo de la iteración 1	0	23/10/2020	23/10/2020
Correcciones leves y mejoras	25	23/10/2020	16/11/2020
Pruebas	20	16/11/2020	24/11/2020
Despliegue	10	24/11/2020	26/11/2020
Manuales	5	26/11/2020	27/11/2020

2.3. COSTES DEL PLAN DE PROYECTO

A continuación se calculan los costes estimados del desarrollo del proyecto. Estos costes incluyen los costes humanos y materiales desde que se inicia el proyecto hasta la fecha en que es presentado.

2.3.1. Coste horas-persona

Según los costes parciales obtenidos en las fases de Inicio, Elaboración, Construcción y Transición del plan de proyecto detallado anteriormente, los costes estimados para este proyecto son:

$$223horas + 240horas + 352horas + 60horas = 875horasPersona \tag{2.1}$$

2.3.2. Coste humano

El salario promedio de un desarrollador junior en las fechas en que se escribió este documento es aproximadamente de unos *1423 €/mes* según la página https://es.indeed.com/career/programador-junior/salaries. Considerando que al mes se realizan unas 160 horas laborales, el salario promedio aproximado que cobra por hora un desarrollador junior es el siguiente:

$$1424euros/mes * 1mes/160horas = 9euros/hora \tag{2.2}$$

Considerando el coste en horas-persona y el salario aproximado por horas de un desarrollador de software junior obtenemos que el coste humano del desarrollo del proyecto es el siguiente:

$$875horasPersona * 9euros/hora = 7875euros \tag{2.3}$$

2.3.3. Costes materiales

Todas las herramientas y tecnologías utilizadas para la implementación del proyecto son gratuitas.

Para el despliegue del sistema se utilizará una cuenta de prueba en Google Cloud Platform con 248 euros iniciales.

2.4. PLAN DE CONTROL DEL PROYECTO

En esta sección se analizan los riesgos existentes para el proyecto. En la Tabla 2.9 se muestra el valor de la exposición al riesgo en función de del impacto del riesgo y de la probabilidad.

Tabla 2.9: Exposición al riesgo

Probabilidad / **Impacto**	**Muy alta**	**Alta**	**Medio**	**Bajo**	**Muy bajo**
Catastrófico	Alto	Alto	Moderado	Moderado	Bajo
Crítico	Alto	Alto	Moderado	Bajo	Ninguno
Marginal	Moderado	Moderado	Bajo	Ninguno	Ninguno
Despreciable	Moderno	Bajo	Bajo	Ninguno	Ninguno

En las tablas siguientes se analizan los riesgos que se han considerado para este proyecto.

Tabla 2.10: Riesgo R1: Fallos en la planificación.

Identificador	R1
Nombre	Fallos en la planificación
Descripción	Los tiempos estimados en la planificación de las tareas son insuficientes.
Probabilidad	Muy alta
Impacto	Crítico
Exposición	Alta
Plan de mitigación	Realizar las tareas cumpliendo con los tiempos estimados.
Plan de contingencia	Priorizar tareas y realizar los ajustes necesarios en la planificación.

Tabla 2.11: Riesgo R2: Propuesta propia de proyecto que pueda resultar demasiado ambiciosa

Identificador	R2
Nombre	Propuesta propia de proyecto que pueda resultar demasiado ambiciosa
Descripción	‘ ▪ Una necesidad mayor de tomar decisiones. ▪ Mayor responsabilidad. ▪ Esfuerzo añadido para fijar el alcance del proyecto. ▪ Afrontar técnicas o herramientas excesivamente complejas. ▪ Capacidad para adquirir conocimientos de fuentes que exigen un alto nivel técnico. ▪ Capacidad para afrontar los problemas sin depender de nadie, etc.
Probabilidad	Alta
Impacto	Crítico
Exposición	Alta
Plan de mitigación	Afrontar el proyecto con mentalidad ambiciosa y la concienciación necesaria para asumir el reto personal.
Plan de contingencia	Añadir horas de estudio previo antes de tomar decisiones y elegir entre las diferentes alternativas de tecnologías a utilizar.

Tabla 2.12: Riesgo R3: Experiencia insuficiente.

Identificador	R3
Nombre	Experiencia insuficiente.
Descripción	Experiencia insuficiente para gestionar e implementar un sistema basado en una arquitectura de microservicios.
Probabilidad	Alta
Impacto	Crítico
Exposición	Alta
Plan de mitigación	Antes de iniciar el proyecto, realizar una formación intensiva en sistemas basados en arquitecturas de microservicios.
Plan de contingencia	Realizar estimaciones de tiempos elevadas, permitiendo márgenes de error altos en el proceso de desarrollo.

Tabla 2.13: Riesgo R4: Curva de aprendizaje más larga de lo esperado.

Identificador	R4
Nombre	Curva de aprendizaje más larga de lo esperado
Descripción	Una curva de aprendizaje elevada debido al gran número de tecnologías implicadas puede retrasar significativamente la fecha de finalización del proyecto.
Probabilidad	Muy alta
Impacto	Crítico
Exposición	Alta
Plan de mitigación	Antes de iniciar la implementación, realizar una formación intensiva en el uso adecuado de las diferentes tecnologías.
Plan de contingencia	Optar por tecnologías que no impliquen una curva de aprendizaje demasiado alta.

Tabla 2.14: Riesgo R5: Arquitectura que exige un alto nivel de conocimientos para ser implementada con éxito.

Identificador	R5
Nombre	Arquitectura que exige un alto nivel de conocimientos para ser implementada con éxito.
Descripción	No ser capaz de conseguir diseñar e implementar una aplicación basada en una arquitectura de microservicios, abandonando de esta forma el proyecto u obteniendo un resultado no deseado.
Probabilidad	Alta
Impacto	Catastrófico
Exposición	Alta
Plan de mitigación	Conseguir y estudiar una documentación técnica adecuada.
Plan de contingencia	Ampliar los tiempos estimados.

Tabla 2.15: Riesgo R6: Diseñar una aplicación demasiado grande.

Identificador	R6
Nombre	Diseñar una aplicación demasiado grande.
Descripción	Realizar una aplicación demasiado grande requerirá destinar mucho tiempo a la implementación de funcionalidad, en vez de centrarse en la arquitectura.
Probabilidad	Alta
Impacto	Crítico
Exposición	Alta
Plan de mitigación	Antes de iniciar el proyecto, realizar una formación intensiva en sistemas basados en arquitecturas de microservicios.
Plan de contingencia	Realizar estimaciones de tiempos elevadas, permitiendo márgenes de error en el proceso de desarrollo altos.

Tabla 2.16: Riesgo R7: Descuidar la apariencia y la usabilidad de la UI.

Identificador	R7
Nombre	Descuidar la apariencia y la usabilidad de la UI.
Descripción	Realizar una aplicación basada en una arquitectura de microservicios requerirá una bolsa de horas muy grande, esto puede causar prisas a la hora de cuidar el aspecto y la usabilidad de la UI.
Probabilidad	Medio
Impacto	Crítico
Exposición	Moderado
Plan de mitigación	Realizar una UI sencilla, con una apariencia simple pero profesional.
Plan de contingencia	Ampliar las horas estimadas a la totalidad del proyecto.

Tabla 2.17: Riesgo R8: Despliegue complejo y profesional.

Identificador	R8
Nombre	Despliegue complejo y profesional.
Descripción	Realizar el despliegue de una aplicación de microservicios en la nube, puede llegar a ser un proceso bastante complejo si se quiere aprovechar todos los beneficios que ofrece esta arquitectura.
Probabilidad	Muy Alta
Impacto	Marginal
Exposición	Moderada
Plan de mitigación	Realizar un estudio de los servicios que ofrecen las cuentas gratuitas, estudiar los recursos que se necesitan e intentar adaptar la aplicación para que el despliegue sea viable.
Plan de contingencia	Optar por otra alternativa menos profesional pero viable.

Tabla 2.18: Riesgo R9: Agotar los servicios gratuitos que ofrece el proveedor de servicios cloud.

Identificador	R9
Nombre	Agotar los servicios gratuitos que ofrece el proveedor de servicios cloud.
Descripción	Sobrepasar el consumo de servicios gratuitos puede causar que el proveedor corte los servicios desplegados y en algunos casos la posibilidad de costes económicos en la cuenta del usuario que realizó el despliegue.
Probabilidad	Media
Impacto	Crítico
Exposición	Moderada
Plan de mitigación	Planificar muy detenidamente el despliegue y no gastar servicios estudiando. Es muy probable que la forma más profesional no sea posible por falta de recursos en este proyecto.
Plan de contingencia	Optar por otra alternativa, intentar negociar con el proveedor de servicios cloud o conseguir otra cuenta gratuita, y con lo aprendido de los errores sufridos, realizar un despliegue viable con todos los servicios gratuitos al máximo.

Tabla 2.19: Riesgo R10: Incompatibilidad entre tecnologías.

Identificador	R10
Nombre	Incompatibilidad entre tecnologías.
Descripción	Realizar una aplicación cloud políglota envuelve a muchas tecnologías y puede que no todas sean compatibles entre sí.
Probabilidad	Baja
Impacto	Crítico
Exposición	Baja
Plan de mitigación	Estudiar los diferentes tipos de tecnologías y asegurarse de que son compatibles entre sí.
Plan de contingencia	Cambiar de tecnología.

Tabla 2.20: Riesgo R11: Enfermedades, problemas psicológicos y una pandemia mundial.

Identificador	R11
Nombre	Enfermedades, problemas psicológicos y una pandemia mundial.
Descripción	Puede producirse una demora en los tiempos si el desarrollador sufre alguna enfermedad o algún tipo de problema psicológico. La probabilidad de que ocurra alguno de estos hechos pueden verse intensificada al realizar el proyecto en tiempos de una pandemia mundial, en una situación social diferente a la normal.
Probabilidad	Baja
Impacto	Crítico
Exposición	Baja
Plan de mitigación	Ninguno.
Plan de contingencia	Aumentar la estimación de duración del proyecto.

Tabla 2.21: Riesgo R12: Ausencia de un equipo.

Identificador	R12
Nombre	Ausencia de un equipo.
Descripción	Realizar un proyecto de estas características de forma individual implica que no se dispondrá del punto de vista ni las ideas de otras personas trabajando de forma coordinada.
Probabilidad	Muy alta
Impacto	Marginal
Exposición	Moderada
Plan de mitigación	Conseguir una documentación de calidad y contrastar fuentes.
Plan de contingencia	Seguir las recomendaciones de las fuentes.

Tabla 2.22: Riesgo R13: El sistema operativo Windows Home y recursos del entorno de desarrollo justos.

Identificador	R13
Nombre	El sistema operativo Windows Home y recursos del entorno de desarrollo justos.
Descripción	Realizar una aplicación que involucra a tantas tecnologías implica tener una cantidad de programas e IDEs trabajando de forma paralela muy alta. La instalación de las herramientas puede requerir la necesidad de instalar una versión profesional del sistema operativo.
Probabilidad	Alta
Impacto	Crítico
Exposición	Alta
Plan de mitigación	Estudiar todas las posibilidades de herramientas para seleccionar las que vayan a causar menos problemas y un mayor rendimiento.
Plan de contingencia	Cambiar de herramienta si gasta muchos recursos del entorno de desarrollo, actualizar a Windows Pro, para poder instalar herramientas más adecuadas.

2.5. PLAN DE PROCESO TÉCNICO

En esta sección se enumeran las tecnologías y herramientas que se han utilizado durante el desarrollo del proyecto.

2.5.1. Tecnologías

- **ASP.Net Core:** framework web, modular, open source, multiplataforma, compatible con C# y desarrollado por Microsoft. Ha sido utilizada la versión 3.10 de este framework para implementar varios de los microservicios que forman el back end del sistema.
- **Spring Boot:** basado en Spring Framework y compatible con Java. Ha sido utilizado junto a Java 11 para implementar varios de los microservicios que forman el back end del sistema.
- **Angular:** framework para aplicaciones web desarrollado en TypeScript, open source, que se usa para crear aplicaciones web de una sola página y es mantenido por Google. La versión 10 de este framework ha sido utilizada para implementar la aplicación web, junto a HTML5 y CSS.
- **Netflix OSS:** conjunto de herramientas que se han utilizado junto al framework Spring para implementar algunos de los patrones más comunes que se usan en arquitecturas de microservicios.
- **Docker:** se ha utilizado para automatizar el despliegue de los microservicios en contenedores de software, proporcionando una capa de abstracción en múltiples sistemas operativos.
- **Kubernetes:** se ha utilizado para organizar los clústeres de máquinas virtuales y programar los contenedores que ejecutan cada microservicio para que se ejecuten en máquinas en función de los recursos que necesiten. Además facilita el escalado y mejora la disponibilidad del sistema.
- **GitHub:** repositorio de proyectos en la nube que se ha utilizado para el control de versiones.

2.5.2. Herramientas

- **Visual Studio 2019:** entorno de desarrollo para Windows. Este IDE ha sido utilizado para desarrollar en .Net Core, al tratarse del IDE recomendado.
- **Spring Boot Suite:** IDE basado en Eclipse recomendado para el desarrollo de cualquier producto de Spring. Este IDE ha sido utilizado para desarrollar con Spring Boot y Java.
- **Visual Studio Code:** editor de código fuente desarrollado por Microsoft. Ha sido utilizado para el desarrollo en Angular al ser un editor de código muy cómodo para este framework gracias a su catálogo de librerías.
- **Postman:** herramienta utilizada para facilitar el desarrollo de los tests de APIs.
- **HeidiSQL:** Gestor de bases de datos utilizado para la gestión de bases de Datos MySQL y SQL Server.
- **SQL Server Management Studio:** Gestor de bases de datos utilizado para la gestión de algunas operaciones en bases de datos SQL Server.
- **Draw.io:** herramienta online para crear alguna de las figuras de la memoria.
- **Astah UML:** se ha usado para realizar los diferentes diagramas UML.

- **Sourcetree:** cliente git para Windows.
- **Docker Desktop:** aplicación nativa para Windows 10, es la forma más eficiente de desarrollar aplicaciones Docker en Windows. Docker Desktop ha sido configurado para ejecutarse por completo en **WSL 2 (Windows Subsystem for Linux)**, de esta forma los contenedores pueden ejecutarse de forma nativa en Linux, sin necesidad de emulación, dando lugar a un mejor rendimiento. También incluye Kubernetes para realizar despliegues de forma local.
- **Overleaf:** para realizar la memoria en LaTeX

2.6. SEGUIMIENTO DEL PLAN DE PROYECTO

2.6.1. Fase de Inicio

Tabla 2.23: Seguimiento del proyecto: Fase de inicio - Iteración 1

Nombre de la tarea	Duración estimada (horas)	Duración real (horas)	Comienzo estimado	Final estimado	Comienzo Real	Final real
FASE DE INICIO - ITERACIÓN 1	**189**	**379**	**02/03/2020**	**01/05/2020**	**02/03/2020**	**20/05/2020**
Comienzo de la iteración 1	0	0	02/03/2020	02/03/2020	02/03/2020	02/03/2020
Planteamiento del proyecto	30	60	02/03/2020	17/04/2020	02/03/2020	24/04/2020
Adquirir documentación y contrastar información	20	20	02/03/2020	27/03/2020	02/03/2020	27/03/2020
Estudio de las arquitecturas de microservicios	80	200	02/03/2020	04/05/2020	02/03/2020	22/05/2020
Estudio tecnologías implicadas	40	80	02/03/2020	01/05/2020	02/03/2020	22/05/2020
Plan de fases	4	4	01/04/2020	01/04/2020	18/05/2020	18/05/2020
Plan del proyecto	8	8	20/04/2020	27/04/2020	19/05/2020	19/05/2020
Plan de riesgos	6	6	27/04/2020	01/05/2020	20/05/2020	20/05/2020
Plan de iteración siguiente	1	1	01/05/2020	01/05/2020	22/05/2020	22/05/2020
Fin de la fase de inicio	0	0	01/05/2020	01/05/2020	22/05/2020	22/05/2020

Como se puede observar en la tabla anterior, en esta fase se adquirió documentación sobre la arquitectura basada en microservicios y se comenzó con el estudio. Al tratarse de un proyecto que abarca conocimientos nuevos que no se habían dado en profundidad durante la carrera (R2), todo resultó ser más complejo y desconocido para el desarrollador del proyecto de lo que se estimó inicialmente (R5), por lo que las horas de estudio necesarias fueron mucho mayores (R1). También, el hecho de que la cantidad de tecnologías implicadas en el proyecto fuera alta, la necesidad de analizar la compatibilidad entre las tecnologías (R10) y el hecho de que el desarrollador no tuviera la experiencia suficiente (R3), causó retrasos considerable en la primera fase del proyecto.

Tabla 2.24: Seguimiento del proyecto: Fase de inicio - Iteración 2

Nombre de la tarea	Duración estimada (horas)	Duración real (horas)	Comienzo estimado	Final estimado	Comienzo Real	Final real
FASE DE INICIO - ITERACIÓN 2	**34**	**40**	**01/05/2020**	**22/05/2020**	**22/05/2020**	**10/06/2020**
Comienzo de la iteración 2	0	0	01/05/2020	01/05/2020	22/05/2020	22/05/2020
Análisis de requisitos iniciales	6	6	01/05/2020	06/05/2020	22/05/2020	22/05/2020
Estudio entorno cloud	12	12	06/05/2020	12/05/2020	22/05/2020	29/05/2020
Preparar entorno de desarrollo	12	18	06/05/2020	22/05/2020	29/05/2020	10/06/2020
Documento costes	1	1	22/05/2020	22/05/2020	09/06/2020	09/06/2020
Documento seguimiento	2	2	22/05/2020	22/05/2020	09/06/2020	09/06/2020
Plan de iteración siguiente	1	1	22/05/2020	22/05/2020	09/06/2020	09/06/2020
Fin de la fase de inicio	0	0	22/05/2020	22/05/2020	10/06/2020	10/06/2020

Desarrollar un proyecto que implica muchas tecnologías y además se utiliza inicialmente un entorno de desarrollo con el sistema operativo Windows Home, implica que la preparación del entorno de desarrollo sea más compleja de lo esperado (R13).

2.6.2. Fase de Elaboración

Tabla 2.25: Seguimiento del proyecto: Fase de elaboración - Iteración 1

Nombre de la tarea	Duración estimada (horas)	Duración real (horas)	Comienzo estimado	Final estimado	Comienzo Real	Final real
FASE DE ELABORACIÓN - ITERACIÓN 1	**46**	**57**	**22/05/2020**	**12/06/2020**	**10/06/2020**	**26/06/2020**
Comienzo de la iteración 1	0	0	22/05/2020	22/05/2020	10/06/2020	10/06/2020
Elicitación de requisitos	8	8	22/05/2020	25/05/2020	10/06/2020	11/06/2020
Diagrama de casos de uso	10	12	25/05/2020	29/05/2020	11/06/2020	15/06/2020
Descripción de casos de uso	12	16	29/05/2020	05/06/2020	15/06/2020	19/06/2020
Modelo de dominio en análisis	15	20	01/06/2020	12/06/2020	18/06/2020	26/06/2020
Plan de iteración siguiente	1	1	12/06/2020	12/06/2020	26/06/2020	26/06/2020

Diseñar una aplicación demasiado grande (R6), junto al retos de diseño para el desarrollador del proyecto de un tipo de arquitectura de una complejidad considerable (R5), han sido las principales causas de retrasos en la Fase de Elaboración.

Tabla 2.26: Seguimiento del proyecto: Fase de elaboración - Iteración 2

Nombre de la tarea	Duración estimada (horas)	Duración real (horas)	Comienzo estimado	Final estimado	Comienzo Real	Final real
FASE DE ELABORACIÓN - ITERACIÓN 2	**63**	**88**	**12/06/2020**	**09/07/2020**	**26/06/2020**	**24/07/2020**
Comienzo de la iteración 2	0	0	12/06/2020	12/06/2020	26/06/2020	26/06/2020
Revisión requisitos	3	3	12/06/2020	12/06/2020	26/06/2020	26/06/2020
Revisión casos de uso	4	4	15/06/2020	15/06/2020	26/06/2020	30/06/2020
Aplicar Domain Driven Desing	15	20	16/06/2020	22/06/2020	30/06/2020	06/07/2020
Diseño de las bases de datos	10	10	22/06/2020	24/06/2020	06/07/2020	10/07/2020
Implementación del modelo y del acceso a persistencia	30	50	24/06/2020	09/07/2020	10/07/2020	24/07/2020
Plan de iteración siguiente	1	1	09/07/2020	09/07/2020	24/07/2020	24/07/2020

Tabla 2.27: Seguimiento del proyecto: Fase de elaboración - Iteración 3

Nombre de la tarea	Duración estimada (horas)	Duración real (horas)	Comienzo estimado	Final estimado	Comienzo Real	Final real
FASE DE ELABORACIÓN - ITERACIÓN 3	**131**	**161**	**09/07/2020**	**31/07/2020**	**24/08/2020**	**09/10/2020**
Comienzo de la iteración 3	0	0	09/07/2020	09/07/2020	24/08/2020	24/08/2020
Diseño de la API Gateway	15	20	09/07/2020	16/07/2020	24/08/2020	03/09/2020
Diseño de la autenticación del sistema	10	20	09/07/2020	16/07/2020	24/08/2020	03/09/2020
Implementación de la API Gateway	25	25	16/07/2020	22/07/2020	03/09/2020	09/09/2020
Diseño de la arquitectura de los microservicios	20	25	22/07/2020	24/07/2020	09/09/2020	18/09/2020
Implementación de la lógica de los microservicios	60	70	24/07/2020	31/07/2020	18/09/2020	09/10/2020
Plan de iteración siguiente	1	1	31/07/2020	31/07/2020	09/10/2020	09/10/2020

2.6.3. Fase de Construcción

Tabla 2.28: Seguimiento del proyecto: Fase de construcción - Iteración 1

Nombre de la tarea	Duración estimada (horas)	Duración real (horas)	Comienzo estimado	Final estimado	Comienzo Real	Final real
FASE DE CONSTRUCCIÓN - ITERACIÓN 1	**221**	**321**	**31/07/2020**	**21/09/2020**	**09/10/2020**	**16/12/2020**
Comienzo de la iteración 1	0	0	31/07/2020	31/07/2020	09/10/2020	09/10/2020
Diseño de la API de los microservicios	20	20	31/07/2020	06/08/2020	09/10/2020	15/10/2020
Implementación de las APIs de los microservicios	50	70	06/08/2020	21/08/2020	15/10/2020	03/11/2020
Diseño de la comunicación entre los microservicios	40	70	21/08/2020	01/09/2020	03/11/2020	13/11/2020
Implementación de la comunicación entre microservicios	100	150	01/09/2020	21/09/2020	13/11/2020	16/12/2020
Pruebas de los elementos desarrollados en esta fase	10	10	15/09/2020	21/09/2020	11/12/2020	16/12/2020
Plan de iteración siguiente	1	1	21/09/2020	21/09/2020	16/12/2020	16/12/2020

Tabla 2.29: Seguimiento del proyecto: Fase de construcción - Iteración 2

Nombre de la tarea	Duración estimada (horas)	Duración real (horas)	Comienzo estimado	Final estimado	Comienzo Real	Final real
FASE DE CONSTRUCCIÓN - ITERACIÓN 2	**131**	**151**	**21/09/2020**	**23/10/2020**	**16/12/2020**	**22/01/2021**
Comienzo de la iteración 2	0	0	21/09/2020	21/09/2020	16/12/2020	16/12/2020
Diseño de la arquitectura front end	15	15	21/09/2020	25/09/2020	16/12/2020	22/12/2020
Implementación del frontend	80	100	25/09/2020	23/10/2020	22/12/2020	22/01/2021
Completar la implementación de la lógica de negocio	15	15	10/10/2020	23/10/2020	15/01/2021	22/01/2021
Corrección de bugs graves	20	20	10/10/2020	23/10/2020	15/01/2021	22/01/2021
Plan de iteración siguiente	1	1	23/10/2020	23/10/2020	22/01/2021	22/01/2021

El uso de patrones de diseño nuevos (R3), la complejidad técnica (R5) y la curva de aprendizaje alta (R4), ha causado muchas dificultades a la hora de implementar la comunicación entre microservicios mediante un message boker. Esto junto a las dimensiones de la aplicación diseñada (R6) han sido la principal causa de retrasos en la Fase de Construcción.

2.6.4. Fase de Transición

Tabla 2.30: Seguimiento del proyecto: Fase de transición - Iteración 1

Nombre de la tarea	Duración estimada (horas)	Duración real (horas)	Comienzo estimado	Final estimado	Comienzo Real	Final real
FASE DE TRANSICIÓN - ITERACIÓN 1	**60**	**75**	**23/10/2020**	**27/11/2020**	**22/01/2021**	**12/02/2021**
Comienzo de la iteración 1	0	0	23/10/2020	23/10/2020	22/01/2021	22/01/2021
Correcciones leves y mejoras	25	25	23/10/2020	16/11/2020	22/01/2021	01/02/2021
Pruebas	20	20	16/11/2020	24/11/2020	22/01/2021	03/02/2021
Despliegue	10	25	24/11/2020	26/11/2020	03/02/2021	11/02/2021
Manuales	5	5	26/11/2020	27/11/2020	11/02/2021	12/02/2021

En la Fase de Transición los retrasos producidos han sido causados principalmente por el despliegue de un sistema complejo de forma profesional en Google Kubernetes Engine (R8). La falta de conocimientos con las tecnologías y el proveedor de servicios cloud (R4), han causado que se agote los servicios gratuitos de la cuenta de prueba (R9).

2.7. SEGUIMIENTO DE COSTES

A continuación se presentan los costes reales una vez finalizado el proyecto.

2.7.1. Coste horas-persona

Según las duraciones reales de las fases de Inicio, Elaboración, Construcción y Transición obtenidas durante el seguimiento del proyecto, los costes horas-persona totales son los siguientes:

$$419horas + 306horas + 452horas + 75horas = 1272horasPersona \tag{2.4}$$

2.7.2. Coste humano

Considerando el coste en horas-persona y el salario aproximado por horas de un desarrollador de software junior obtenemos que el coste humano del desarrollo del proyecto es el siguiente:

$$1272horasPersona * 9euros/hora = 11448euros \tag{2.5}$$

2.7.3. Costes materiales

Tabla 2.31: Costes materiales

Recurso	**Descripción**	**Coste (€)**
Ordenador personal	Ya amortizado	0
Herramientas	Todas las herramientas utilizadas en este proyecto son gratuitas o se ha utilizado una versión gratuita para estudiantes.	0
Proveedor de servicios cloud	Cuenta de prueba Google Cloud Platform	306,53
TOTAL		**306,53**

Tal y como se describe en los riesgos R8 y R9, la cuenta de Google Cloud Platform que se usó para estudiar y posteriormente realizar el despliegue en producción, ha sido insuficiente para mantener el despliegue tal y como se planteó inicialmente. Ha sido necesaria otra cuenta y replantear un despliegue menos ambicioso.

2.7.4. Coste total del proyecto

El coste total del proyecto es el siguiente:

$$CosteTotal = CosteHumano + CostesMateriales = 11448euros + 306,53euros = 11754,63euros \tag{2.6}$$

CAPÍTULO 3

ELICITACIÓN DE REQUISITOS Y ANÁLISIS

3.1. REQUISITOS DEL SISTEMA

3.1.1. Requisitos funcionales

RF-1 El sistema deberá permitir a un *Administrador* registrar un Empleado.
RF-2 El sistema deberá permitir a un *Administrador* crear una categoría de productos.
RF-3 El sistema deberá permitir a un *Administrador* crear una subcategoría de productos dentro de una categoría de productos.
RF-4 El sistema deberá permitir a un *Administrador* añadir productos a una subcategoría.
RF-5 El sistema deberá permitir a un *Administrador* añadir productos a una subcategoría.
RF-6 El sistema deberá permitir a un *Administrador* modificar un producto de una subcategoría.
RF-7 El sistema deberá permitir a un *Administrador* eliminar un producto.
RF-8 El sistema deberá permitir a un *Administrador* crear una tienda.
RF-9 El sistema deberá permitir a un *Administrador* crear un almacén.
RF-10 El sistema deberá permitir a un *Administrador* modificar una tienda.
RF-11 El sistema deberá permitir a un *Administrador* modificar un almacén.
RF-12 El sistema deberá permitir a un *Administrador* borrar una tienda.
RF-13 El sistema deberá permitir a un *Administrador* borrar un almacén.
RF-14 El sistema deberá permitir a un *Administrador* consultar las tiendas.
RF-15 El sistema deberá permitir a un *Administrador* consultar los almacenes.
RF-16 El sistema deberá permitir a un *Administrador* consultar el stock de los productos en tiendas.
RF-17 El sistema deberá permitir a un *Administrador* consultar el stock de los productos en alamcenes.
RF-18 El sistema deberá permitir a un *Administrador* asignar un producto a una o más tiendas.
RF-19 El sistema deberá permitir a un *Administrador* asignar un producto a uno o más almacenes.
RF-20 El sistema deberá permitir a un *Administrador* consultar los empleados de una tienda.
RF-21 El sistema deberá permitir a un *Administrador* consultar los empleados de un almacén.
RF-22 El sistema deberá permitir a un *Administrador* añadir un nuevo empleado a una tienda.
RF-23 El sistema deberá permitir a un *Administrador* añadir un nuevo empleado a un almacén.
RF-24 El sistema deberá permitir a un *Administrador* modificar la información de un empleado.
RF-25 El sistema deberá permitir a un *Administrador* eliminar un empleado.
RF-26 El sistema deberá permitir a un *Administrador* consultar los pedidos de productos por estado.
RF-27 El sistema deberá permitir a un *Cliente* y a un *Administrador* consultar las categorías.
RF-28 El sistema deberá permitir a un *Cliente* y a un *Administrador* consultar las subcategorías de una categoría.
RF-29 El sistema deberá permitir a un *Cliente* y a un *Administrador* consultar los productos de una subcategoría.
RF-30 El sistema deberá permitir a un *Administrador*, a un *Cliente Identificado* y a un *Empleado*

identificarse.

RF-31 El sistema deberá permitir a un *Cliente* registrarse.

RF-32 El sistema deberá permitir a un *Cliente* consultar las novedades de productos.

RF-33 El sistema deberá permitir a un *Empleado Tienda* consular el stock de los productos asignados a su tienda.

RF-34 El sistema deberá permitir a un *Empleado Almacén* consular el stock de los productos asignados a su almacén.

RF-35 El sistema deberá permitir a un *Cliente* consular la disponibilidad de un producto en la tienda online.

RF-36 El sistema deberá permitir a un *Cliente* consular la disponibilidad de un producto en una tienda.

RF-37 El sistema deberá permitir a un *Cliente* consultar el carro de la compra.

RF-38 El sistema deberá permitir a un *Cliente* modificar el carro de la compra.

RF-39 El sistema deberá permitir a un *Cliente* añadir productos al carro de la compra.

RF-40 El sistema deberá permitir a un *Cliente Identificado* crear una cuenta cliente.

RF-41 El sistema deberá permitir a un *Cliente Identificado* consultar los datos de su cuenta.

RF-42 El sistema deberá permitir a un *Cliente Identificado* modificar los datos de su cuenta.

RF-43 El sistema recuperará el estado del carro de un *Cliente Identificado*.

RF-44 El sistema añadirá al sistema el estado actual de un carro en el momento que el *Cliente Identificado* se identifique.

RF-45 El sistema registrará en el sistema el estado actual de un carro en el momento que el *Cliente* se registre.

RF-46 El sistema deberá permitir a un *Empleado* consultar la información de su cuenta.

RF-47 El sistema deberá permitir a un *Cliente Identificado* realizar un pedido.

RF-48 El sistema deberá permitir a un *Cliente Identificado* consultar sus pedidos realizados.

RF-49 El sistema deberá permitir a un *Empleado Almacén* consultar los pedidos de productos de su almacén.

RF-50 El sistema deberá permitir a un *Empleado Almacén* modificar el estado de un pedido de producto a su almacén de Confirmado a Preparado.

3.1.2. Requisitos no funcionales

RNF-1 La contraseña de un usuario debe almacenarse en el sistema de forma encriptada.

RNF-2 La aplicación debe tener una alta disponibilidad.

RNF-3 La aplicación debe tener una alta escalabilidad.

RNF-4 La aplicación debe ser tolerante a fallos.

RNF-5 La aplicación debe ser desplegada en servidores cloud.

RNF-6 La identificación del usuario en el sistema debe ser segura.

RNF-7 La aplicación deberá funcionar como mínimo en los navegadores web más utilizados.

RNF-8 La aplicación tendrá un tiempo de respuesta por cada interacción del usuario inferior a 4 segundos.

3.1.3. Reglas de negocio

RN-1 La última tarjeta de crédito que use un cliente será almacenada en el sistema.

RN-2 Los estados de un pedido de producto son *Pendiente*, *Confirmado*, *Preparado*, *En camino*, *Entregado* y *Cancelado*.

RN-3 Para que un Empleado Almacén pueda cambiar el estado de un pedido producto a *Preparado*, el pedido producto debe tener el estado *Confirmado*.

RN-4 En esta versión de la aplicación, el stock de un producto para cada una de sus tallas será inicializado con un valor por defecto.

RN-5 El estado de un pedido de producto pasa de *Pendiente* a *Confirmado* una vez que se ha verificado que hay existencias suficientes de la talla indicada, en caso contrario pasa a estado *Cancelado*.

RN-6 Cuando un usuario anónimo inicie sesión como cliente, se fusionará el carro que tenía como anónimo con el carro de su cuenta de cliente.

RN-7 Cuando un usuario anónimo se registre como cliente, su carro de la compra será añadido al carro de la compra de su cuenta cliente.

RN-8 La cantidad de stock de un producto para una talla disponible en la tienda online, será la suma del stock de la talla de ese producto en todos los almacenes de la franquicia.

3.2. CASOS DE USO

Con el fin de mejorar la legibilidad del diagrama, se ha dividido el diagrama de Casos de Uso en subdiagramas. Cada subdiagrama contiene los casos de uso de un actor que interactúa con en el sistema.

3.2.1. Diagrama de Casos de Uso

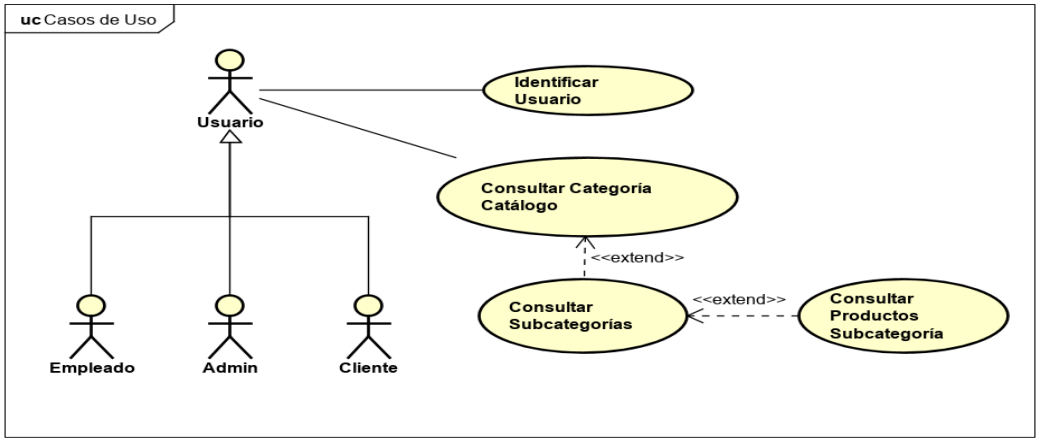

Figura 3.1: Casos de uso del actor Usuario.

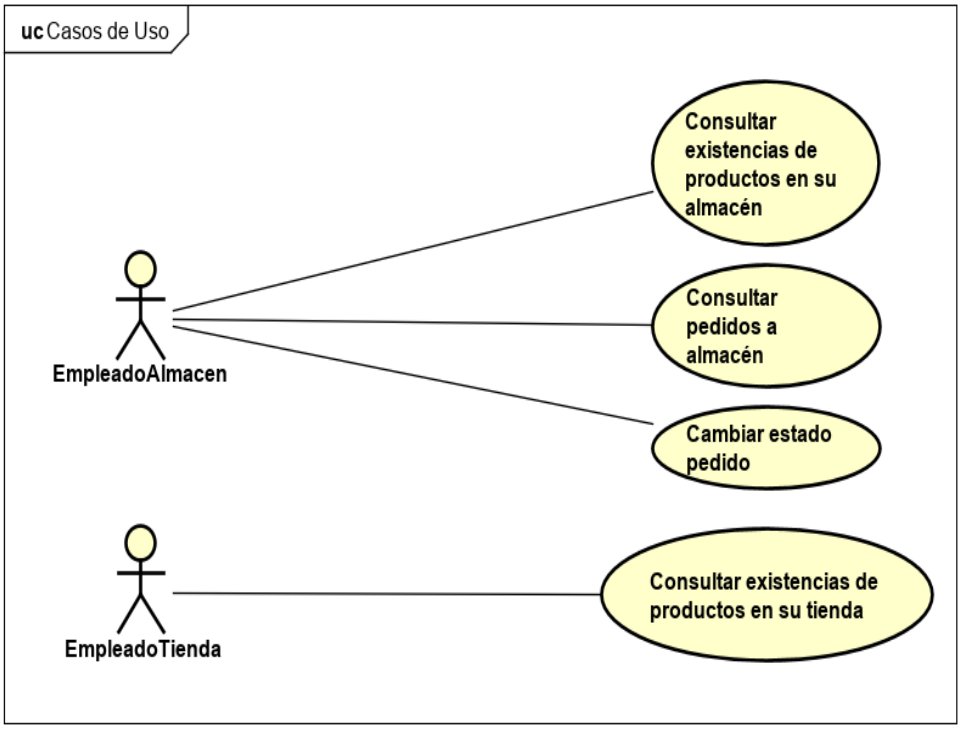

Figura 3.2: Casos de uso del actor EmpleadoAlmacen y del actor EmpleadoTienda.

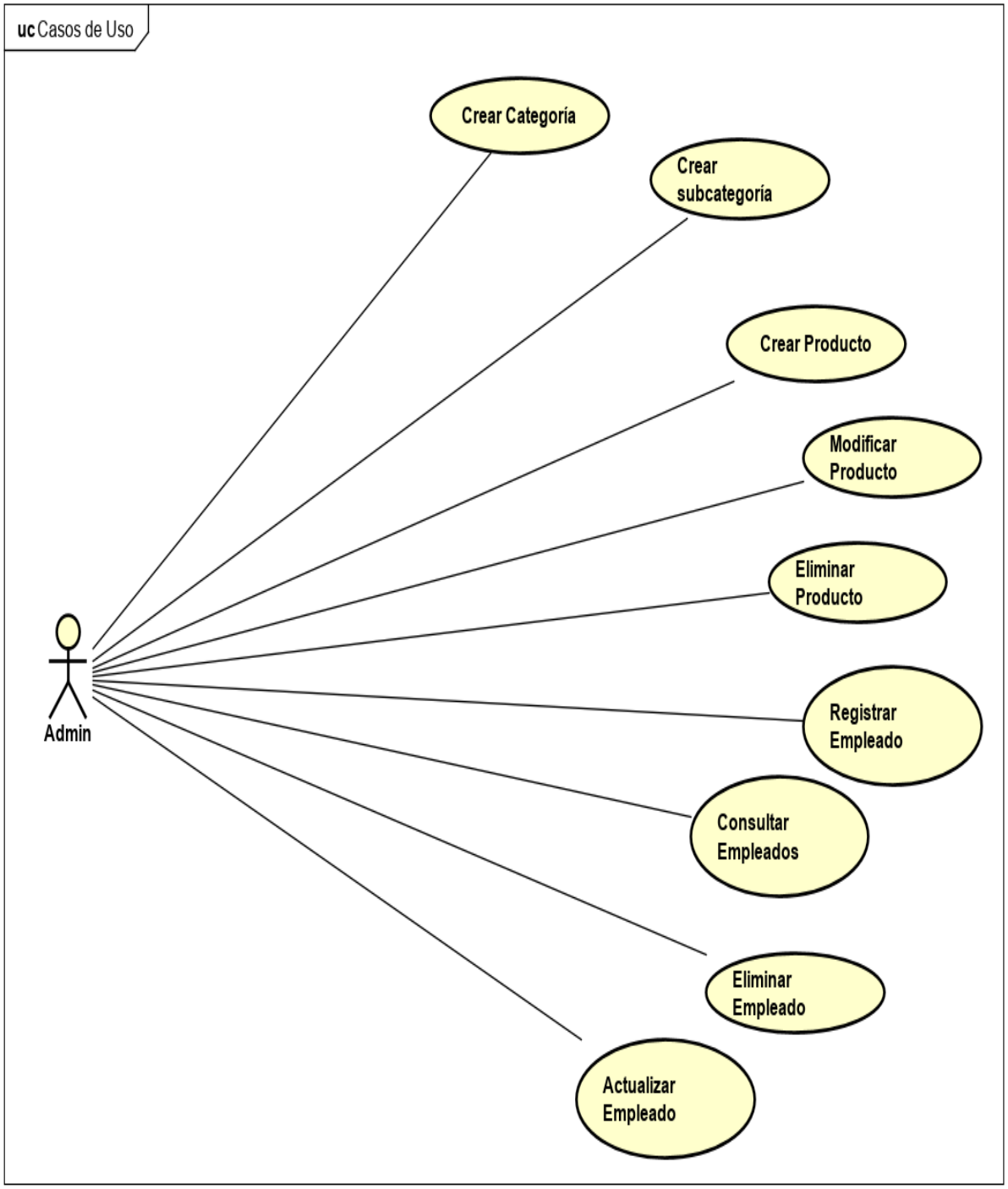

Figura 3.3: Casos de uso del actor Administrador 1.

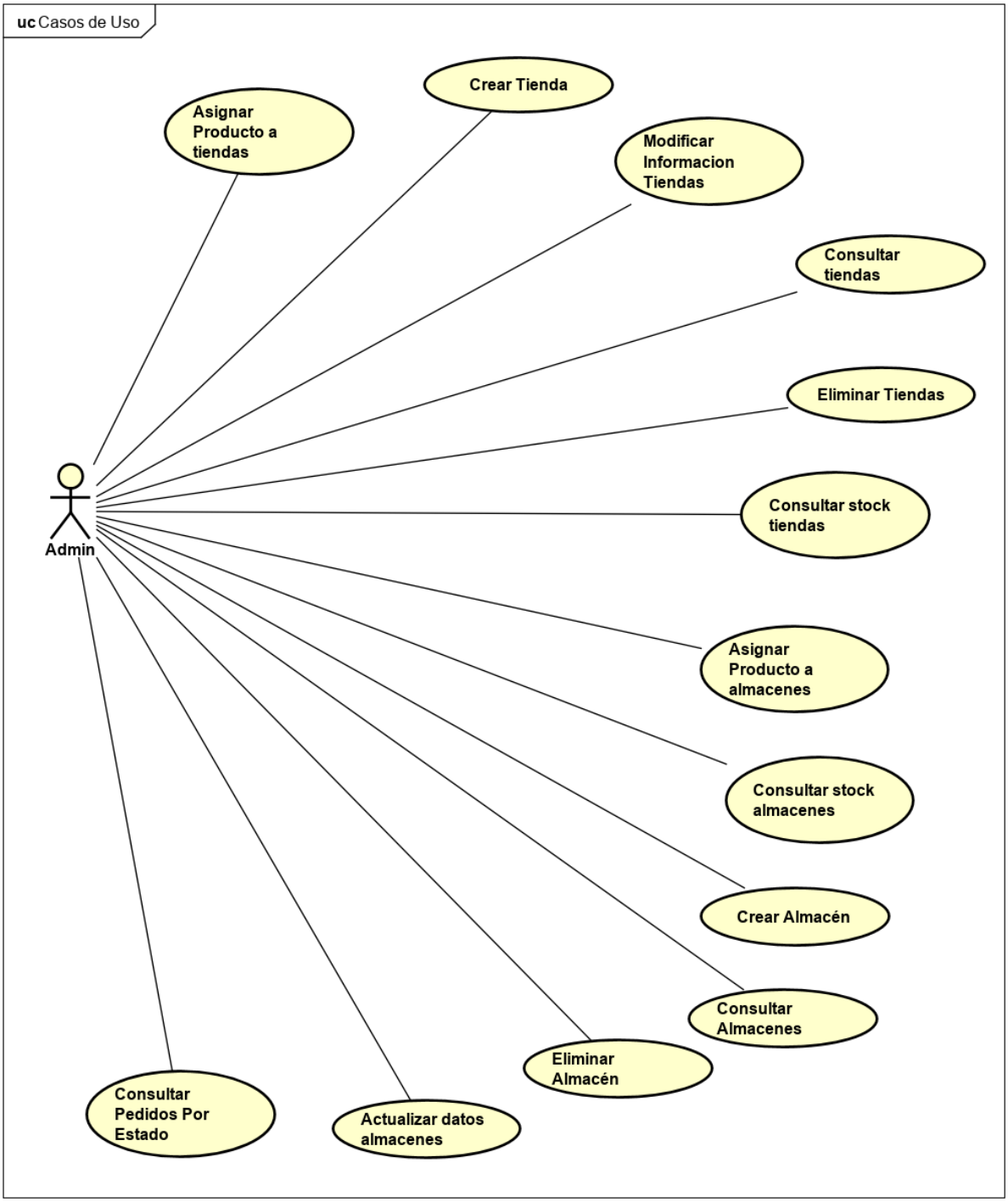

Figura 3.4: Casos de uso del actor Administrador 2.

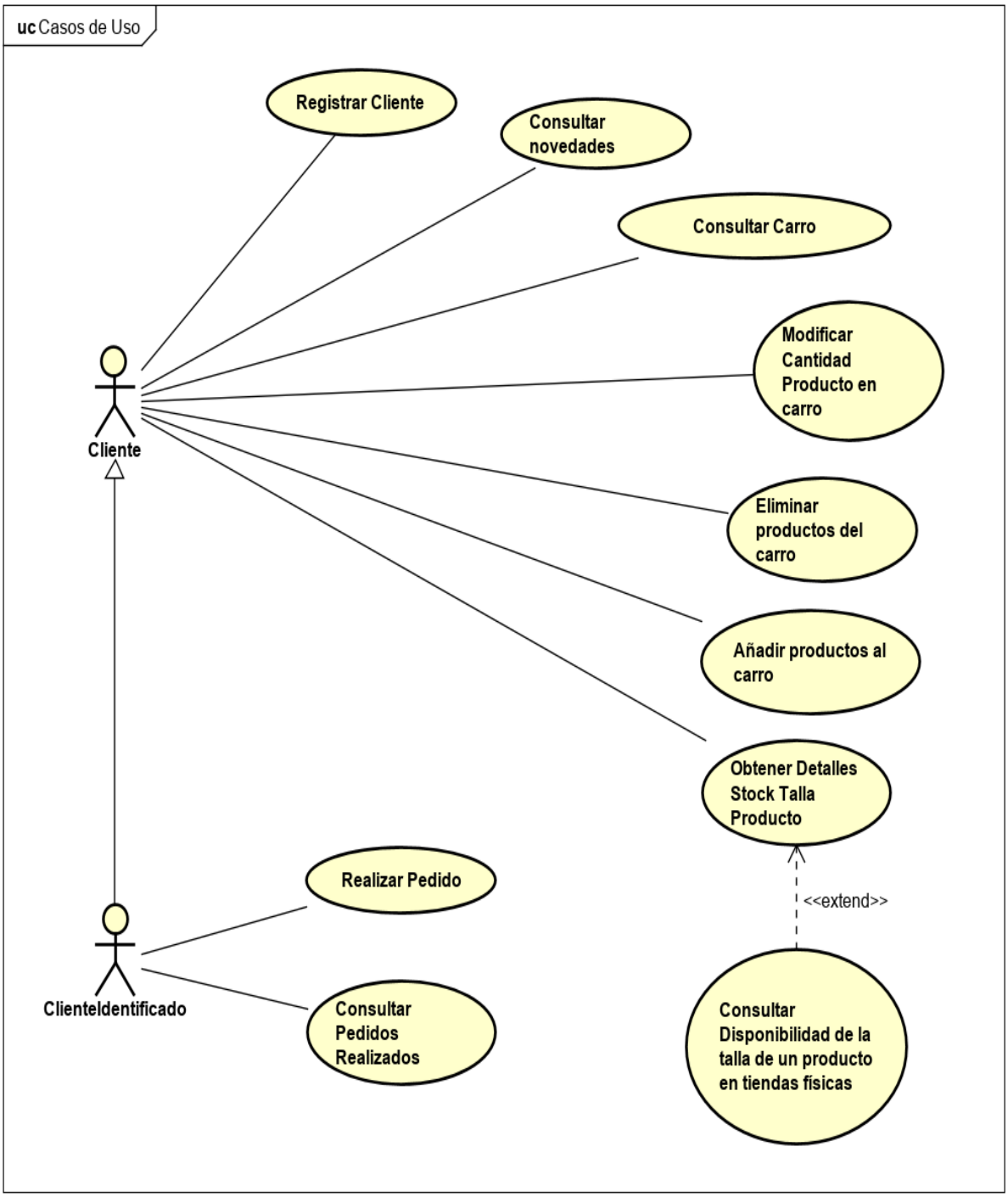

Figura 3.5: Casos de uso del actor Cliente.

3.2.2. Descripción de los Casos de Uso

Tabla 3.1: Caso de Uso: *Registrar Empleado*

Título	Registrar Empleado
Resumen	El actor *Administrador* quiere registrar un empleado en el sistema.
Precondición	El actor *Administrador* está identificado en el sistema.
Postcondición	El empleado queda registrado en el sistema.
Secuencia Principal	1. El actor *Administrador* selecciona la opción registrar empleado en el sistema. 2. El sistema solicita al actor *Administrador* introducir el nombre de usuario y contraseña del empleado. 3. El actor *Administrador* introduce los datos y selecciona *Registrar*. 4. El sistema registra al nuevo empleado.
Secuencia Excepcional	2.1 El actor *Administrador* cancela y el caso de uso queda sin efecto. 4.1 El sistema rechaza el registro debido a que el nombre de usuario ya existe y muestra un mensaje.

Tabla 3.2: Caso de Uso: *Crear Categoría*

Título	Crear Categoría
Resumen	El actor *Administrador* quiere crear una categoría.
Precondición	El actor *Administrador* está identificado en el sistema.
Postcondición	El sistema crea la categoría.
Secuencia Principal	1. El actor *Administrador* selecciona la opción crear Categoría. 2. El sistema solicita el nombre de la categoría. 3. El actor *Administrador* introduce el nombre de la categoría y selecciona *Crear*. 4. El sistema crea la categoría.
Secuencia Excepcional	3.3 El actor *Administrador* cancela y el caso de uso queda sin efecto.

Tabla 3.3: Caso de Uso: *Crear Subcategoría*

Título	Crear Subcategoría
Resumen	El actor *Administrador* quiere crear una subcategoría.
Precondición	1. El actor *Administrador* está identificado en el sistema. 2. El actor *Administrador* ha seleccionado previamente una categoría.
Postcondición	El sistema añade la subcategoría creada a la categoría seleccionada.
Secuencia Principal	1. El actor *Administrador* selecciona la opción crear subcategoría. 2. El sistema solicita el nombre de la subcategoría y el tipo de talla de los productos que serán asignados. 3. El actor *Administrador* introduce los datos y selecciona *Crear*. 4. El sistema crea la categoría.
Secuencia Excepcional	3.1 El actor *Administrador* cancela y el caso de uso queda sin efecto.

Tabla 3.4: Caso de Uso: *Crear Producto*

Título	Crear Producto
Resumen	El actor *Administrador* quiere crear una producto.
Precondición	1. El actor *Administrador* está identificado en el sistema. 2. El actor *Administrador* ha seleccionado previamente una subcategoría.
Postcondición	El sistema añade el producto creado a la subcategoría seleccionada.
Secuencia Principal	1. El actor *Administrador* selecciona la opción crear producto. 2. El sistema solicita el nombre, el precio y la url de la imagen del producto que será creado. 3. El actor *Administrador* introduce los datos y selecciona *Crear*. 4. El sistema crea el producto.
Secuencia Excepcional	2.1 El actor *Administrador* cancela y el caso de uso queda sin efecto.

Tabla 3.5: Caso de Uso: *Modificar Producto*

Título	Modificar Producto
Resumen	El actor *Administrador* quiere modificar una producto.
Precondición	El actor *Administrador* está identificado en el sistema.
Postcondición	El sistema actualiza el producto modificado.
Secuencia Principal	1. El actor *Administrador* selecciona la opción modificar un producto. 2. El sistema permite al *Administrador* modificar los datos del producto. 3. El actor *Administrador* modifica los datos y selecciona *Actualizar*. 4. El sistema actualiza el producto.
Secuencia Excepcional	2.1 El actor *Administrador* cancela y el caso de uso queda sin efecto.

Tabla 3.6: Caso de Uso: *Eliminar Producto*

Título	Eliminar Producto
Resumen	El actor *Administrador* quiere eliminar una producto.
Precondición	El actor *Administrador* está identificado en el sistema.
Postcondición	El sistema elimina el producto.
Secuencia Principal	1. El actor *Administrador* selecciona la opción eliminar de un producto. 2. El sistema consulta al *Administrador* si desea eliminar el producto del sistema. 3. El actor *Administrador* selecciona *Confirmar*. 4. El sistema elimina el producto.
Secuencia Excepcional	2.1 El actor *Administrador* cancela y el caso de uso queda sin efecto.

Tabla 3.7: Caso de Uso: *Crear tienda*

Título	Crear tienda
Resumen	El actor *Administrador* quiere añadir una tienda al sistema.
Precondición	El actor *Administrador* está identificado en el sistema.
Postcondición	El sistema crea una tienda.
Secuencia Principal	1. El actor *Administrador* selecciona la opción crear tienda. 2. El sistema solicita al actor *Administrador* la dirección y el teléfono del almacén que quiere crear. 3. El actor *Administrador* introduce los datos y selecciona *Crear*. 4. El sistema crea el almacén.
Secuencia Excepcional	2.1 El actor *Administrador* cancela y el caso de uso queda sin efecto.

Tabla 3.8: Caso de Uso: *Crear almacén*

Título	Crear almacén
Resumen	El actor *Administrador* quiere añadir un almacén al sistema.
Precondición	El actor *Administrador* está identificado en el sistema.
Postcondición	El sistema crea un almacén.
Secuencia Principal	1. El actor *Administrador* selecciona la opción crear almacén. 2. El sistema solicita al actor *Administrador* la dirección y el teléfono del almacén que va a crear. 3. El actor *Administrador* introduce los datos y selecciona *Crear*. 4. El sistema crea el almacén.
Secuencia Excepcional	2.1 El actor *Administrador* cancela y el caso de uso queda sin efecto.

Tabla 3.9: Caso de Uso: *Consultar almacenes*

Título	Consultar almacenes
Resumen	El actor *Administrador* quiere consultar los almacenes.
Precondición	El actor *Administrador* está identificado en el sistema.
Postcondición	El sistema muestra una lista con todos los almacenes del sistema.
Secuencia Principal	1. El actor *Administrador* selecciona la opción consultar almacenes. 2. El sistema muestra al actor *Administrador* una lista con todos los almacenes del sistema.
Secuencia Excepcional	Ninguna.

Tabla 3.10: Caso de Uso: *Consultar tiendas*

Título	Consultar tiendas
Resumen	El actor *Administrador* quiere consultar las tiendas.
Precondición	El actor *Administrador* está identificado en el sistema.
Postcondición	El sistema muestra una lista con todas las tiendas del sistema.
Secuencia Principal	1. El actor *Administrador* selecciona la opción consultar tiendas. 2. El sistema muestra al actor *Administrador* una lista con todas las tiendas del sistema.
Secuencia Excepcional	Ninguna.

Tabla 3.11: Caso de Uso: *Consultar stock almacenes*

Título	Consultar stock almacenes
Resumen	El actor *Administrador* quiere consultar el stock de un almacén.
Precondición	El actor *Administrador* está identificado en el sistema.
Postcondición	El sistema muestra la información sobre el stock de un almacén.
Secuencia Principal	1. El actor *Administrador* selecciona un almacén. 2. El sistema muestra el stock de productos en el almacén seleccionado.
Secuencia Excepcional	Ninguna.

Tabla 3.12: Caso de Uso: *Actualizar datos almacén*

Título	Actualizar datos almacén
Resumen	El actor *Administrador* quiere modificar la información de un almacén.
Precondición	El actor *Administrador* está identificado en el sistema.
Postcondición	El sistema actualiza los datos de un almacén.
Secuencia Principal	1. El actor *Administrador* selecciona la opción modificar almacén. 2. El sistema permite al actor *Administrador* modificar la dirección y el teléfono de un almacén. 3. El actor *Administrador* modifica los datos y selecciona *Modificar*. 4. El sistema actualiza el almacén.
Secuencia Excepcional	2.1 El actor *Administrador* cancela y el caso de uso queda sin efecto.

Tabla 3.13: Caso de Uso: *Eliminar almacén*

Título	Eliminar almacén
Resumen	El actor *Administrador* quiere eliminar un almacén.
Precondición	El actor *Administrador* está identificado en el sistema.
Postcondición	El sistema elimina un almacén.
Secuencia Principal	1. El actor *Administrador* selecciona la opción eliminar almacén. 2. El sistema pide al actor *Administrador* confirmar la eliminación del almacén seleccionado. 3. El actor *Administrador* selecciona *Confirmar*. 4. El sistema elimina el almacén.
Secuencia Excepcional	2.1 El actor *Administrador* cancela y el caso de uso queda sin efecto.

Tabla 3.14: Caso de Uso: *Asignar producto a almacenes.*

Título	Asignar producto a almacenes.
Resumen	El actor *Administrador* quiere asignar un producto a varios almacenes.
Precondición	1. El actor *Administrador* está identificado en el sistema. 2. El actor *Administrador* ha seleccionado previamente un producto.
Postcondición	El sistema asigna un producto a los almacenes seleccionados.
Secuencia Principal	1. El actor *Administrador* selecciona asignar el producto a almacenes. 2. El sistema muestra al actor *Administrador* una lista con todos los almacenes que no tienen asignado el producto seleccionado. 3. El actor *Administrador* selecciona uno o varios almacenes y pulsa *Asignar*. 4. El sistema asigna el producto a los almacenes seleccionados.
Secuencia Excepcional	3.1 El actor *Administrador* cancela y el caso de uso queda sin efecto.

Tabla 3.15: Caso de Uso: *Asignar producto a tiendas.*

Título	Asignar producto a tiendas.
Resumen	El actor *Administrador* quiere asignar un producto a varias tiendas.
Precondición	1. El actor *Administrador* está identificado en el sistema. 2. El actor *Administrador* ha seleccionado previamente un producto.
Postcondición	El sistema asigna un producto a los almacenes seleccionados.
Secuencia Principal	1. El actor *Administrador* selecciona asignar el producto a almacenes. 2. El sistema muestra al actor *Administrador* una lista con todos los almacenes que no tienen asignado el producto seleccionado. 3. El actor *Administrador* selecciona uno o varios almacenes y pulsa *Asignar*. 4. El sistema asigna el producto a los almacenes seleccionados.
Secuencia Excepcional	3.1 El actor *Administrador* cancela y el caso de uso queda sin efecto.

Tabla 3.16: Caso de Uso: *Consultar Empleados.*

Título	Consultar Empleados.
Resumen	El actor *Administrador* quiere consultar los empleados de una tienda o de un almacén.
Precondición	El actor *Administrador* está identificado en el sistema.
Postcondición	El sistema muestra los empleados de la tienda o del almacén seleccionado.
Secuencia Principal	1. El actor *Administrador* selecciona una tienda o un almacén en la sección *Empleados*. 2. El sistema muestra una lista con los empleados de la tienda o el almacén seleccionado.
Secuencia Excepcional	Ninguna.

Tabla 3.17: Caso de Uso: *Eliminar Empleado.*

Título	Eliminar Empleado.
Resumen	El actor *Administrador* quiere borrar un empleado del sistema.
Precondición	El actor *Administrador* está identificado en el sistema.
Postcondición	El sistema ha eliminado el empleado seleccionado.
Secuencia Principal	1. El actor *Administrador* selecciona una tienda o un almacén en la sección *Empleados*. 2. El sistema muestra una lista con los empleados de la tienda o el almacén seleccionado. 3. El actor *Administrador* selecciona la opción *Eliminar* de un empleado. 4. El sistema consulta al actor *Administrador* si está seguro que desea eliminar el empleado seleccionado 5. El actor confirma. 6. El sistema elimina el empleado seleccionado.
Secuencia Excepcional	5.1 El actor *Administrador* cancela y el caso de uso queda sin efecto.

Tabla 3.18: Caso de Uso: *Actualizar Empleado.*

Título	Actualizar Empleado.
Resumen	El actor *Administrador* quiere actualizar la información de un empleado del sistema.
Precondición	El actor *Administrador* está identificado en el sistema.
Postcondición	El sistema ha actualizado el empleado seleccionado.
Secuencia Principal	1. El actor *Administrador* selecciona la opción *Modificar* de un empleado. 2. El sistema permite al actor *Administrador* modificar los datos de un empleado. 3. El actor *Administrador* modifica los datos deseados y selecciona *Actualizar*. 4. El sistema actualiza los datos del empleado seleccionado.
Secuencia Excepcional	3.1 El actor *Administrador* cancela y el caso de uso queda sin efecto.

Tabla 3.19: Caso de Uso: *Consultar Pedidos por estado.*

Título	Consultar Pedidos por estado.
Resumen	El actor *Administrador* quiere consultar los pedidos que se han realizados por estado.
Precondición	El actor *Administrador* está identificado en el sistema.
Postcondición	El sistema muestra todos los pedidos con el estado seleccionado.
Secuencia Principal	1. El actor *Administrador* selecciona el estado deseado y selecciona la opción *Consultar Pedidos*. 2. El sistema muestra todos los pedidos con el estado seleccionado.
Secuencia Excepcional	Ninguno.

Tabla 3.20: Caso de Uso: *Registrar Cliente*

Título	Registrar Cliente
Resumen	El actor *Cliente* quiere registrarse en el sistema.
Precondición	Ninguna.
Postcondición	El usuario queda registrado en el sistema.
Secuencia Principal	1. El actor *Cliente* selecciona la opción registrarse en el sistema. 2. El sistema solicita al actor *Cliente* su nombre de usuario y contraseña. 3. El actor *Cliente* introduce los datos y selecciona *Registrarse*. 4. El sistema registra al actor *Cliente*.
Secuencia Excepcional	2.1 El actor *Cliente* cancela y el caso de uso queda sin efecto. 4.1 El sistema rechaza el registro debido a que el nombre de usuario ya existe y muestra un mensaje.

Tabla 3.21: Caso de Uso: *Obtener Detalles Stock Talla Producto*

Título	Obtener Detalles Stock Talla Producto
Resumen	El actor *Cliente* quiere obtener los detalles de un producto.
Precondición	Ninguna.
Postcondición	El sistema obtiene detalles del stock del producto seleccionado.
Secuencia Principal	1. El actor *Cliente* selecciona un producto del catálogo. 2. El sistema muestra los detalles del stock del producto seleccionado.
Secuencia Excepcional	Ninguna.

Tabla 3.22: Caso de Uso: *Consultar Disponibilidad de la talla de un producto en tiendas físicas*

Título	Consultar Disponibilidad de la talla de un producto en tiendas físicas
Resumen	El actor *Cliente* quiere consultar la disponibilidad en tiendas físicas del producto seleccionado.
Precondición	El actor *Cliente* ha seleccionado previamente un producto y su talla.
Postcondición	El sistema muestra la disponibilidad de la talla de ese producto en una tienda física.
Secuencia Principal	1. El actor *Cliente* selecciona la opción consultar disponibilidad en tiendas físicas. 2. El sistema muestra las tiendas que tienen stock de ese producto. 3. El actor *Cliente* selecciona una tienda. 4. El sistema muestra si la talla del producto seleccionado está disponible en esa tienda.
Secuencia Excepcional	3.2 Si el producto no está asignado a ninguna tienda el sistema indica que no hay stock en tiendas.

Tabla 3.23: Caso de Uso: *Añadir productos al carro.*

Título	Añadir productos al carro.
Resumen	El actor *Cliente* quiere añadir productos al carro de la compra.
Precondición	Ninguna.
Postcondición	El sistema ha añadido un producto al carro de la compra.
Secuencia Principal	1. El actor *Cliente* selecciona un producto y una talla. 2. El sistema confirma la existencia de stock para ese producto y talla en la tienda online. 3. El actor *Cliente* selecciona *Añadir al carro.* 4. El sistema añade el producto al carro de la compra.
Secuencia Excepcional	2.1 El sistema notifica que el producto para la talla seleccionada no se encuentra disponible en la tienda online y el caso de uso queda sin efecto.

Tabla 3.24: Caso de Uso: *Modificar cantidad producto en carro.*

Título	Modificar cantidad producto en carro.
Resumen	El actor *Cliente* quiere modificar la cantidad de un producto en el carro de la compra.
Precondición	El carro de la compra no está vacío.
Postcondición	El sistema ha modificado la cantidad de un producto en el carro de la compra.
Secuencia Principal	1. El actor *Cliente* cambia la cantidad de un producto en el carro de la compra. 2. El sistema modifica el número de productos en el carro de la compra.
Secuencia Excepcional	Ninguna.

Tabla 3.25: Caso de Uso: *Eliminar producto del carro.*

Título	Eliminar producto del carro.
Resumen	El actor *Cliente* quiere eliminar un producto del carro de la compra.
Precondición	El carro de la compra no está vacío.
Postcondición	El sistema ha eliminado un producto del carro de la compra.
Secuencia Principal	1. El actor *Cliente* selecciona la opción *Eliminar* en un producto. 2. El sistema elimina el producto del carro de la compra.
Secuencia Excepcional	Ninguna.

Tabla 3.26: Caso de Uso: *Consultar Pedidos Realizados*

Título	Consultar Pedidos Realizados.
Resumen	El actor *Cliente Identificado* quiere consultar sus pedidos realizados.
Precondición	El actor *Cliente Identificado* está identificado en el sistema.
Postcondición	El sistema muestra sus pedidos realizados.
Secuencia Principal	1. El actor *Cliente Identificado* selecciona la opción *Consultar Pedidos*. 2. El sistema muestra en detalle todos los pedidos realizados por el actor *Cliente Identificado*. 3. El actor *Cliente Identificado* confirma el pedido. 4. El sistema registra el pedido y vacía el carro de la compra.
Secuencia Excepcional	3.1 El actor *Cliente Identificado* no está identificado en el sistema y se le ofrece registrarse. 3.2 El actor *Cliente Identificado* cancela y el caso de uso queda sin efecto.

Tabla 3.27: Caso de Uso: *Realizar Pedido.*

Título	Realizar Pedido.
Resumen	El actor *Cliente Identificado* quiere realizar un pedido de productos.
Precondición	Ninguna.
Postcondición	El sistema ha registrado un nuevo pedido para el cliente.
Secuencia Principal	1. El actor *Cliente Identificado* selecciona la opción *Checkout* en el carro de la compra. 2. El sistema muestra la factura del pedido y solicita confirmación. 3. El actor *Cliente Identificado* confirma el pedido. 4. El sistema registra el pedido y vacía el carro de la compra.
Secuencia Excepcional	3.1 El actor *Cliente Identificado* no está identificado en el sistema y se le ofrece registrarse. 3.2 El actor *Cliente Identificado* cancela y el caso de uso queda sin efecto.

Tabla 3.28: Caso de Uso: *Identificar Usuario*

Título	Identificar Usuario
Resumen	El actor *Usuario* quiere identificarse en el sistema.
Precondición	Ninguna.
Postcondición	El usuario queda identificado en el sistema.
Secuencia Principal	1. El actor *Usuario* selecciona la opción identificarse en el sistema. 2. El sistema solicita al actor *Usuario* su nombre de usuario y contraseña. 3. El actor *Usuario* introduce los datos y selecciona *Iniciar Sesión*. 4. El sistema identifica al actor *Usuario*.
Secuencia Excepcional	2.1 El actor *Usuario* cancela y el caso de uso queda sin efecto. 4.1 El sistema rechaza la identificación del actor *Usuario* y muestra un mensaje indicando que el nombre de usuario o la contraseña son incorrectos.

Tabla 3.29: Caso de Uso: *Consultar Categorías Catálogo*

Título	Consultar Categorías Catálogo
Resumen	El actor *Usuario* quiere obtener las categorías de productos.
Precondición	Ninguna.
Postcondición	El sistema muestra al actor *Usuario* la lista de categorías del catálogo.
Secuencia Principal	1. El actor *Usuario* selecciona la opción Categorías. 2. El sistema muestra una lista con todas las categorías del catálogo.
Secuencia Excepcional	Ninguna.

Tabla 3.30: Caso de Uso: *Consultar Subcategorías*

Título	Consultar Subcategorías
Resumen	El actor *Usuario* quiere obtener las subcategorías de una categoría.
Precondición	El actor *Usuario* ha seleccionado previamente la opción categorías.
Postcondición	El sistema muestra al actor *Usuario* la lista de subcategorías de la categoría seleccionada.
Secuencia Principal	1. El actor *Usuario* selecciona una Categoría. 2. El sistema muestra una lista con todas las subcategorías de la categoría seleccionada.
Secuencia Excepcional	Ninguna.

Tabla 3.31: Caso de Uso: *Consultar Productos Subcategoría*

Título	Consultar Productos Subcategoría
Resumen	El actor *Usuario* quiere obtener los productos de una subcategoría.
Precondición	El actor *Usuario* ha seleccionado previamente una subcategoría.
Postcondición	El sistema muestra al actor *Usuario* la lista de productos de la subcategoría seleccionada.
Secuencia Principal	1. El actor *Usuario* selecciona una Subcategoría. 2. El sistema muestra una lista con todos los productos de la subcategoría seleccionada.
Secuencia Excepcional	Ninguna.

Tabla 3.32: Caso de Uso: *Consultar existencias de productos en su tienda*

Título	Consultar existencias de productos en su tienda
Resumen	El actor *Empleado Tienda* necesita consultar las existencias de su tienda.
Precondición	El actor *Empleado Tienda* está identificado en el sistema.
Postcondición	El sistema obtiene la información sobre las existencias de su tienda.
Secuencia Principal	1. El actor *Empleado Tienda* selecciona la opción consultar existencias productos. 2. El sistema muestra información sobre el stock de los productos en su tienda.
Secuencia Excepcional	Ninguna.

Tabla 3.33: Caso de Uso: *Consultar existencias de productos en su almacén*

Título	Consultar existencias de productos en su almacén
Resumen	El actor *Empleado Almacén* necesita consultar las existencias de su almacén.
Precondición	El actor *Empleado Almacén* está identificado en el sistema.
Postcondición	El sistema obtiene la información sobre las existencias de su almacén.
Secuencia Principal	1. El actor *Empleado Almacén* selecciona la opción consultar existencias productos. 2. El sistema muestra información sobre el stock de los productos en su almacén.
Secuencia Excepcional	Ninguna.

Tabla 3.34: Caso de Uso: *Consultar Pedidos a almacén.*

Título	Consultar Pedidos a almacén.
Resumen	El actor *Empleado Almacén* quiere consultar los pedidos que han sido realizados a su almacén.
Precondición	El actor *Empleado Almacén* está identificado en el sistema.
Postcondición	El sistema muestra todos los pedidos asignados al almacén del actor *Empleado Almacén.*
Secuencia Principal	1. El actor *Empleado Almacén* selecciona la opción consultar pedidos. 2. El sistema muestra todos los pedidos asignados al almacén del actor *Empleado Almacén.*
Secuencia Excepcional	Ninguno.

Tabla 3.35: Caso de Uso: *Cambiar estado pedido.*

Título	Cambiar estado pedido.
Resumen	El actor *Empleado Almacén* quiere cambiar el estado de unos de los pedidos que se ha realizado a su almacén a *Preparado.*
Precondición	El actor *Empleado Almacén* está identificado en el sistema.
Postcondición	El sistema cambia el estado del pedido a *Preparado.*
Secuencia Principal	1. El actor *Empleado Almacén* selecciona el pedido que desea cambiar de estado. 2. El sistema muestra las opciones de estado disponibles. 3. El actor *Empleado Almacén* selecciona una de las opciones de estado y pulsa confirmar. 4. El sistema cambia el estado del pedido seleccionado.
Secuencia Excepcional	3.1 El actor *Administrador* cancela y el caso de uso queda sin efecto.

3.3. MODELO DE DOMINIO

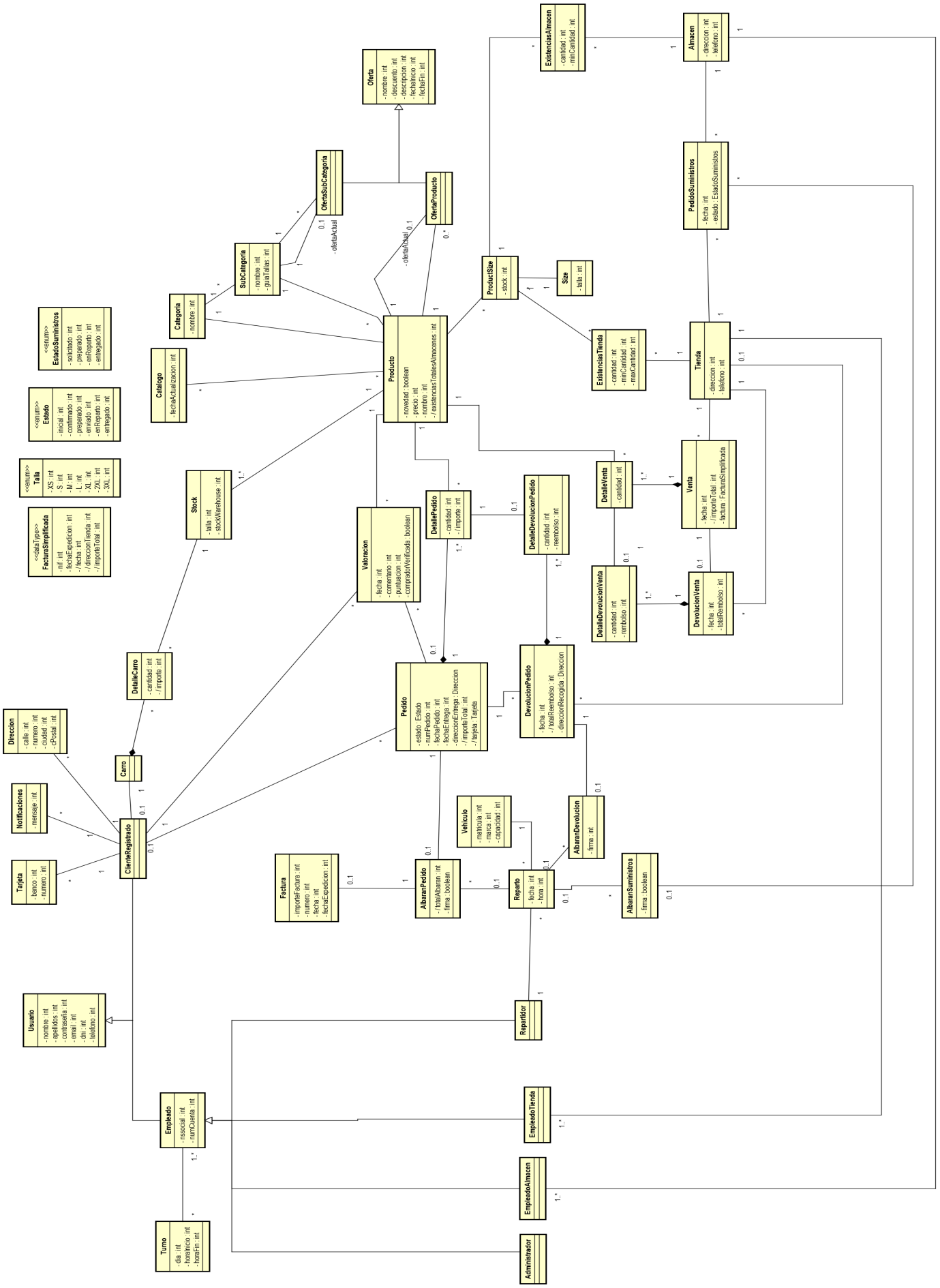

Figura 3.6: Modelo de dominio en análisis.

CAPÍTULO 4

DISEÑO DE LA ARQUITECTURA DEL SISTEMA

Este capítulo aborda la descripción detallada del diseño de un sistema basado en una arquitectura de microservicios que se ha llevado a cabo para la implementación de esta aplicación cloud.

La manera de utilizar la información obtenida del sistema durante la fase de análisis para el diseño de un sistema basado en la arquitectura de microservicios difiere al tradicional diseño de un monolito. En una arquitectura monolítica el software se estructura de tal forma que todos los aspectos funcionales y toda la información quedan acoplados dentro de un mismo programa. Este programa es desplegado en un único servidor.

En cambio, en una arquitectura de microservicios, el software del sistema queda distribuido en un conjunto de pequeños servicios, débilmente acoplados, aislados lo máximo posible entre sí, que se ejecutan de manera independiente y autónoma, alojados en servidores diferentes. Cada microservicio desempeña una función específica del sistema y puede implementarse con una tecnología diferente.

La comunicación entre microservicios se realiza únicamente a través de APIs. Cada microservicio cuenta con un sistema de almacenamiento propio, lo que evita sobrecargas y la caída total de la aplicación, pero que requiere un diseño que aborde el problema de la consistencia de datos distribuida y la lógica distribuida.

A lo largo de este capítulo se explicarán los principales patrones de diseño que han sido necesarios llevar a cabo para trasformar la información obtenida durante la fase de análisis y diseñar un sistema basado en una arquitectura de microservicios.

4.1. DOMAIN DRIVEN DESIGN (DDD)

DDD es una técnica de modelado de software utilizada para construir aplicaciones de software complejas que están centradas en un modelo de dominio orientado a objetos. Esta técnica es bastante diferente al tradicional enfoque de modelado, que crea un único modelo para la totalidad del sistema empresarial, tal y como se ha representado en el modelo de domino en análisis, Figura 3.6. En dicho modelo hay una única definición para cada entidad de negocio, como por ejemplo Almacén, Producto, etc.

El problema con este tipo de modelado es que tener satisfecha a la totalidad de la organización requiere un trabajo enorme. Además, desde la perspectiva de cada parte de la organización, el modelo alcanzaría tal complejidad que dificultaría su comprensión, ya que cada parte de la organización puede usar diferentes términos para los mismos conceptos del negocio. Por ejemplo, no es el mismo concepto de un Producto el que tiene un Cliente que el que tiene el empleado de un Almacén. DDD evita este problema definiendo múltiples modelos de dominio, cada uno de ellos con un alcance explícito.

DDD consta de dos conceptos muy útiles cuando modelamos una aplicación basada en microservicios: *subdominios* y *contexto ligado (bounded context).*

DDD define un modelo de dominio separado en *subdominios*. Cada *subdominios* es una parte del dominio y son identificados analizando el negocio e identificando las diferentes áreas de especialización.

DDD denomina al alcance del modelo de dominio *bounded context*. En una arquitectura de microservicios, cada límite del dominio es un servicio o un conjunto de servicios.

Para realizar el diseño del dominio de la arquitectura de microservicios que se utilizará para la implementación de la aplicación, se ha aplicado DDD y definido un microservicio por cada *subdominio*. Una vez estudiada y aplicada esta técnica con el apoyo de la información obtenida durante la fase de análisis, se ha desglosado el modelo de dominio en *subdominios* que definen a los siguientes microservicios:

- Microservicio *Authentication*: se encargará de gestionar el almacenamiento y autenticación de los usuarios del sistema generando JWTs (JSON Web Tokens).
- Microservicio *Customers*: se encargará de gestionar los clientes registrados en el sistema y su carro de la compra.
- Microservicio *Catalog*: se encargará de la gestión de los productos del catálogo, de sus categorías y ofertas.
- Microservicio *Inventory*: se encargará de la gestión del inventario de productos en tiendas y almacenes.
- Microservicio *Employees*: se encargará de la gestión de los empleados registrado en el sistema.
- Microservicio *Sales*: se encargará de la gestión de las ventas online y en tiendas físicas.

En la Figura 4.1 se muestra el modelo de dominio del sistema desglosado en *subdominios* tras aplicar DDD. Se puede observar a simple vista las diferencias con el modelo de dominio inicial obtenido en la fase de análisis, Figura 3.6. Cada *subdominio* representa el modelo de un Microservicio del sistema.

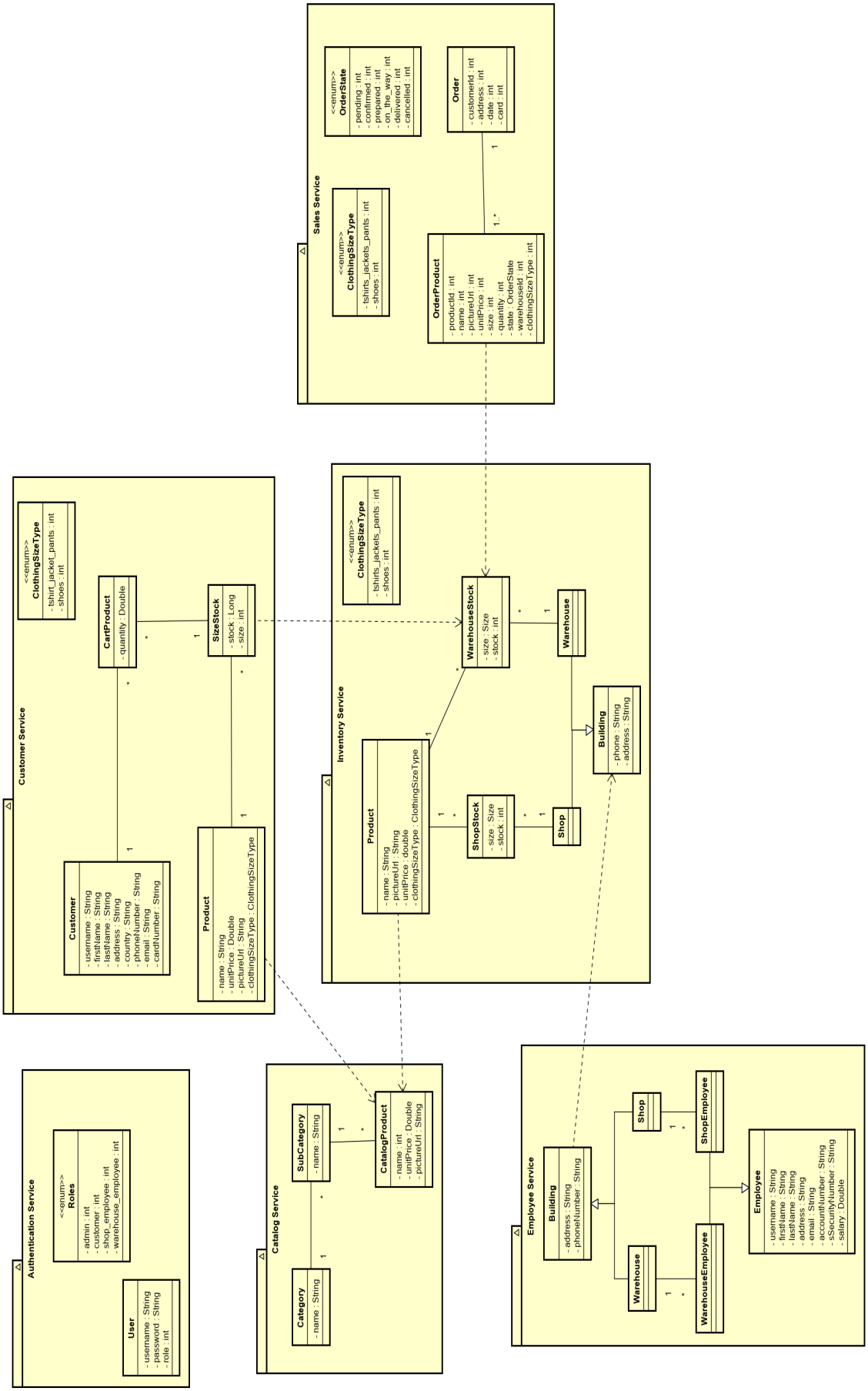

Figura 4.1: Modelo de dominio en diseño.

Se puede ver en el modelo de dominio en diseño Figura 4.1 que hay clases que están replicadas en *subdominios* distintos, como por ejemplo, la clase Producto del Microservicio Inventory que depende de la clase Producto del Microservicio Catalog. Estas clases representan a un Producto en una parte de la organización diferente, el Catálogo y el Inventario. Cada una de ellas contiene la información necesaria para satisfacer las necesidades de la funcionalidad del Microservicio en el que se encuentra.

Descomponer una aplicación en microservicios implica varios retos a los que hay que enfrentarse:

- Latencia de red debido al acoplamiento que se crea en las comunicaciones entre microservicios.
- Reducción de la disponibilidad debido a comunicaciones síncronas entre los microservicios, si un microservicio sufre una caída, todos los microservicios que dependen de él verán afectada su funcionalidad.
- Mantener la consistencia de datos a través de los microservicios, cada microservicio dispondrá de una base datos propia de almacenamiento y no podrá acceder a la base de datos de otro microservicio.
- La obtención de una vista consistente de los datos en un sistema en el que no existe una base de datos centralizada.

4.2. RESISTENCIA A FALLOS Y DISPONIBILIDAD

Lidiar con fallos inesperados es uno de los problemas más difíciles de resolver, especialmente en un sistema distribuido.

Un microservicio necesita ser resistente a fallos y poder reiniciarse con frecuencia sin que suponga un problema para la totalidad del sistema. Si un microservicio falla o es reiniciado no se debe producir ninguna perdida de datos en el sistema y estos datos deben seguir siendo consistentes.

Un sistema cloud debe tolerar los fallos e intentar recuperarse automáticamente de ellos. Por ejemplo, en el caso de un fallo de red, las apps clientes o los servicios clientes deben de tener una estrategia de recuperación reenviando mensajes o reintentando las solicitudes.

En cuanto a la disponibilidad, se debe diseñar un sistema en el que los microservicios se encuentren lo más aislados posible e independientes entre sí. Aislar microservicios evitará que la funcionalidad de un microservicio se vea afectada por un fallo en el microservico del que depende.

Desplegar los microservicios en contenedores permite crear fácilmente réplicas del microservicio, lo que evita problemas de sobrecarga.

Estas características se han tenido en cuenta a la hora de diseñar este sistema basado en una arquitectura de microservicios y se volverá a hablar de ellas a lo largo de esta memoria.

4.3. DISEÑO DE LA API GATEWAY

La API Gateway (Puerta de enlace) proporciona a los clientes un único punto de entrada al sistema. Su principal función es la gestión del tráfico de entrada a los microservicios a través del enrutado y filtrado de solicitudes, aunque puede implementar otras funciones como la autenticación. También puede realizar la función de traductor de protocolos, por ejemplo, comunicarse con el exterior de la aplicación a través de REST API y con el interior a través de una combinación de APIs REST y gRPC. En el caso de la aplicación desarrollada en este trabajo la comunicación de la API Gateway con el exterior y el interior de la aplicación es a través de APIs REST.

En la Figura 4.2 se muestra el diseño del enrutado que la API Gatway realiza a los microservicios.

La API Gateway está compuesta por un Service Discovery, un Load Balancer (Balanceador de carga) y un Circuit Breaker (cortacircuitos) que complementan su funcionalidad.

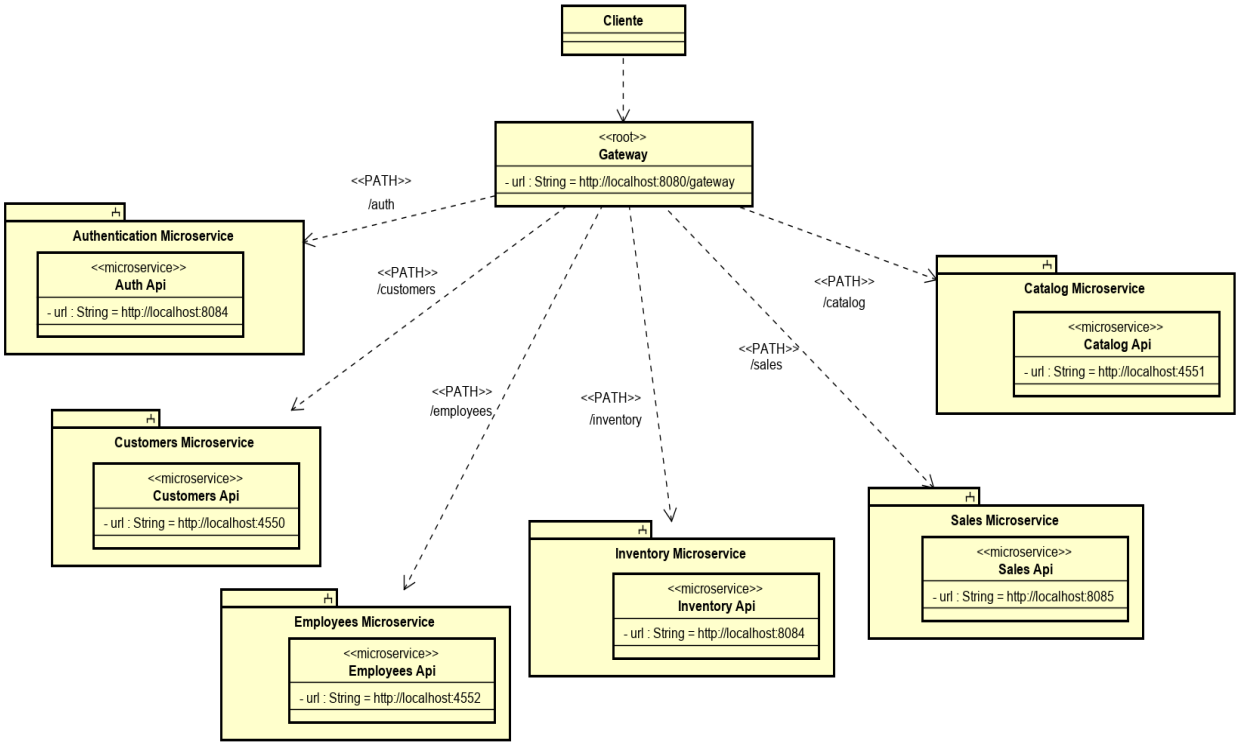

Figura 4.2: Diseño de la API Gateway.

4.3.1. Service Discovery

Su componente clave es un *service registry (servidor de registros)* que consta de una base de datos que almacena las ubicaciones de red de las instancias de servicio de una aplicación.

El *service discovery (descubridor de servicios)* actualiza el *service registry* cada vez que una instancia del servicio es parada o iniciada. Cuando un cliente invoca a un servicio el *service discovery* consulta el registro de servicios para obtener una lista de las instancias del servicio disponibles y enrutar la solicitud a la instancia correspondiente. Gracias a este patrón de diseño el cliente no necesita saber a que servidor enviar la petición.

4.3.2. Load Balancer

Una arquitectura basada en microservicios debe diseñarse para tener una gran escalabilidad y fiabilidad. Una forma de mejorar la escalabilidad y la fiabilidad es ejecutar múltiples instancias de un mismo servicio. Si una instancia falla o esta sobrecargada, el *load balancer* enrutará la solicitud a otra instancia.

4.3.3. Circuit Breaker

En un sistema distribuido, siempre que un servicio hace una petición síncrona a otro servicio, hay riesgo de que se produzca un fallo parcial. Debido a que el cliente y el servidor son procesos separados, es posible que un servicio no pueda responder de manera oportuna a la solicitud de un cliente. El servicio podría estar caído o el servicio podría estar sobrecargado y respondiendo muy lento a las solicitudes. Debido a esto, el cliente es bloqueado esperando una respuesta, el riesgo de esto es que el fallo podría causar un fallo en cascada y afectar a los clientes del cliente afectado.

Para diseñar aplicaciones robustas es necesario una combinación de mecanismos de timeouts, de limitadores de acumulación de fallos y de seguimiento de solicitudes exitosas y fallidas, de tal modo

que si se excede de un umbral, el *circuit breaker* rechazará de forma inmediata cualquier solicitud hasta pasado un periodo de tiempo.

4.4. CONSISTENCIA DE DATOS EN ARQUITECTURAS DE MICROSERVICIOS

Cada microservicio del sistema tiene su propia base de datos. Los datos almacenados en la base de datos del microservicio son privados y solo pueden ser accedidos a través de las APIs del microservicio. Esto supone un reto de a la hora de implementar procesos de negocio ento-to-end mientras se mantiene la consistencia a través de múltiples microservicios.

Tomando como ejemplo la aplicación desarrollada, el microservicio *Catalog* contiene la información de todos los Productos. El microservicio *Inventory* gestiona los datos temporales sobre el inventario de productos en tiendas y almacenes. Cuando el precio o el nombre de un producto cambia en el microservicio *Catalog*, este producto también debe de ser actualizado en el microservicio *Inventory*.

En una hipotética versión monolítica de esta aplicación, una actualización de un producto del catálogo sería una transacción inmediata en la tabla ProductInventory.

Sin embargo, en una aplicación basada en microservicios, la tabla ProductCatalog y la tabla ProductInventory son propiedad de sus respectivos microservicios. Ningún microservicio debe acceder nunca a la base de datos propiedad de otro microservicio de forma directa a través de queries, como se muestra en la Figura 4.3.

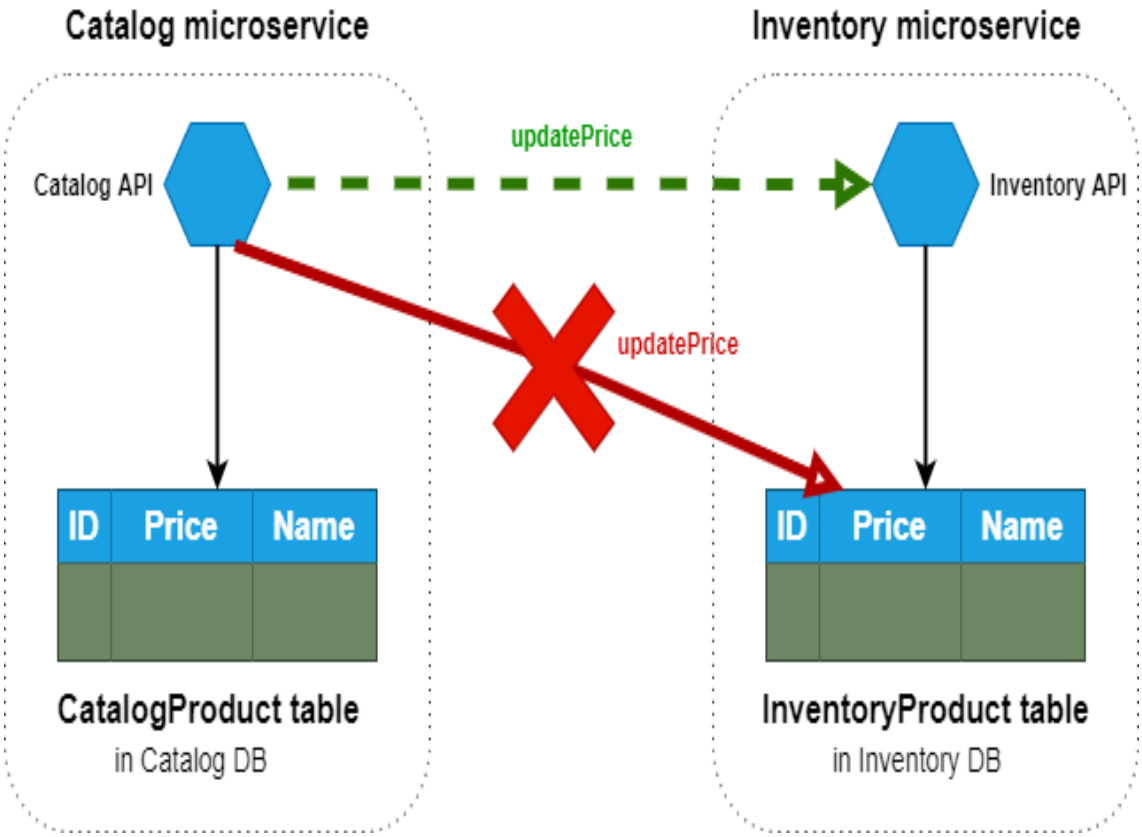

Figura 4.3: Un microservicio no puede tener acceso directo a la base de datos de otro microservicio.

El microservicio *Catalog* no debe actualizar la tabla InventoryProduct directamente. Para actualizar el microservice *Inventory*, el microservicio *Catalog* utilizará un tipo de consistencia futura basada en

una comunicación asíncrona entre microservicios, mediante la integración de eventos (comunicación basada en mensaje y eventos), tal y como se ha diseñado en la aplicación presentada en este trabajo.

En la mayoría de los escenarios en los que se utiliza una arquitectura de microservicios se demanda un sistema con una gran disponibilidad y escalabilidad, sacrificando a cambio una alta consistencia. Estos sistemas se deben desarrollar utilizando técnicas que permitan trabajar con una consistencia débil.

El reto de mantener la consistencia de datos entre los microservicios, está directamente relacionado con la cuestión de como propagar los cambios a través de múltiples microservicios cuando ciertos datos necesitan estar replicados en aquellos microservicios que los necesitan. Por ejemplo, cuando es necesario tener el precio o el nombre de un producto en el microservicio *Catalog* y en el microservicio *Inventory*.

La solución que se ha utilizado para resolver este problema es usar técnicas de consistencia futura o eventual, implementando patrones como Event-Driven (comunicación asíncrona manejada por eventos) e implementando un sistema Publish/Subscribe. Este tema de la comunicación entre microservicios será abordado a continuación.

4.5. DISEÑO DE LA COMUNICACIÓN ENTRE MICROSERVICIOS

Uno de los grandes retos que supone una arquitectura de microservicios es la comunicación entre los microservicios. A la hora de diseñar, se deben de analizar los diferentes estilos de comunicación según el nivel de acoplamiento que los microservicios deben de tener. Dependiendo del nivel de acoplamiento, cuando un fallo ocurre, el impacto en el sistema variará significativamente.

Los sistemas basados en una arquitectura de microservicios, se encuentran distribuidos en múltiples servicios alojados en diferentes servidores. Estos servicios pueden fallar, produciendo fallos parciales o fallos mayores que afectarían de forma crítica a la totalidad del sistema.

Una de los enfoques más populares debido a su simplicidad es implementar la comunicación entre microservicios síncrona basada en APIs REST, Figura 4.4.

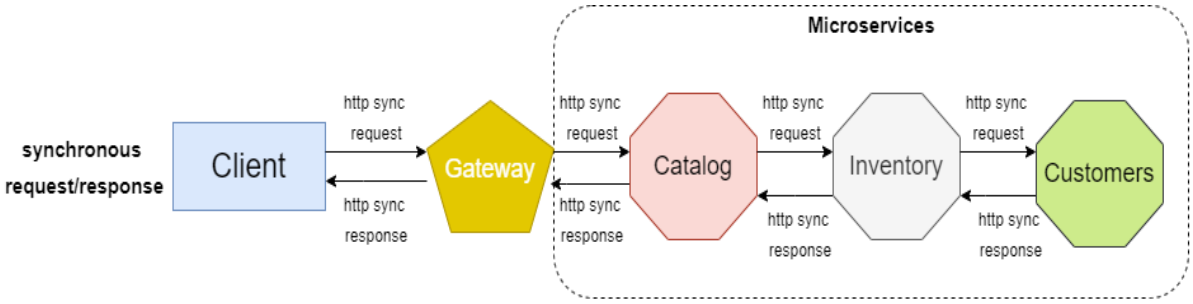

Figura 4.4: Comunicación entre microservicios síncrona basada en APIs REST.

Este enfoque puede ser problemático si creamos cadenas largas de peticiones HTTP síncronas entre los microservicios:

- Bloqueos entre microservicios y elevadas latencias. Debido a la naturaleza síncrona de HTTP, la solicitud original no obtendrá una respuesta hasta que todas las solicitudes internas hayan finalizado. Esto tendría un impacto negativo sobre el rendimiento de la aplicación que empeoraría con el escalado de la aplicación.
- Acoplamiento entre microservicios con HTTP. Los servicios del negocio de un microservicio, no debe de estar acoplado a los servicios del negocio de otro microservicio. Idealmente un microservicio no debería de conocer la existencia de ningún otro microservicio.

- Mantener la consistencia de datos a través de los microservicios, cada microservicio dispondrá de una base datos propia de almacenamiento y no podrá acceder a la base de datos de otro microservicio.
- Fallos en cualquier otro microservicio. En una cadena de microservicios enlazados por peticiones HTTP, cuando cualquier microservico de la cadena falla, todos los microservicios de la cadena fallarán.

La comunicación entre microservicios de la aplicación desarrollada en este trabajo se ha diseñado teniendo en cuenta los riesgos más comunes de este tipo de sistemas distribuidos.

Se ha diseñado una comunicación entre microservicios, de tal modo que los microservicios realicen sus funciones de la forma más autónoma e independiente posible, reduciendo lo máximo posible las comunicaciones entre microservicios. Para ellos, se ha implementado un estilo de comunicación entre microservicios principalmente asíncrona, Figura 4.5, siguiendo los patrones de diseño y utilizando los protocolos de comunicación que se comentarán a continuación.

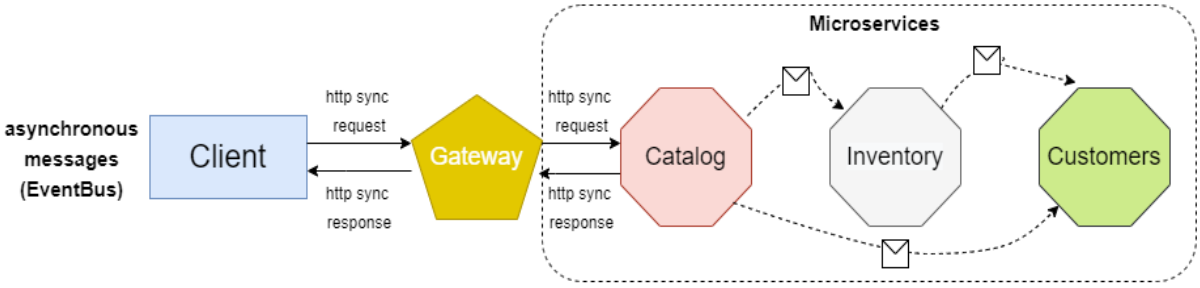

Figura 4.5: Comunicación entre microservicios basada en mensajería asíncrona.

4.5.1. Protocolos de comunicación

El diseño de la comunicación entre microservicios de la aplicación presentada en este trabajo está orientada a reducir la cantidad de comunicaciones entre los microservicios lo máximo posible y a propagar los datos de forma asíncrona. Pero en algunos casos aislados puede darse la necesidad de una comunicación síncrona entre microservicios. En esta aplicación se han implementado dos tipos de protocolos:

- **HTTP (Hypertext Transfer Protocol)**: permite una comunicación mediante solicitud/respuesta síncrona. Se implementará como APIs REST y su principal utilización será en la comunicación entre el cliente y los microservios a través de la API Gateway. El cliente debe de recibir una respuesta del sistema de forma síncrona y este protocolo es ideal. También en la comunicación entre microservicios, puede darse el caso de que un microservicio necesite una respuesta síncrona de otro microservicio, este protocolo sería una opción válida.
- **AMQP (Advanced Message Queuing Protocol)**: permite una comunicación solicitud/respuesta asíncrona y síncrona, además de estar diseñado para utilizarse en un sistema *publish/subcribe*. Se implementará mediante un bus de eventos. Será el principal protocolo de comunicación entre microservicios, debido a su tipo de comunicación asíncrona y a que encaja perfectamente en un sistema publish/subscribe, lo que lo hace ideal para la propagación de datos entre microservicios.

Aunque implementar un bus de eventos respaldado por AMQP es más versátil al permitir un mayor número de métodos de comunicación, además de la posibilidad de persistir los mensajes, implementar APIs REST respaldadas por HTTP es mucho más sencillo y requiere una curva de aprendizaje mucho menor que implementar un bus de eventos.

4.5.2. Comunicación asíncrona basada en mensajes

Como se mencionó en apartados anteriores, las entidades de los modelos como Product, Warehouse, Shop... pueden tener significado diferente en microservicios diferentes. Esto significa que ciertos datos de las entidades replicadas por los microservicios deben de ser actualizadas por el microservicio propietario de la información.

La solución que se implementará para propagar los cambios a través de múltiples microservicios será una consistencia de datos eventual y una comunicación manejada por eventos (event-driven) basada en mensajes asíncronos.

Al utilizar mensajería, los procesos se comunican intercambiando mensajes de forma asíncrona. El cliente envía una solicitud a un servicio enviando un mensaje, si el servicio necesita responder al cliente, envía un mensaje diferente de respuesta al cliente. El cliente asume que la respuesta no será inmediata o que no recibirá respuesta.

Un mensaje está compuesto por una cabecera y un cuerpo. Los mensajes serán enviados mediante el protocolo AMQP.

En una comunicación manejada por eventos (event-driven), un microservicio publica eventos en un bus de eventos cuando algo cambie en su dominio, varios microservicios se subscriben a esos eventos para ser notificados de forma asíncrona siempre que se realice una publicación de dichos eventos. Este sistema *publish/subscribe* será implementado usando un bus de eventos.

4.5.3. Event Sourcing simplificado

Cuando se publican eventos a través de un sistema distribuido de mensajería como un bus de eventos, existe el problema de actualizar de forma atómica la base de datos del microservicio y de publicar el evento. Se debe de evitar el caso en el que por algún motivo se produzca un fallo al persistir los datos en la base de datos del microservicio y estos datos si que se publiquen en el bus de eventos o viceversa. Esto dejaría el sistema en un estado inconsistente.

La solución que se llevará a cabo será implementar una simplificación del patrón *Event Sourcing* combinado con una tabla de base de datos transaccional. Se usarán estados en los eventos como "listo para publicar", esto eventos se almacenarán en una tabla llamada IntegrationEventLog (Figura 4.6) y se actualizará su estado según vaya progresando en sus fases desde que se crea el evento hasta que se publica.

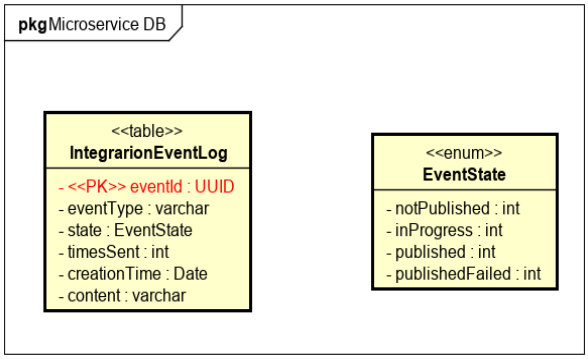

Figura 4.6: Diseño de la tabla IntegrationEventLog que se usará para implementar una simplificación del patrón Event Sourcing.

Cada microservicio tendrá su propia tabla IntegrationEventLog en su base de datos relacional. Esta tabla funciona como un resguardo para lograr la atomicidad, de modo que se persistan los eventos en las mismas transacciones que se persisten los datos de su dominio.

Si la publicación de un evento falla, los datos no serán inconsistentes, ya que el evento seguirá en la tabla IntegrationEventLog con el estado "listo para publicar". Este evento se reintentará publicar cuando se reinicie el microservicio o el bus de eventos.

Con este enfoque se almacenarán los eventos en la tabla IntegrationEventLog únicamente en los microservicios que vayan a publicar el evento (microservicios productores del evento).

4.5.4. Diagramas de comunicación

En este apartado se explicará de manera resumida y con ayuda de diagramas de secuencia como ejemplo, el diseño de la comunicación entre microservicios que se ha realizado.

- **Publish/Subscribe**:

Patrón de mensajería asíncrono que se ha utilizado para propagar los cambios en los datos del dominio en los microservicios en los que se encuentren replicados.

El microservicio *Publisher* publica un evento en el message broker con una filosofía de envía y olvida. Los microservicios subscritos a ese evento (*Subscribers*) recibirán los datos del evento de forma asíncrona. El microservicio *Publisher* es el propietario de los datos que publica, estos datos no deben de ser modificados por ningún microservicio *Subscriber*. En los diagramas de secuencia siguientes se muestra cómo se han diseñado las comunicaciones para la aplicación usando este patrón.

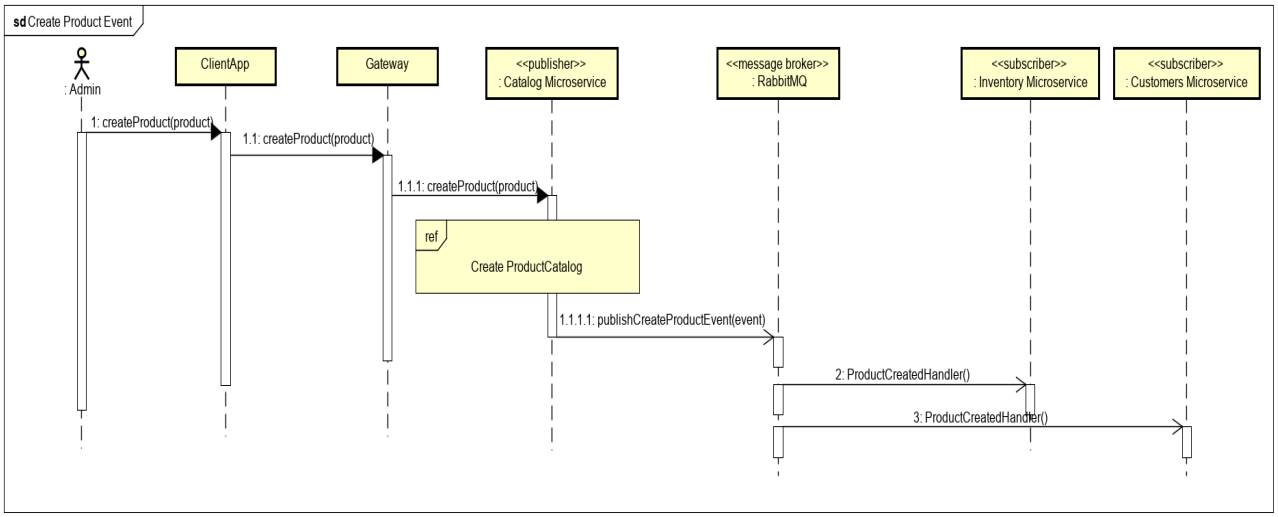

Figura 4.7: Comunicación entre microservicios que se produce cuando un Administrador crea un producto en el Catálogo.

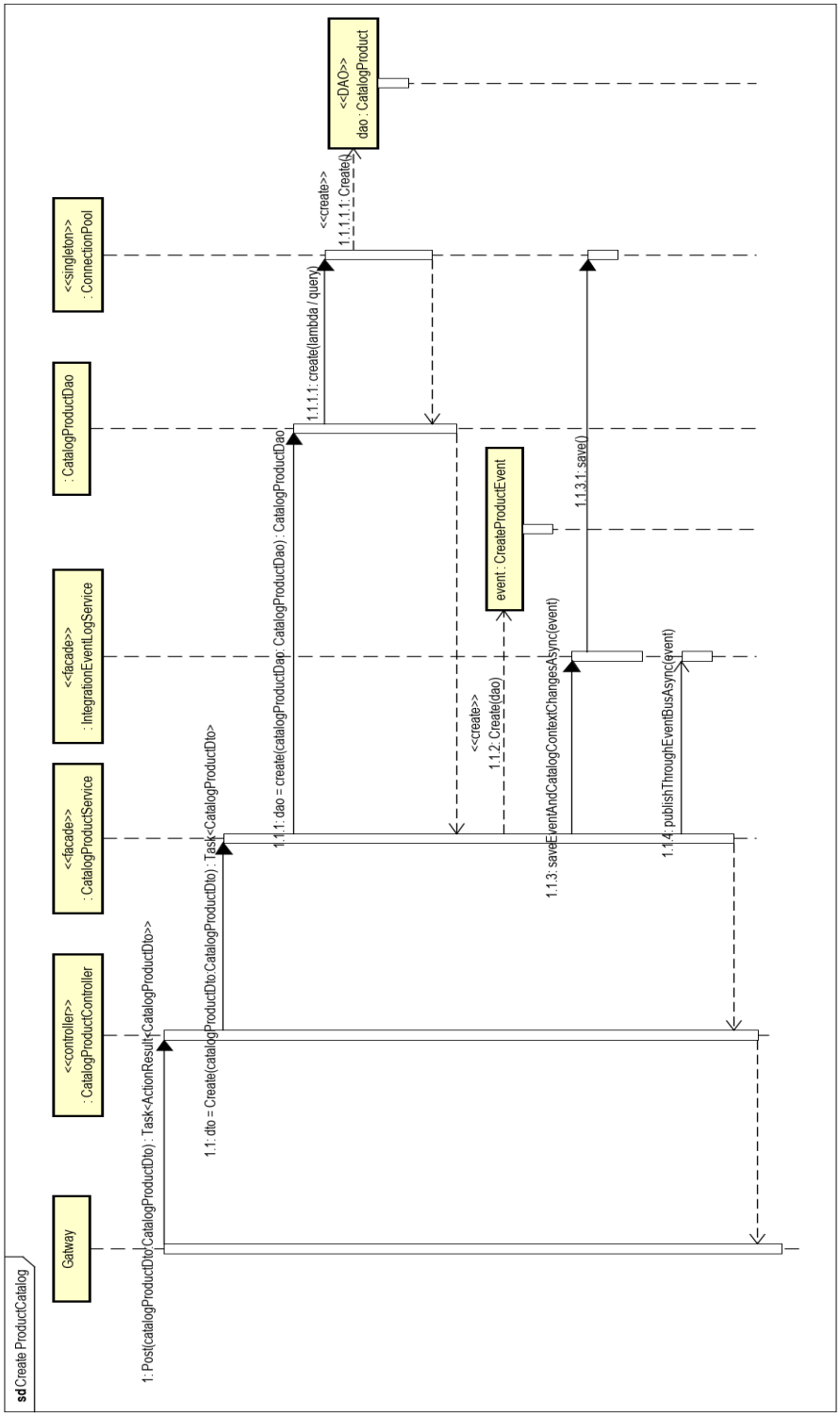

Figura 4.8: Diagrama de secuencia del microservicio *Catalog* al crear un producto.

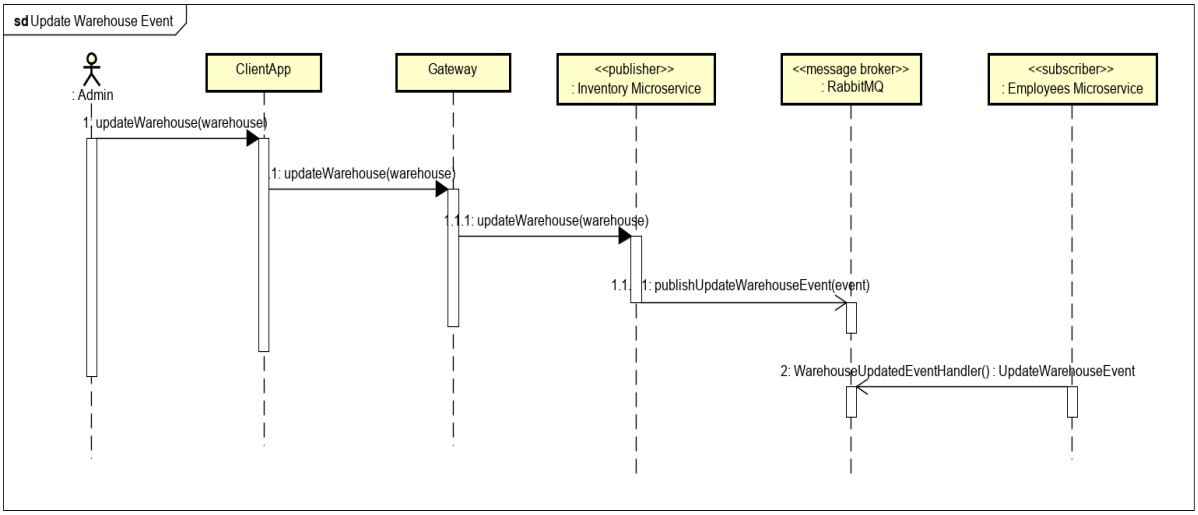

Figura 4.9: Comunicación entre microservicios que se produce cuando un Administrador actualiza un Almacén.

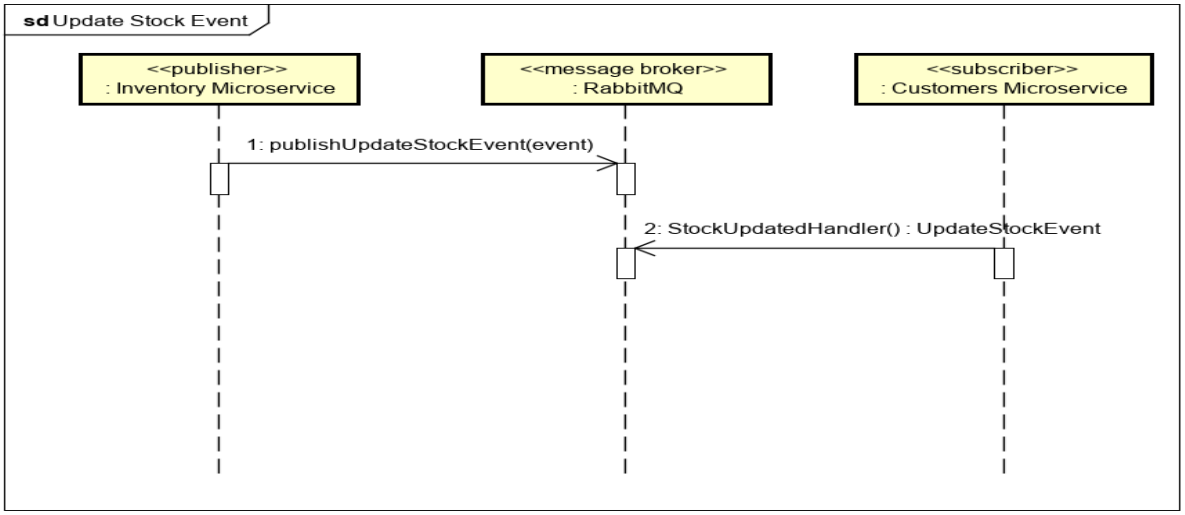

Figura 4.10: Comunicación entre microservicios que se produce cuando se actualiza el stock de un producto.

- **Request/Response asíncrono**:

Método de comunicación punto a punto. El microservicio cliente envía una solicitud que es recibida por el microservicio servidor de forma asíncrona, el microservicio servidor procesa la solicitud y envía una respuesta al cliente que también será recibida de forma asíncrona.

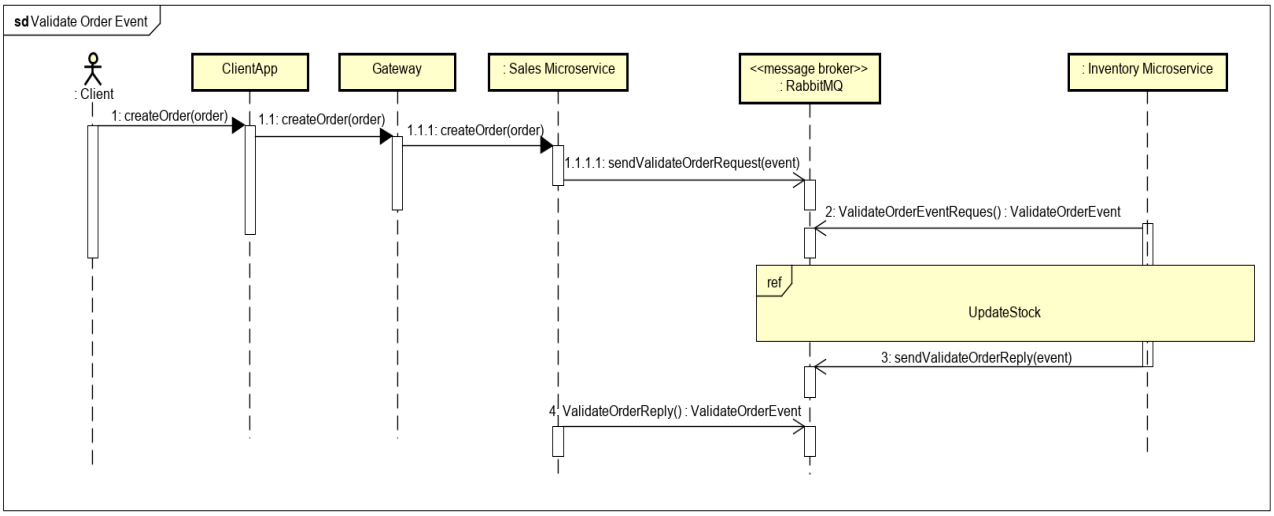

Figura 4.11: Comunicación entre microservicios que se produce cuando un Cliente realiza un pedido.

4.6. DISEÑO DE LA AUTENTICACIÓN DEL SISTEMA

Las cuentas de usuario serán propias del sistema y la autenticación a través de los microservicios se realizará mediante JWT (JSON Web Token).

Con el fin de tener una mayor flexibilidad y control en el proceso de autenticación se implementará un microservicio de autenticación propio. Este microservicio tendrá como función principal gestionar la autenticación del sistema generando tokens de acceso que propagarán la identidad y los privilegios del usuario identificado a través de los microservicios.

Con el fin de asegurar la seguridad del sistema los tokens de acceso generados por el microservicio de autenticación serán firmados y cifrados, esto se realizará siguiendo los siguientes estándares:

- **J**SON Web Signature (JWS): estándar definido en RFC 7515. JWS es usado para firmar digitalmente un token de acceso, es una forma de asegurar la integridad de una información muy serializable con la certeza de que dicha información no ha sido modificada desde que el token fue firmado. Su información puede ser leída por un simple decodificador Base64. No incluye cifrado, pero está diseñado para trabajar con cifrado. La firma del token se realizará con el algoritmo asimétrico RS256 usando una clave pública.
- **J**SON Web Encryption (JWE): estándar definido en RFC 7516. JWE es usado para cifrar un token de acceso, este token será completamente opaco para el cliente que lo usa como medio de autenticación y autorización. El cifrado del token se realizará con el algoritmo HS256 usando una clave privada.

En la Figura 4.12 podemos ver una representación visual del diseño de la autenticación del sistema.

Cuando un cliente inicia sesión, el microservicio authentication genera un token firmado y cifrado que se almacena temporalmente en la aplicación cliente. Cuando la aplicación cliente realice cualquier solicitud al *back end*, este token de acceso será enviado en la cabecera de la solicitud http. La API Gateway descifra y valida el token cifrado (JWE), si todo es correcto enruta la solicitud con el token firmado (JWS) al microservicio correspondiente donde también será validado.

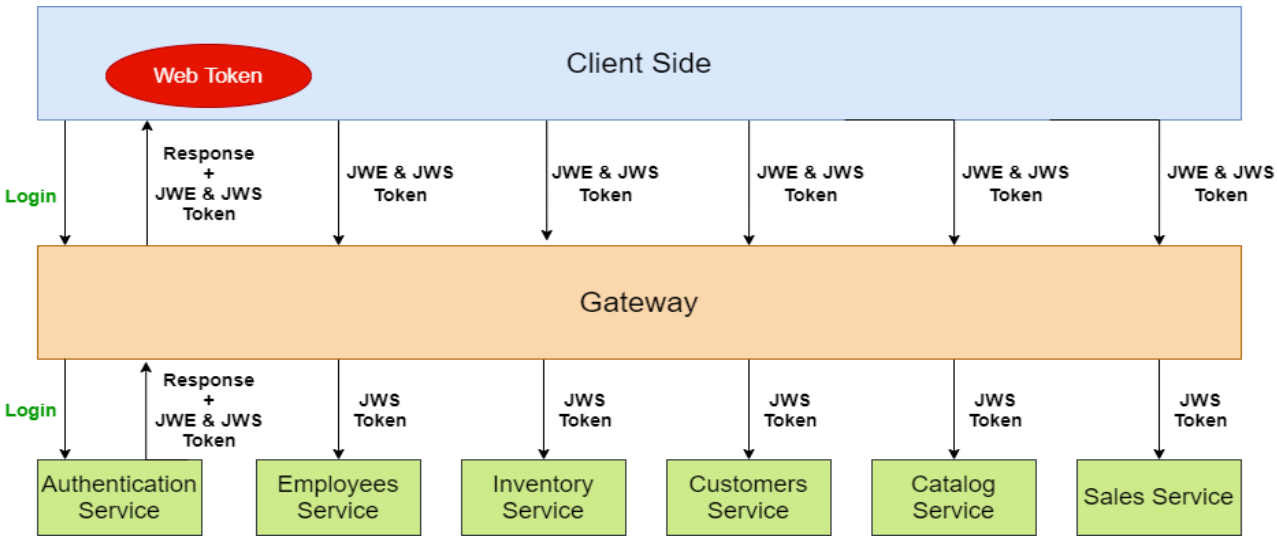

Figura 4.12: Diseño de la autenticación del sistema con JWE y JWS.

4.7. DISEÑO DE LA ARQUITECTURA *FRONT END*

La parte *front end* del sistema será una aplicación web implementada en *Angular*. Esta aplicación web proporcionará la interfaz de usuario del sistema.

La arquitectura de una aplicación angular se basa en bloques que son organizados en *módulos*. Los *módulos* agrupan el código según su funcionalidad, el conjunto de estos *módulos* definen la aplicación Angular. Un *módulo* contiene otros *módulos* o contiene *componentes*. Toda aplicación angular tiene un módulo raíz que permite el arranque de la aplicación y que está compuesto normalmente por más *módulos* con diferentes funcionalidades.

Los *componentes* contienen vistas que pueden ser modificadas o seleccionadas en función de su lógica de negocio y los datos del programa. Estos *componentes* usan *servicios* que proporcionan la lógica de negocio que no está relacionada directamente con la vista (interactúan con *back end*). Estos *servicios* pueden ser usados por los componentes mediante *inyección de dependencias*, haciendo el código modular, reusable y eficiente.

El conjunto de *componentes* define las vistas de la aplicación. Angular proporciona servicios de enrutado que permite definir las rutas de navegación entre las vistas y navegar entre ellas mediante un navegador web de forma bastante sofisticada.

Las *pipes* permite a las vistas adaptar los datos mostrados para facilitar la comprensión del usuario y mejorar así su experiencia mejorando.

La Figura 4.13 se muestra como están relacionadas las piezas básicas de Angular.

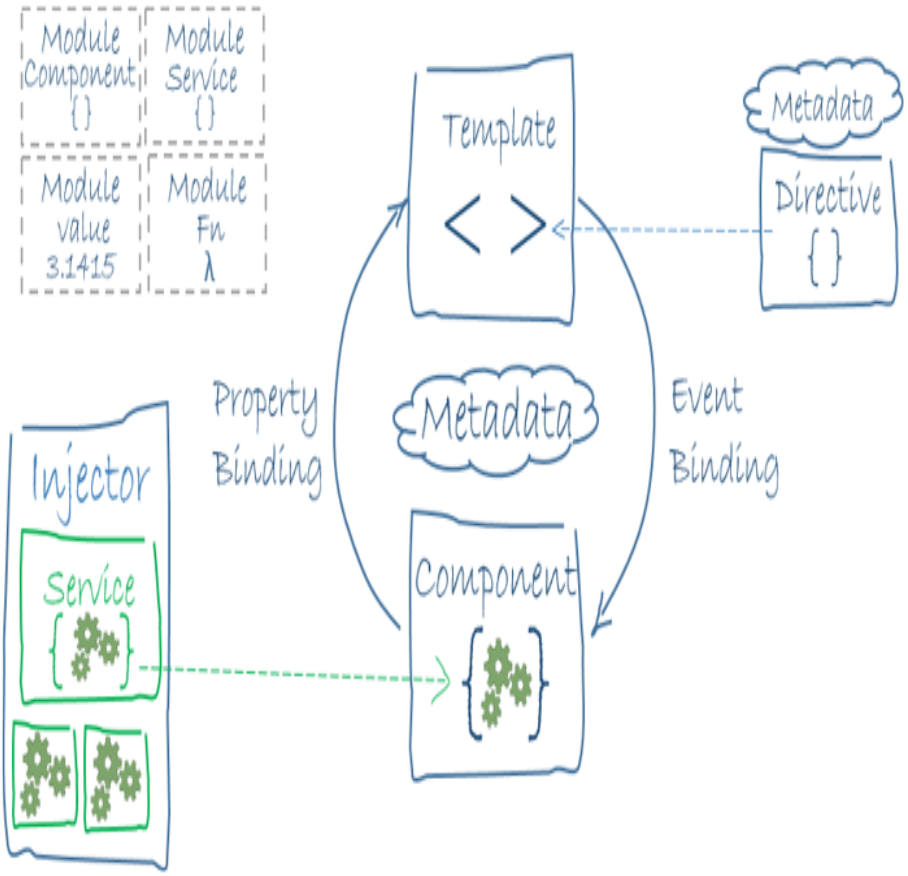

Figura 4.13: Diagrama de relaciones entre piezas básicas de Angular. Tomada de https://angular.io/guide/architecture.

La aplicación Angular desarrollada en este proyecto debe de adaptarse al formato *back end* basado en una arquitectura de microservicios. Para ello se estructurará el código de tal forma que cada uno o dos módulos representen a la parte *front end* de un microservicio.

En la Figura 4.14 podemos ver el diseño de la aplicación Angular.

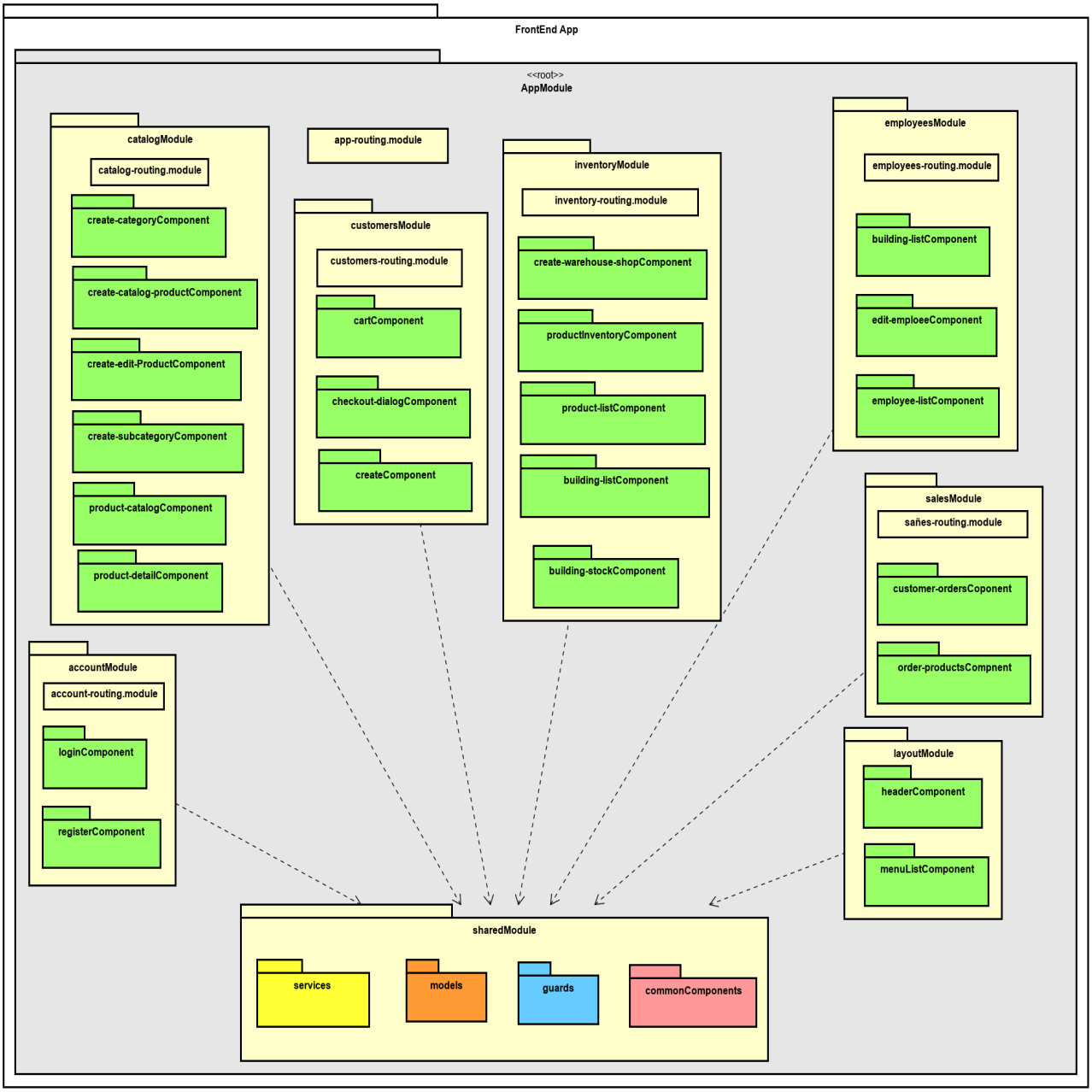

Figura 4.14: Diseño de la arquitectura de la aplicación angular.

CAPÍTULO 5

DISEÑO DE LOS MICROSERVICIOS

5.1. DISEÑO DE LIBRERÍAS COMUNES

En un sistema basado en una arquitectura de microservicios se recomienda que el desarrollo de cada microservicio este lo más aislado posible del desarrollo de otros microservicios relacionados, pero siempre hay excepciones si permiten ahorrarnos trabajo.

Con el objetivo de reutilizar código para evitar duplicados y agilizar el proceso de desarrollo, se han diseñado una librería por cada tipo de tecnología en la que se ha implementado *back end*. Cada una de estas librerías será utilizada por todos los microservicios implementados por su misma tecnología.

A continuación se muestran las librerías que se han diseñado.

5.1.1. Diseño de librería común para microservicios en Spring Boot

Esta librería que se muestra en la Figura 5.1 contendrá los siguientes paquetes:

- Paquete *constants*: contiene las clases enum, diccionarios y valores estáticos comunes en todos los microservicios en Spring Boot.
- Paquete *event*: contiene las clases comunes para la integración de eventos en los microservicios en Spring Boot.
- Paquete *type*: contiene las clases e interfaces comunes para entities y Data Transfer Objects (DTOs) en los microservicios Spring Boot.
- Paquete *exceptions*: contiene las excepciones comunes de los microservicios en Spring Boot.
- Paquete *extensible*: contiene las clases que extienden la funcionalidad de los DTOs en Spring Boot.
- Paquete *security*: contiene las clases comunes encargadas de la autenticación y autorización en los microservicios en Spring Boot.

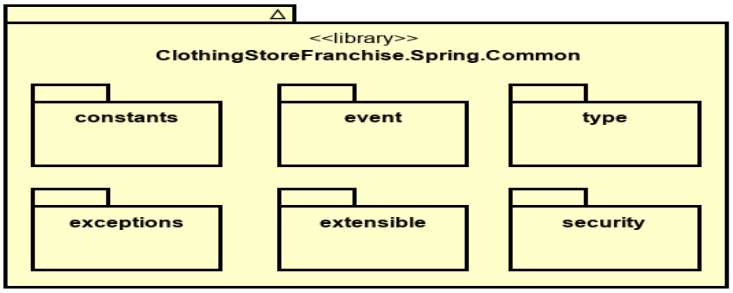

Figura 5.1: Diseño de un librería común para los microservicios en Spring Boot

5.1.2. Diseño de librería común para microservicios en .Net Core

Esta librería que se muestra en la Figura 5.1 contendrá los siguientes paquetes:

- Paquete *constants*: contiene las clases enum, diccionarios y valores estáticos comunes en todos los microservicios en ASP.Net Core.
- Paquete *event*: contiene las clases comunes para la integración de eventos en los microservicios en ASP.Net Core.
- Paquete *type*: contiene las clases e interfaces comunes para Data Access Objects (DAOs), entities y DTOs en los microservicios ASP.Net Core.
- Paquete *exceptions*: contiene las excepciones comunes de los microservicios en ASP.Net Core.
- Paquete *extensible*: contiene las clases que extienden la funcionalidad de los DTOs y entities en ASP.Net Core.
- Paquete *security*: contiene las clases comunes encargadas de la autenticación y autorización en los microservicios en ASP.Net Core.
- Paquete *mapper*: contiene las clases para configurar el *mapper* en los microservicios en ASP.Net Core.
- Paquete *rabbitmq*: contiene las clases para configurar el bus de eventos RabbitMQ en los microservicios en ASP.Net Core.
- Paquete *entityFramework*: contiene las clases base de los DAOs de los microservicios en ASP.Net Core.

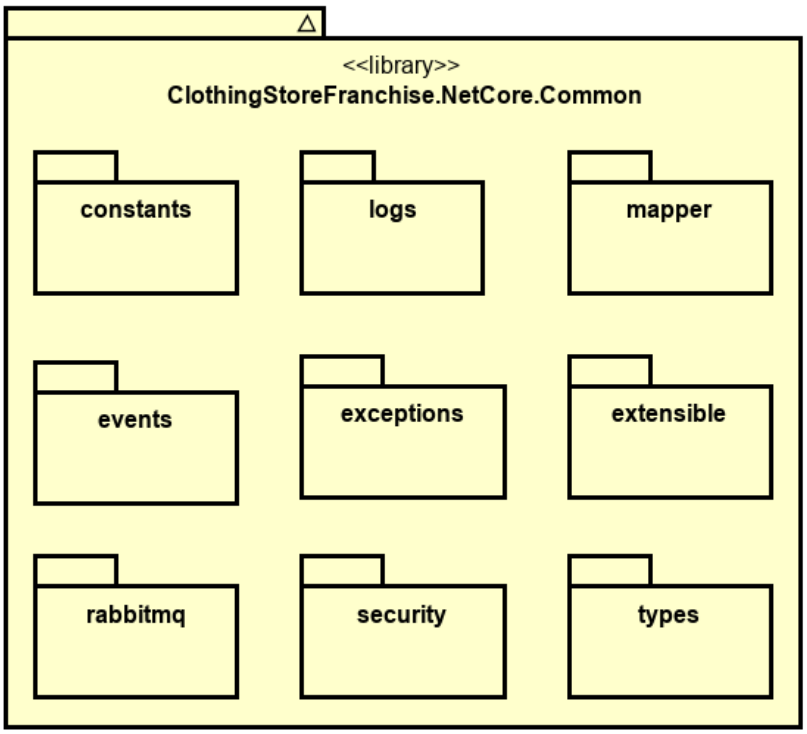

Figura 5.2: Diseño de librería común para los microservicios en ASP.Net Core

5.2. ARQUITECTURA GENERAL DE LOS MICROSERVICIOS

La arquitectura de todos los microservicios se ha diseñado siguiendo un patrón de capas. En este patrón de diseño el código se separa en capas, donde cada capa tiene una responsabilidad concreta y ofrece servicio a las capas superiores.

Siguiendo este patrón de capas se ha definido la estructura de los microservicios, de tal modo que

se ha dividido el código según su responsabilidad en los siguientes paquetes:

- Paquete *api*: contiene los controladores de la API REST del microservicio. A través de estos controladores la API Gateway se comunica con el microservicio y el cliente puede acceder a los servicios que este ofrece.
- Paquete *serviceFacade*: contiene el código que se encarga de gestionar la lógica de negocio. Este paquete se encarga de abstraer la lógica de negocio y el procesamiento de datos a los controladores de la API. Procesa y combina la información obtenida de las múltiples tablas de la base de datos del microservicio. Además, es el responsable del *mapping* entre los DTOs y el modelo. También desde este paquete se realizan la publicación de mensajes a través del bus de mensajería.
- Paquete *eventHandler*: contiene el código que gestiona las acciones que se deben producir cuando el microservicio recibe un evento a través del bus de eventos. Cada *handler* gestiona un único evento y son el punto de entrada de eventos y mensajes al microservicio.
- Paquete *dto*: contiene las clases envoltorio cuya finalidad es crear un objeto plano (POJO) que será enviado a través de la API REST o el bus de eventos. Estos objetos que están compuestos por una serie de atributos que serán obtenidos de una o varias tablas, envuelven los datos solicitados por el cliente.
- Paquete *model*: contiene un conjunto de *entities* que representan el modelo del microservicio.
- Paquete *repository*: implementado en los microservicios en Spring Boot, contiene los Repositorios de cada tabla de la base de datos.
- Paquete *dao*: implementado en los microservicios en .NetCore, contiene los DAOs de cada tabla de la base de datos.
- Paquete *security*: contiene el código encargado del procesamiento de JWT (JSON Web Token) con el que se autentificará y autorizará las solicitudes del usuario al microservicio.

Los microservicios de la aplicación desarrollada en este trabajo se han implementado usando dos tecnologías distintas, por lo que el diseño de la arquitectura presenta ligeras diferencias. En las figuras que se muestra a continuación se muestra el esquema de la arquitectura general para los microservicios implementados en Spring Boot (Figura 5.3) y para los microservicios implementados en ASP.NetCore (Figura 5.4).

5.2.1. Spring Boot Microservicio

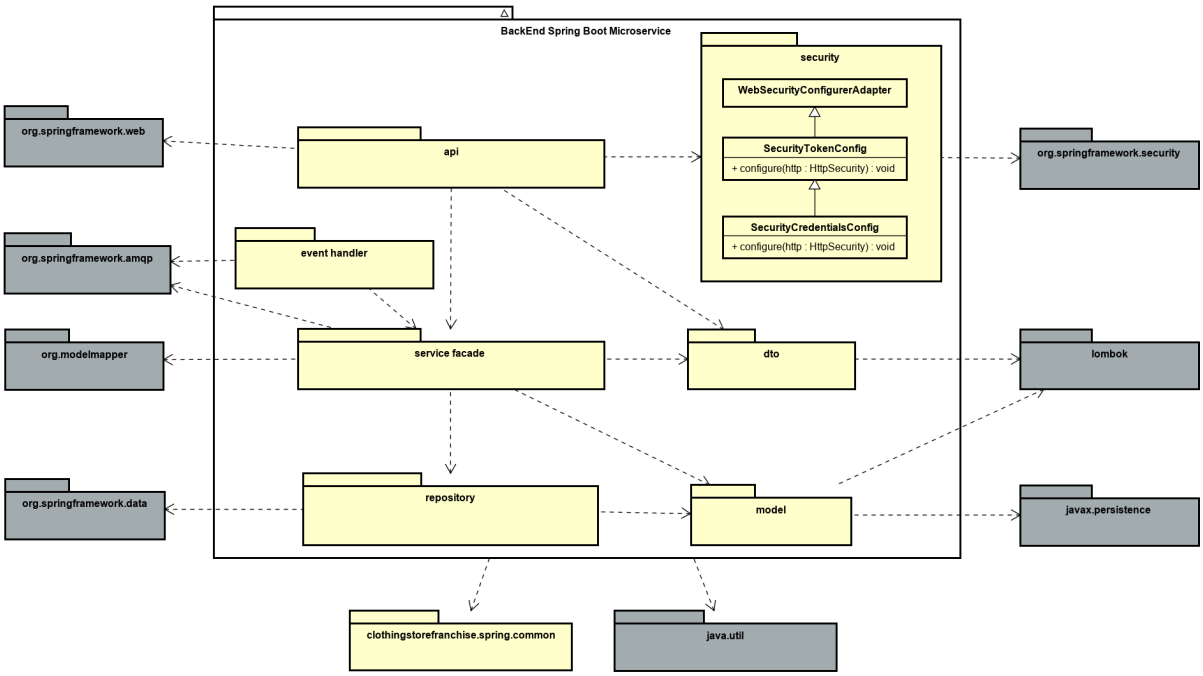

Figura 5.3: Arquitectura general de un microservicio en Spring Boot.

5.2.2. ASP.NetCore Microservicio

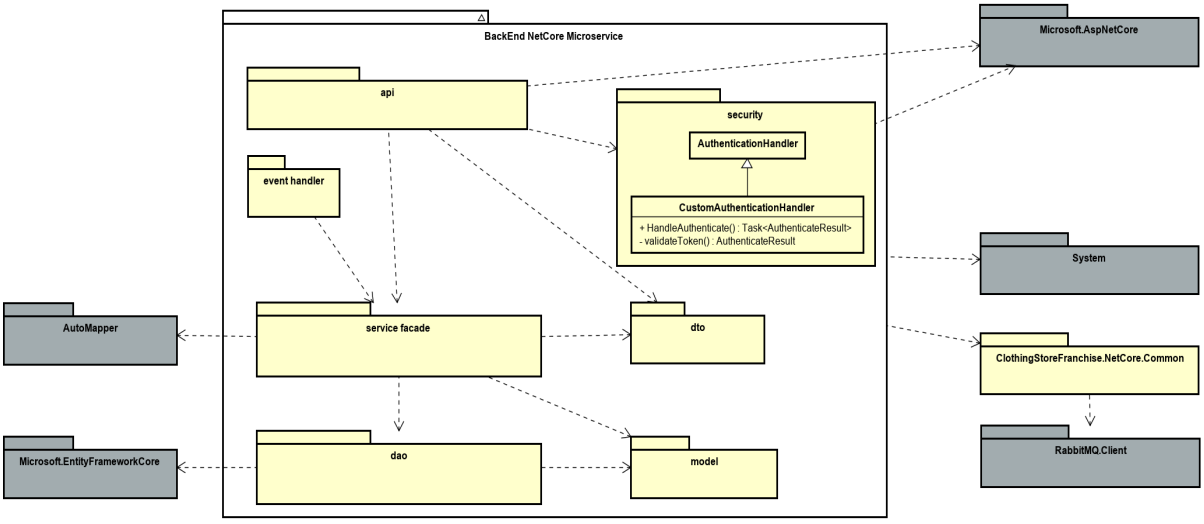

Figura 5.4: Arquitectura general de un microservicio en ASP.NetCore.

5.3. DISEÑO DEL MICROSERVICIO AUTHENTICATION

5.3.1. Diseño de la API REST

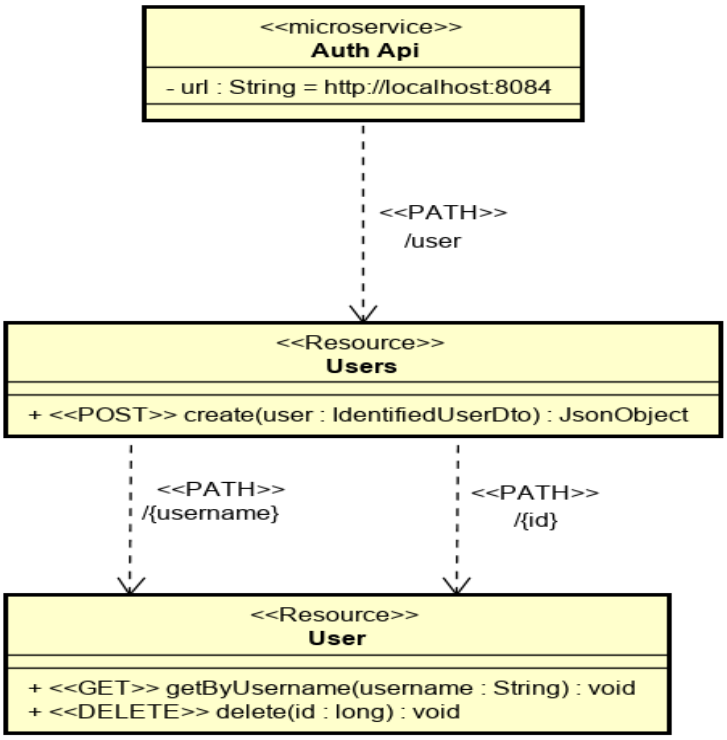

Figura 5.5: Diseño de la API REST del Microservicio Authentication.

5.3.2. Diseño de la base de datos

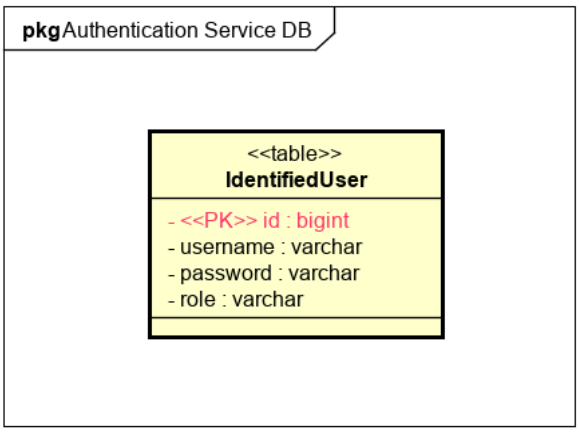

Figura 5.6: Diseño de la base de datos del Microservicio Authentication.

5.3.3. Arquitectura del Microservicio

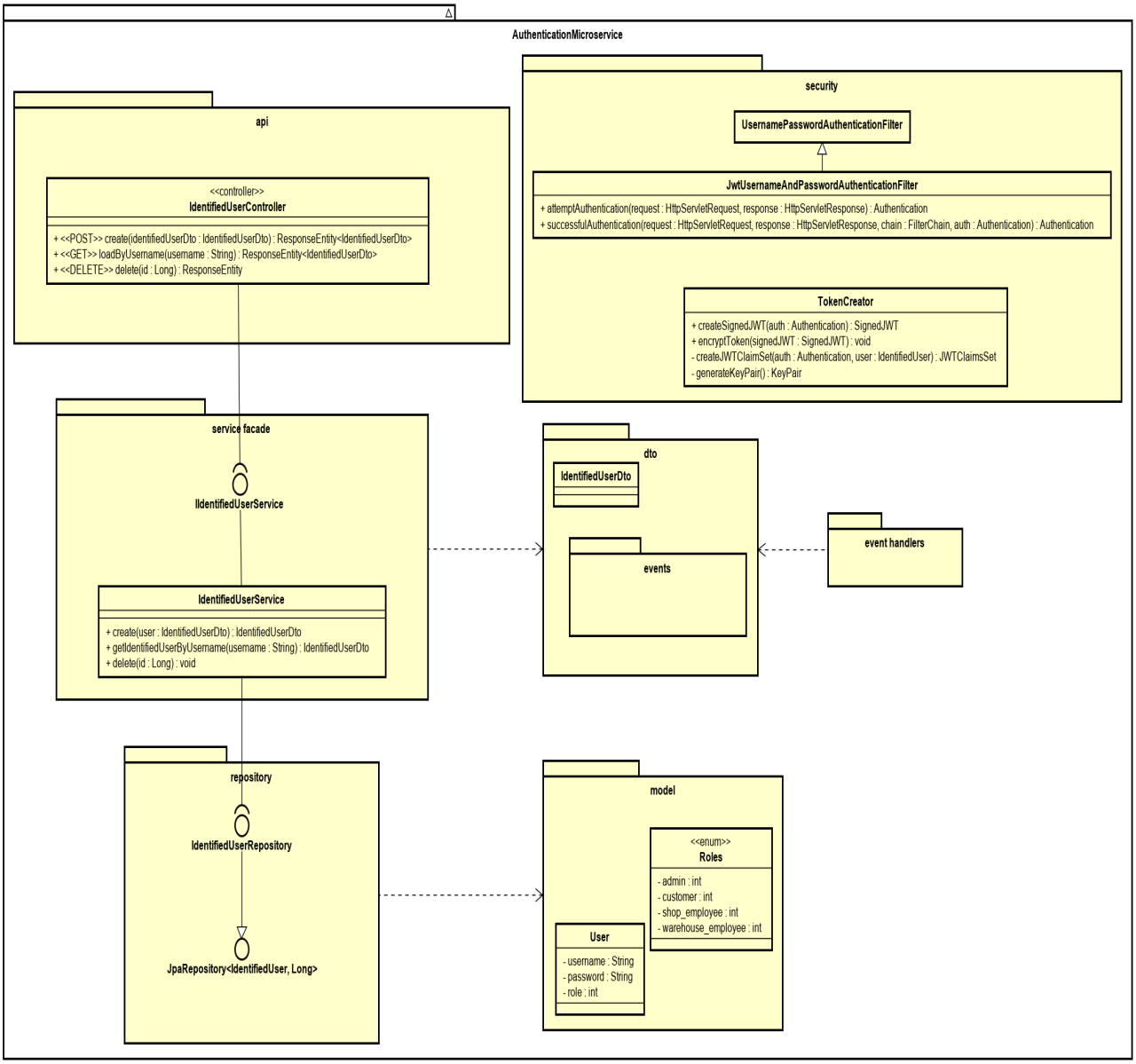

Figura 5.7: Arquitectura del Microservicio Authentication.

5.4. MICROSERVICIO CATALOG

5.4.1. Diseño de la API REST

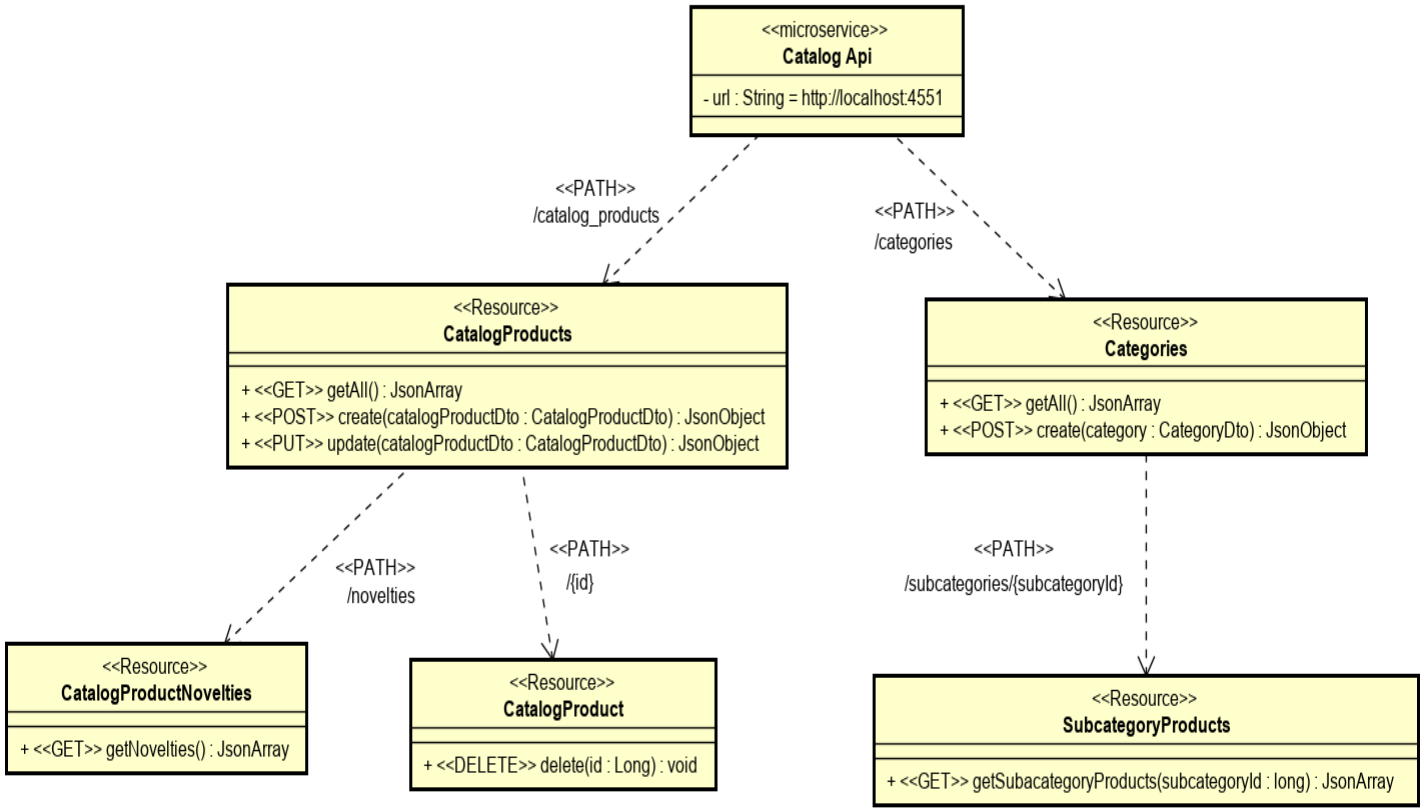

Figura 5.8: Diseño de la API REST del Microservicio Catalog.

5.4.2. Diseño de la base de datos

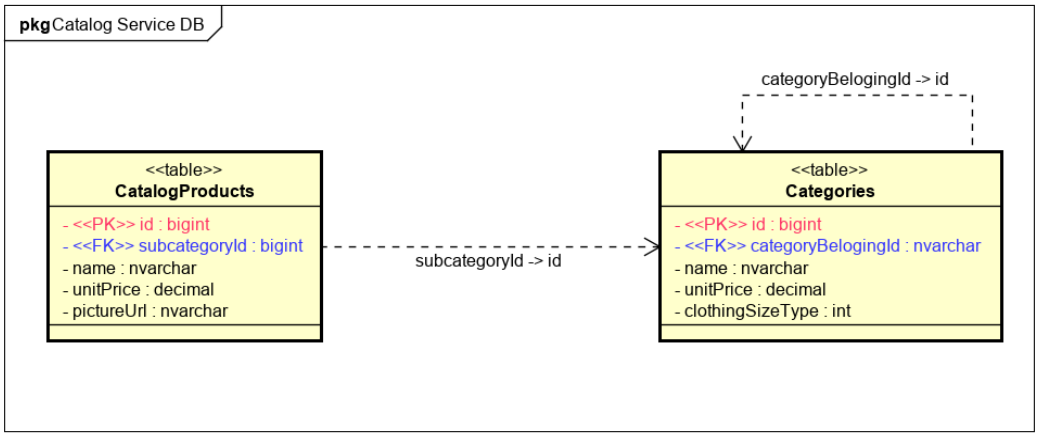

Figura 5.9: Diseño de la base de datos del Microservicio Catalog.

5.4.3. Arquitectura del Microservicio

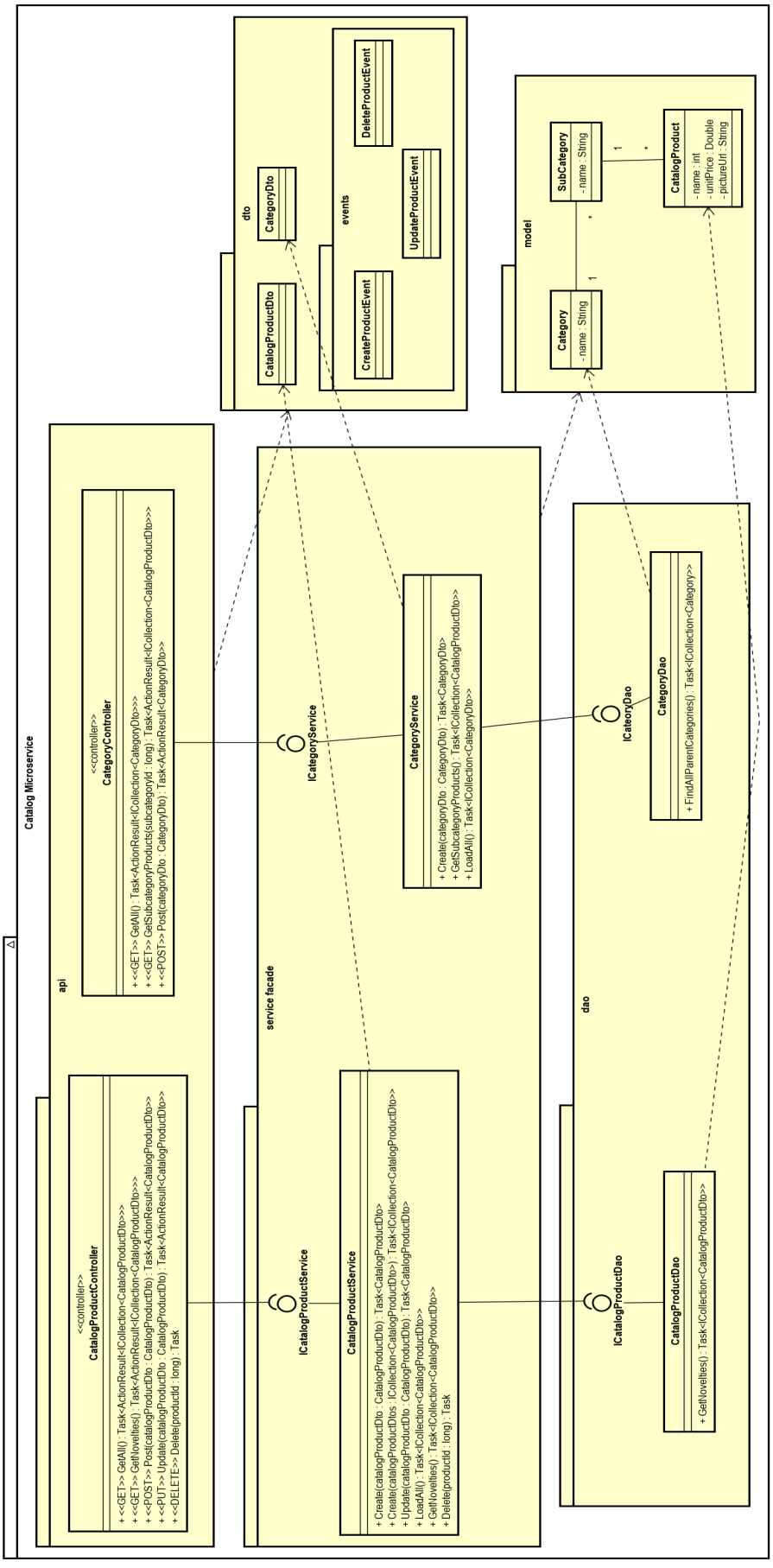

Figura 5.10: Arquitectura del Microservicio Catalog.

5.5. MICROSERVICIO CUSTOMERS

5.5.1. Diseño de la API REST

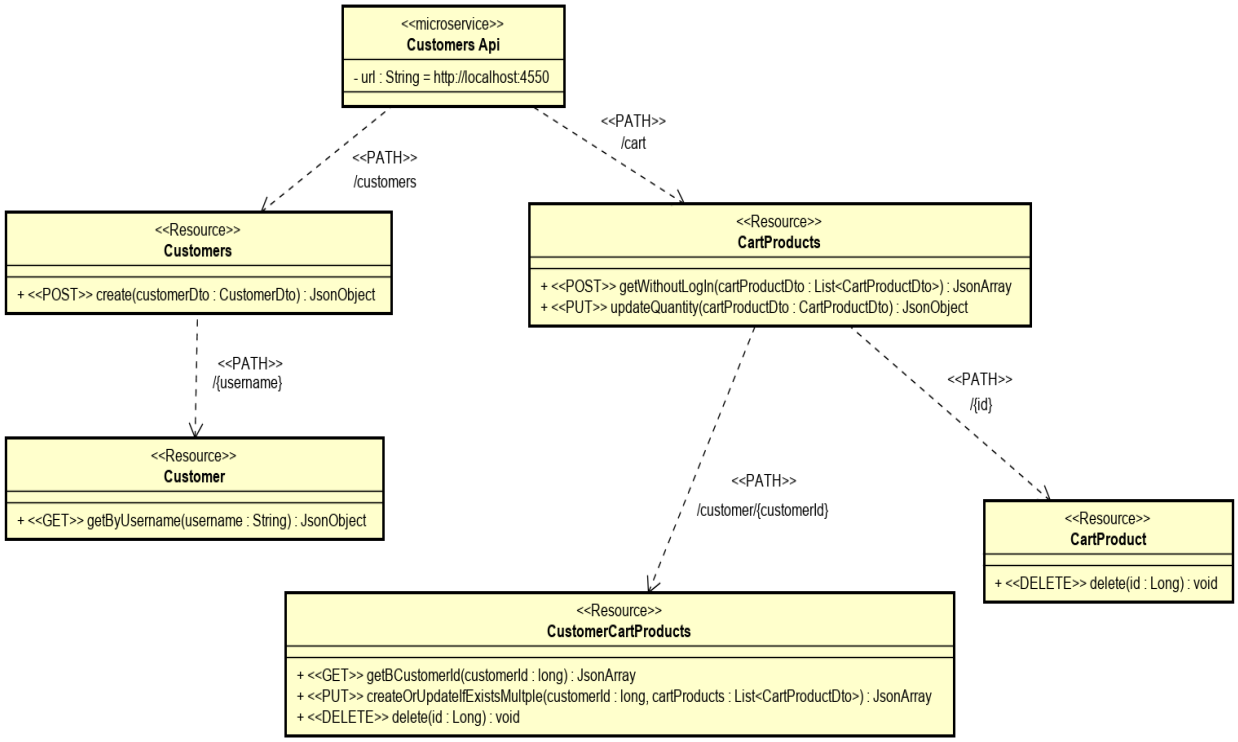

Figura 5.11: Diseño de la API REST del Microservicio Custmoers.

5.5.2. Diseño de la base de datos

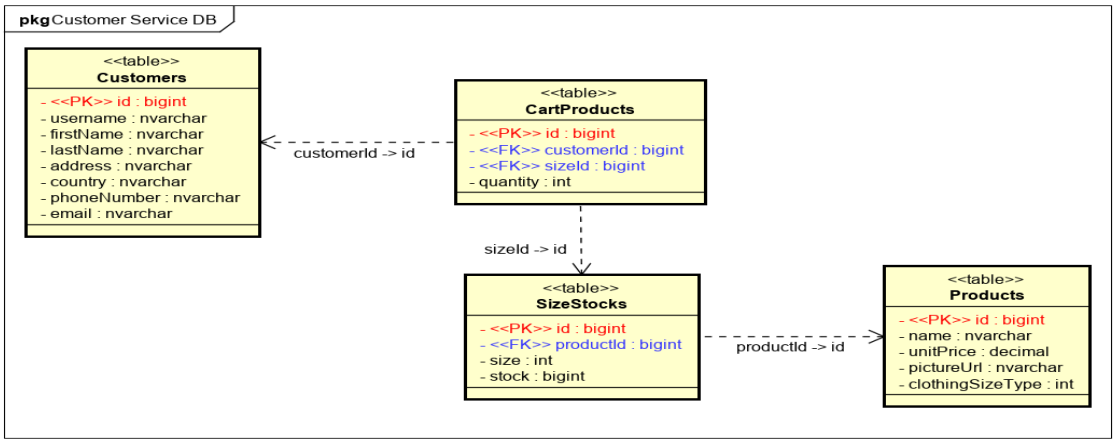

Figura 5.12: Diseño de la base de datos del Microservicio Customers.

5.5.3. Arquitectura del Microservicio

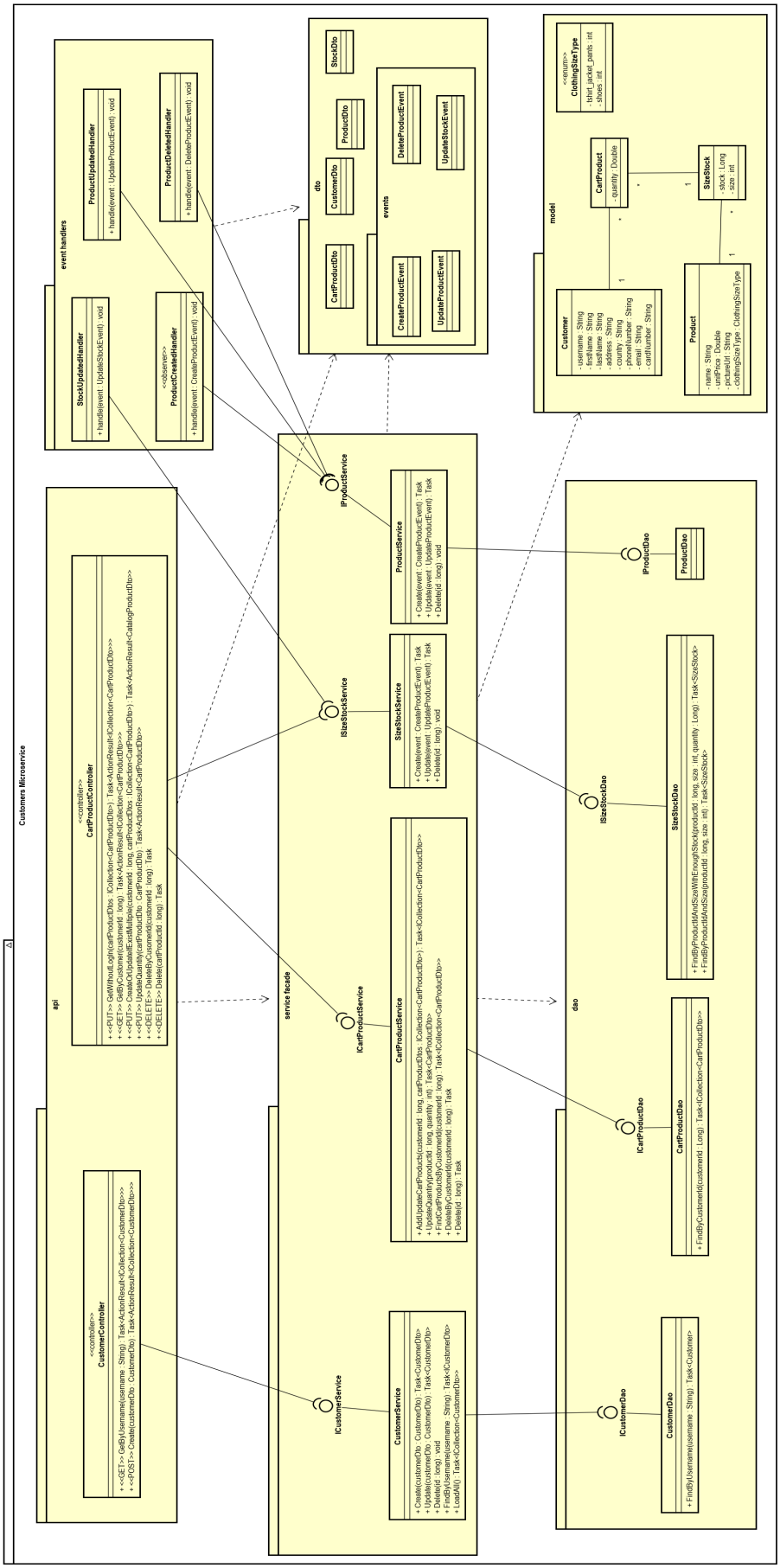

Figura 5.13: Arquitectura del Microservicio Customers.

5.6. MICROSERVICIO EMPLOYEES

5.6.1. Diseño de la API REST

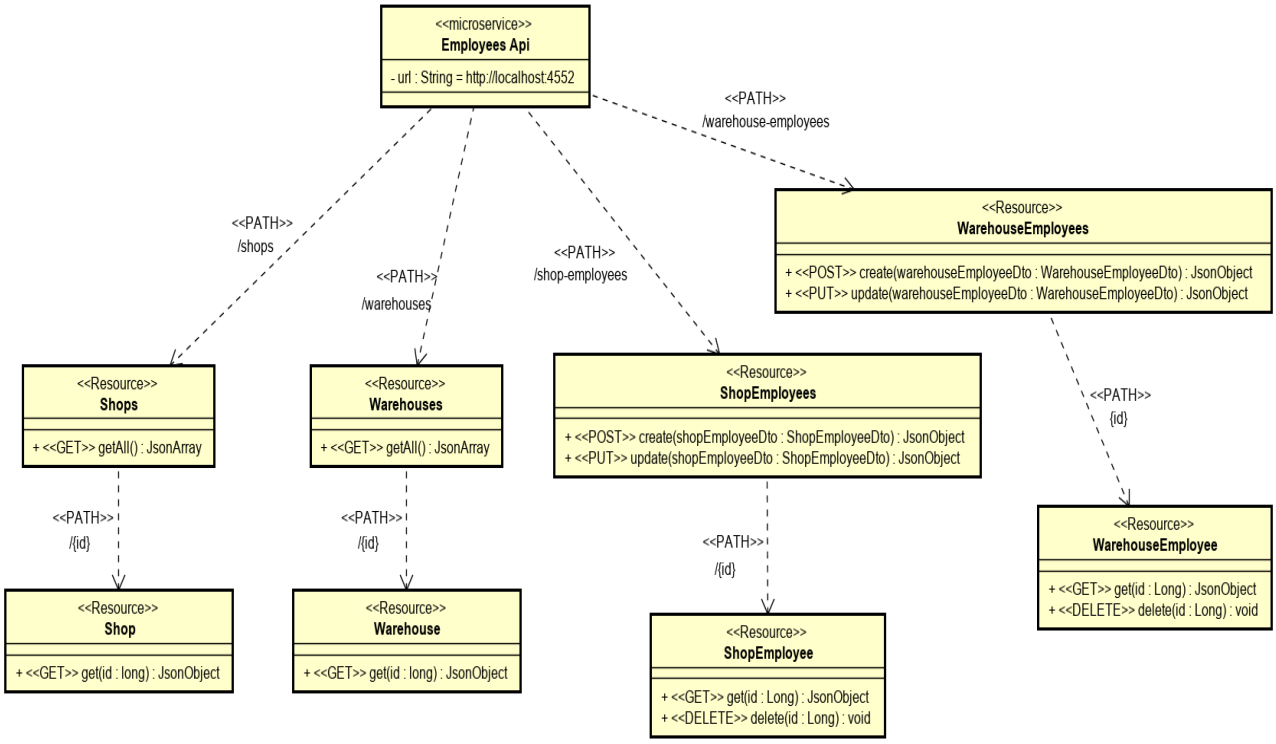

Figura 5.14: Diseño de la API REST del Microservicio Employees.

5.6.2. Diseño de la base de datos

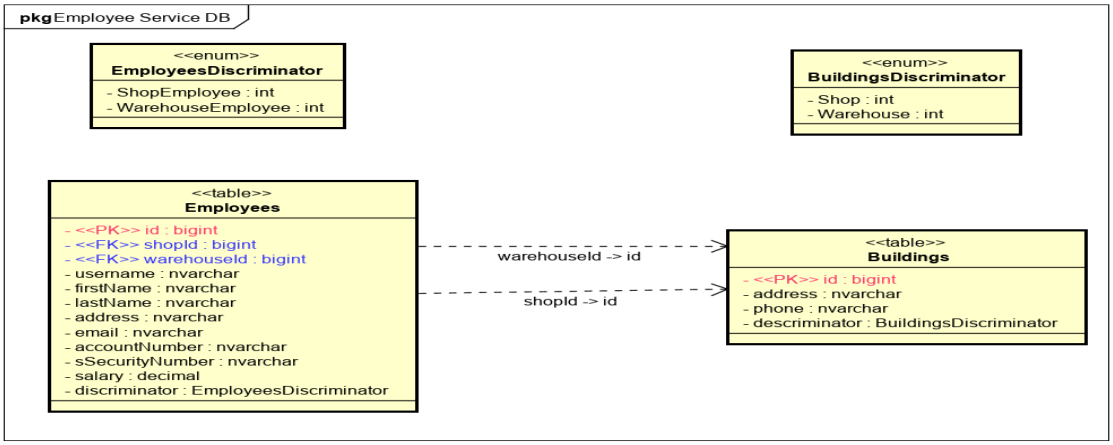

Figura 5.15: Diseño de la base de datos del Microservicio Employees.

5.6.3. Arquitectura del Microservicio

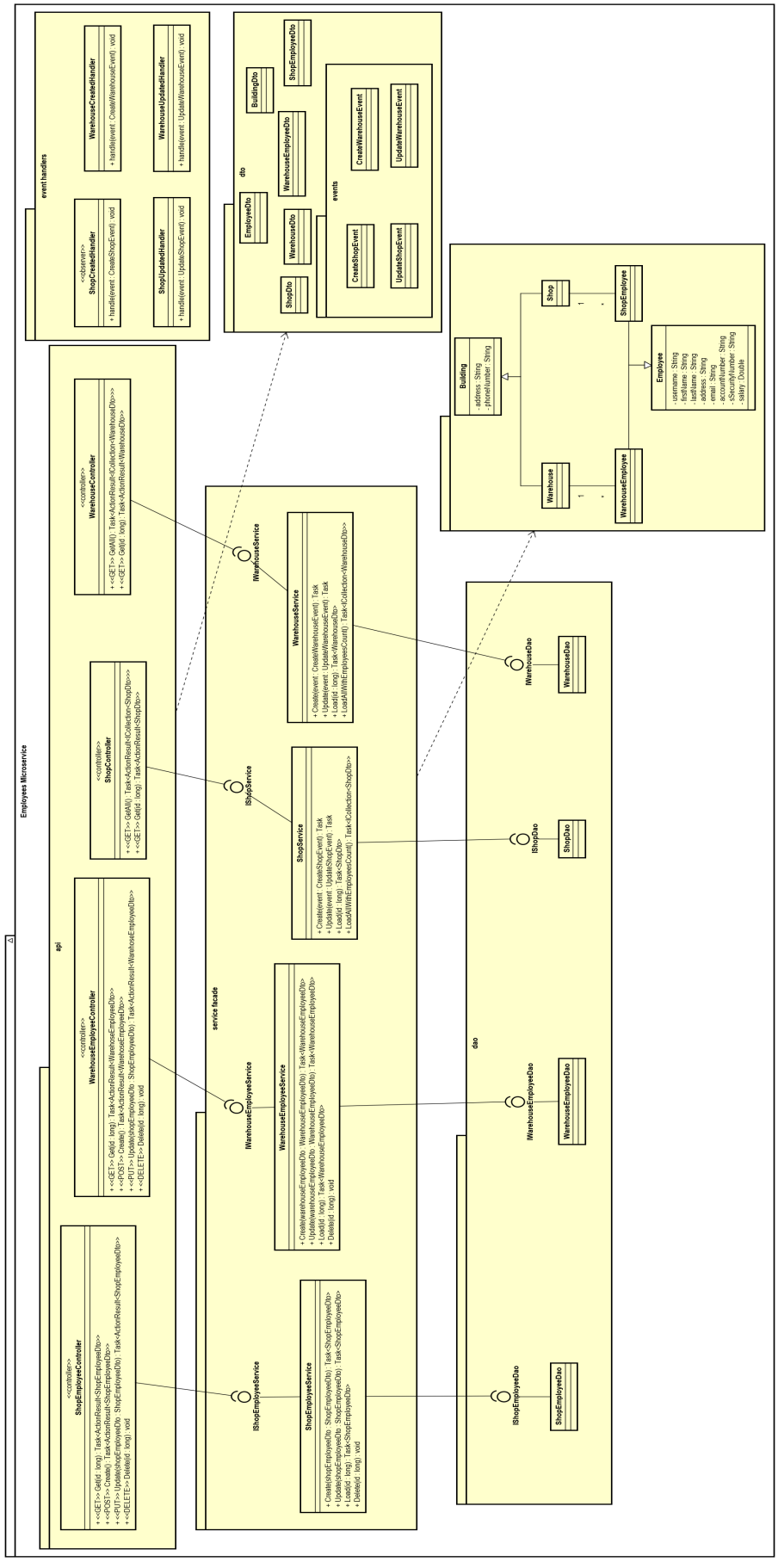

Figura 5.16: Arquitectura del Microservicio Employees.

5.7. MICROSERVICIO INVENTORY

5.7.1. Diseño de la API REST

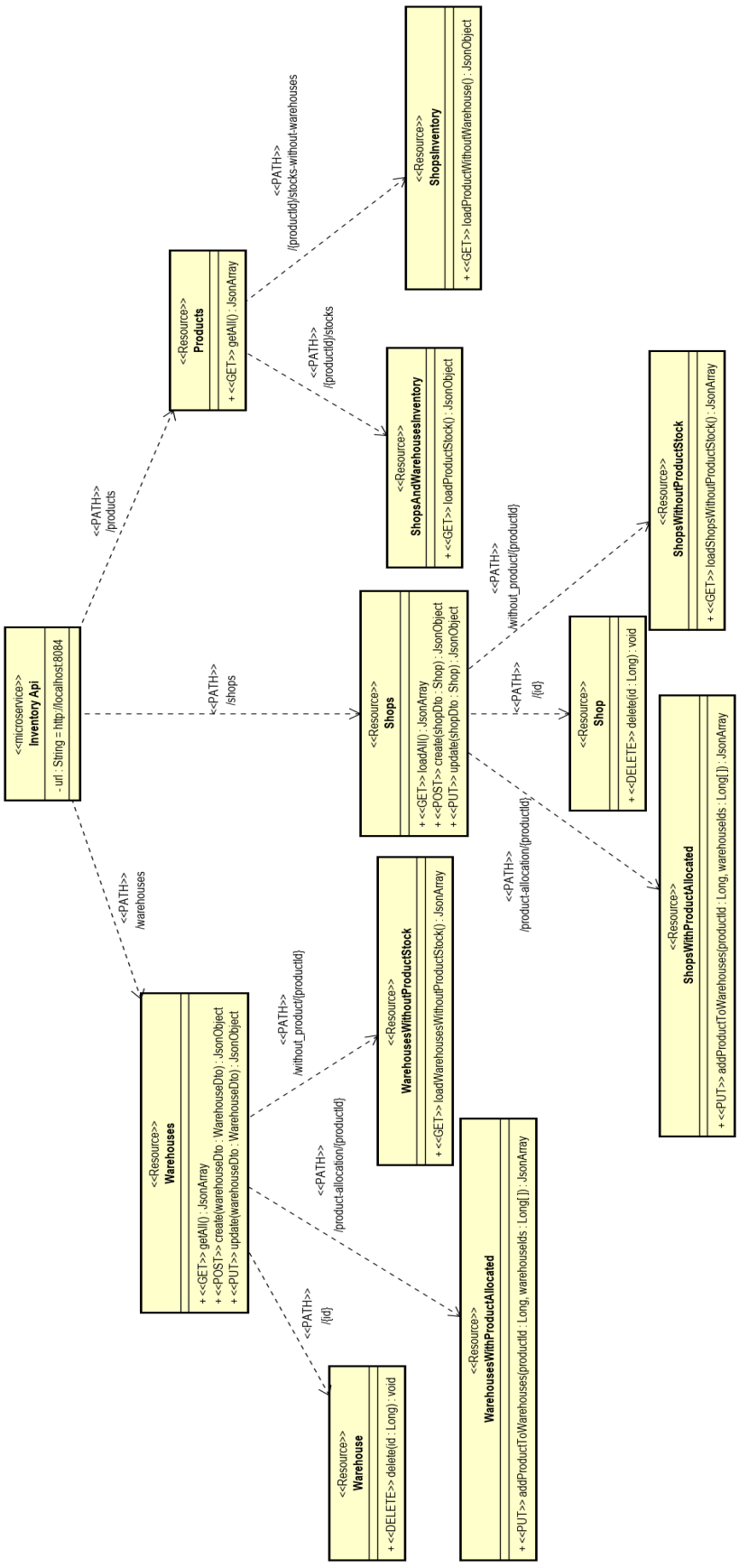

Figura 5.17: Diseño de la API REST del Microservicio Inventory.

5.7.2. Diseño de la base de datos

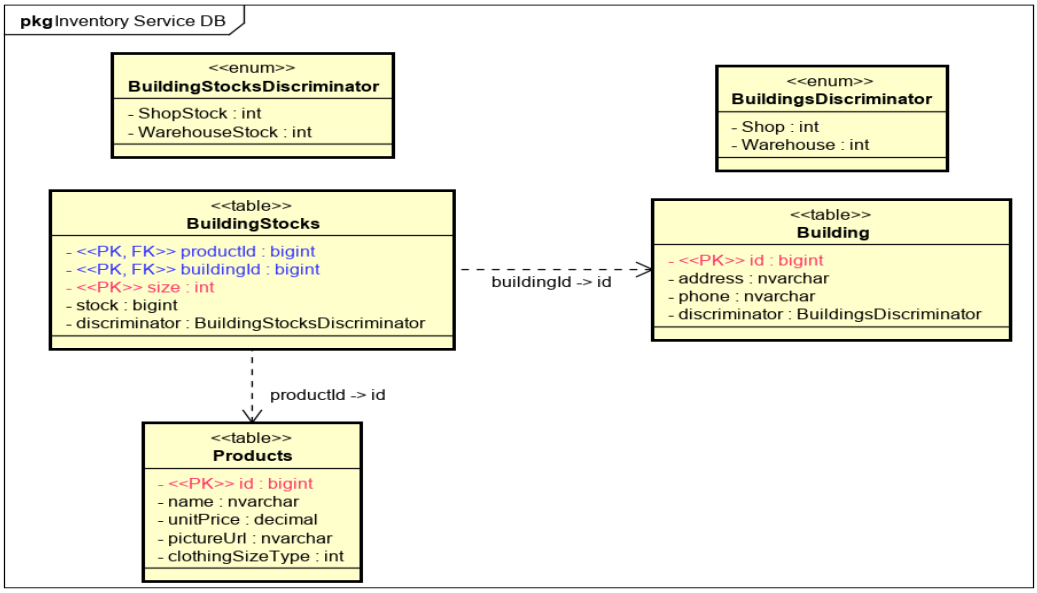

Figura 5.18: Diseño de la base de datos del Microservicio Inventory.

5.7.3. Arquitectura del Microservicio

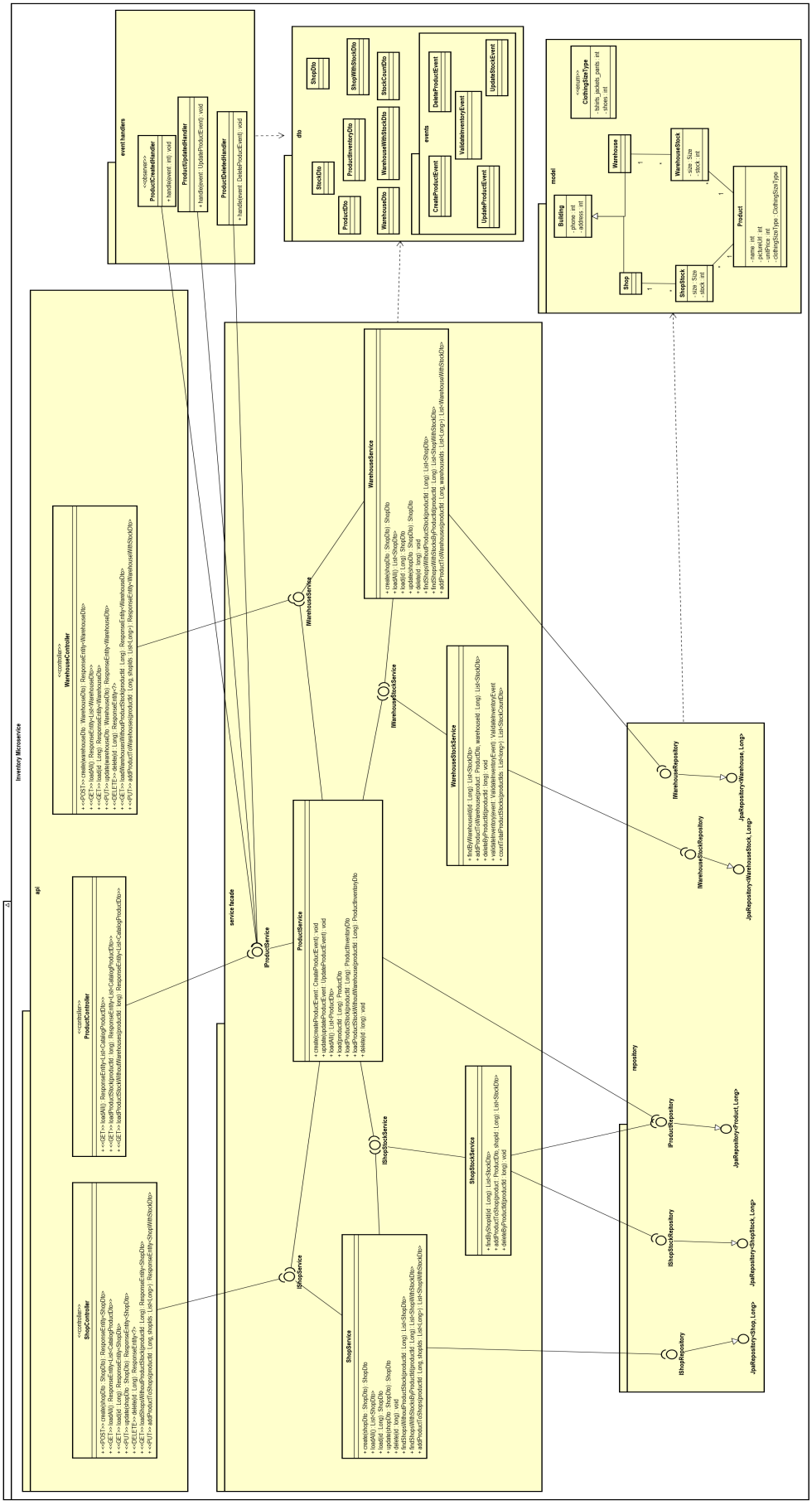

Figura 5.19: Arquitectura del Microservicio Inventory.

5.8. MICROSERVICIO SALES

5.8.1. Diseño de la API REST

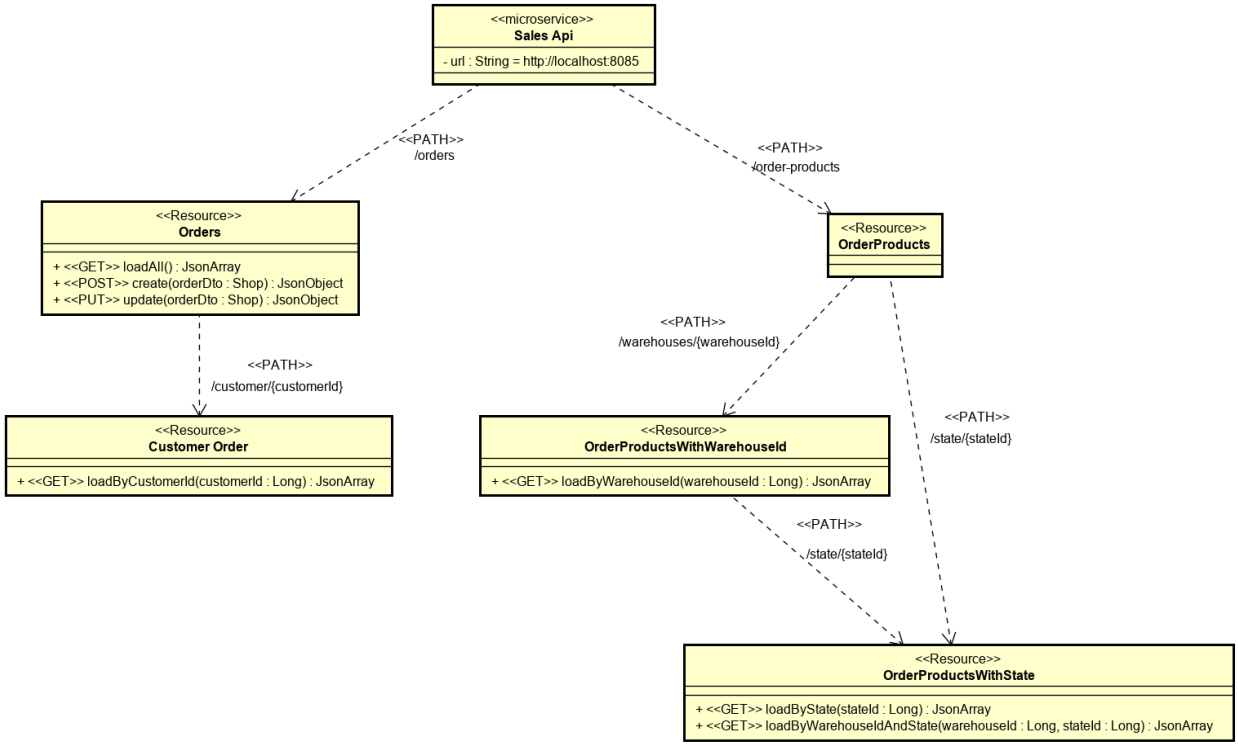

Figura 5.20: Diseño de la API REST del Microservicio Sales.

5.8.2. Diseño de la base de datos

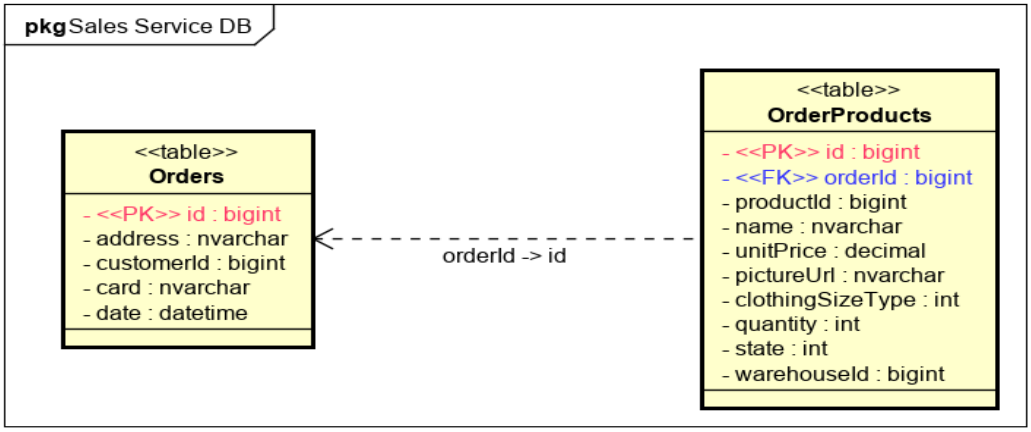

Figura 5.21: Diseño de la base de datos del Microservicio Sales.

5.8.3. Arquitectura del Microservicio

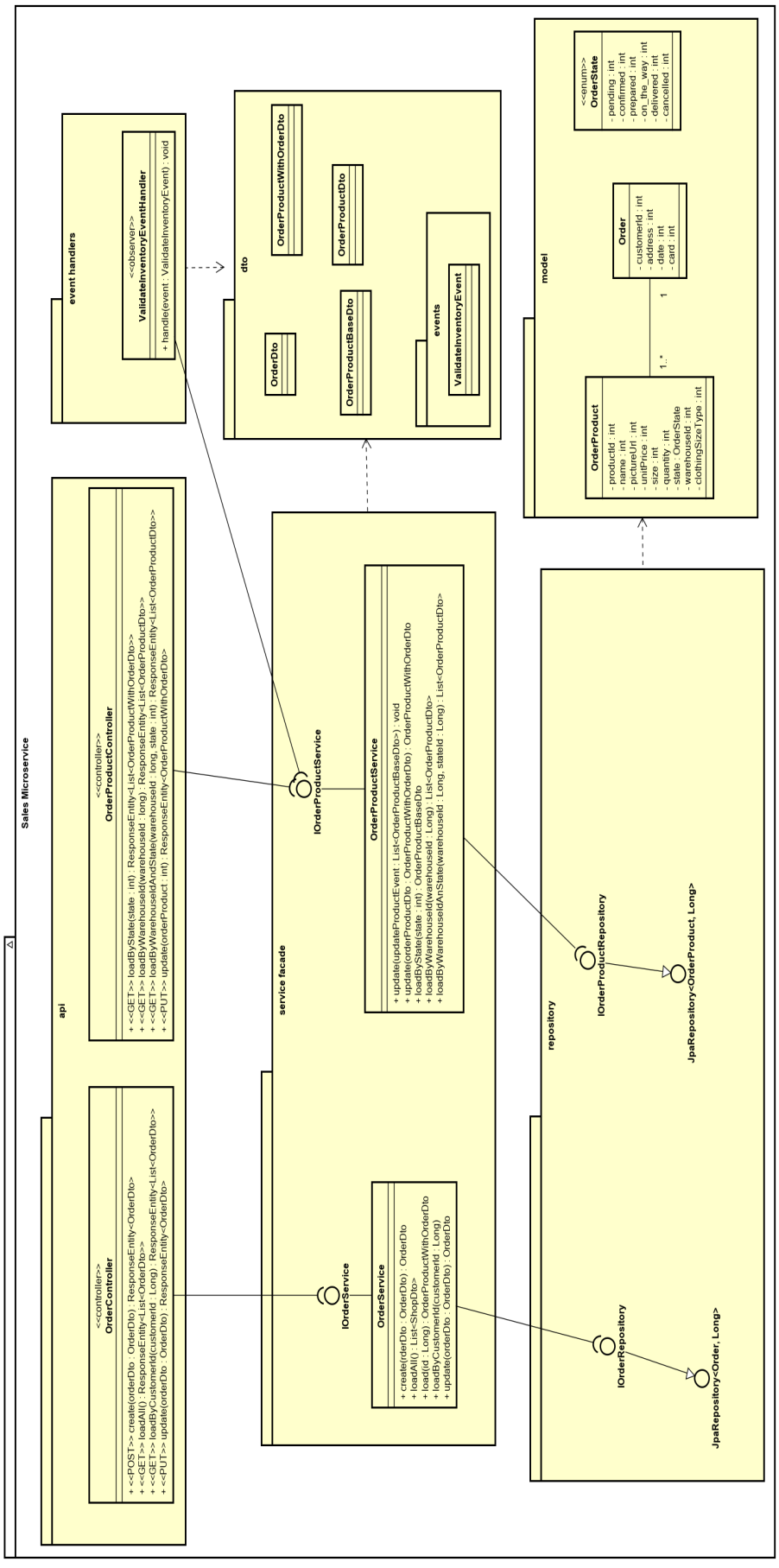

Figura 5.22: Arquitectura del Microservicio Sales.

CAPÍTULO 6

IMPLEMENTACIÓN

En este capítulo se describe más detalladamente las tecnologías implicadas en la implementación del sistema y se muestran ejemplos que se han considerado interesantes sobre la implementación.

6.1. CONTROL DE VERSIONES

El control de versiones del proyecto ha sido llevado a cabo con el software de control de versiones Git y la plataforma de desarrollo colaborativo GitHub. Se ha elegido GitHub por ofrecer una mayor compatibilidad con otras tecnologías, además de otras muchas ventajas.

Siguiendo las buenas prácticas recomendadas, se ha creado un repositorio por cada microservicio que forma el sistema con el objetivo de aislar su desarrollo. Estos repositorios han sido agrupados en una organización, en la que se puede ver toda la implementación del sistema. En el siguiente enlace se accede a dicha organización:

https://github.com/ClothingStoreFranchise

6.2. BACK-END

Con el fin de poner en práctica las posibilidades que ofrecen una arquitectura de microservicios, se ha implementado la parte back-end de forma políglota, utilizando dos frameworks diferentes: *ASP.Net Core* y *Spring Boot*.

6.2.1. Implementación del Service Discovery y de la API Gateway

Para implementar toda la parte que forma la puerta de entrada a back-end, es decir, la API Gateway, el service discovery, el circuit breaker y el load balancer se ha utilizado Netflix OSS.

Netflix OSS son un conjunto de herramientas y componentes de software que permiten desarrollas de manera fácil y rápida aplicaciones y servicios, que implementan algunos de los patrones más comunes que se usan en sistemas distribuidos. Gracias a Spring Cloud Netflix estas herramientas se pueden implementar fácilmente con Spring Boot.

El service discovery se ha implementado como un microservicio Spring Boot con Eureka Server.

Listado 6.1: Implementación del service disconvery con Eureka Server

```
1 @SpringBootApplication
2 @EnableEurekaServer
3 public class DiscoveryApplication
```

La Api Gateway se ha implementado en otro microservicio mediante el ”edge service” Zuul que incorpora el load balancer Ribbon y el circuit breaker Hystrix. Zuul es un servidor compuesto por filtros que permite enrutado dinámico, balanceo de carga, monitorización y controlar la seguridad de peticiones. A continuación se muestra como se ha implementado Zuul.

Listado 6.2: Implementación de Zuul con Spring Cloud Netflix

```
@SpringBootApplication
@EnableZuulProxy
@EnableEurekaClient
@EnableConfigurationProperties(value = JwtConfiguration.class)
@ComponentScan("clothingstorefranchise")
public class GatewayApplication
```

En cuanto al registro de los microservicios en el service registry del service discovery, mediante *@EnableEurekaClient* y la configuración siguiente se registra un micorservicio Spring Boot en el server registry de Eureka Server en el momento que este microservicio es desplegado.

Listado 6.3: Service registry con eureka, microservicio en Spring Boot

```
eureka:
  instance:
    prefer-ip-address: true
  client:
    service-url:
      defaultZone: "http://eureka:8081/eureka/"
    register-with-eureka: true
```

Para registrar los microservicios en .Net con Eureka Server, ha sido necesaria la librería *Steeltoe.Discovery.ClientCore* y añadir la configuración siguiente al fichero *appsettings.json*.

Listado 6.4: Service registry con eureka, microservicio en .Net

```
"eureka": {
  "instance": {
    "PreferIpAddress": true
  },"client": {
    "serviceUrl": "http://eureka:8081/eureka/"
    "shouldRegisterWithEureka": true
  }
}
```

6.2.2. Implementación del message broker

En esta sección se expone de forma simplificada la implementación del message broker RabbitMQ en .Net Core y en Spring Boot.

RabbitMQ es uno de los message brokers open source más populares. Es ligero y se puede desplegar fácilmente en la nube. Esta diseñado para ser desplegado en sistemas distribuidos y ser configurado para satisfacer los requisitos de una alta disponibilidad y una alta escalabilidad.

- **Implementación en Spring Boot**

En Spring Boot la implementación de RabbitMQ ha sido con el framework AMQP de Spring mediante la librería *org.springframework.amqp.core*.

Listado 6.5: Implementación de la API de mensajería

```
1  //Implementa la API de mensajería
2  @Service
3  public class InventoryListener implements MessageListener {
4
5    @Autowired
6    private ApplicationEventPublisher eventPublisher;
7
8    //relaciona los event handler con el típo de evento
9    @Autowired
10   private EventBusSubscriptions subscriptions;
11
12   //punto de entrada de los mensajes
13   @RabbitListener(queues = RabbitMqConfig.QUEUE)
14   public void onMessage(Message message) {
15
16   //según el tipo de evento los mensajes son distribuidos por los event handlers
17     String eventName =
18     message.getMessageProperties().getReceivedRoutingKey();
19     Class<?> eventType = subscriptions.getEventType(eventName);
20     String event = new String(message.getBody(), ↩
           ↪ StandardCharsets.UTF_8);
21     try {
22       eventPublisher.publishEvent(new ObjectMapper()
23         .readValue(event, eventType));
24     } catch (JsonProcessingException e) {
25       e.printStackTrace();
26     }
27   }
28 }
```

Listado 6.6: Implementación y configuración de RabbitMQ en Spring Boot

```
1
2  @Configuration
3  @ComponentScan(value = ↩
       ↪ "clothingstorefranchise.spring.common.event")
4  public class RabbitMqConfig {
5    //otro código...
6
7    @Autowired
8    private EventBusSubscriptions subscriptions;
9
10   //se crea si no existe el exchange que direcciona los mensajes
11   @Bean  //devuelve un objeto que debe ser registrado como un bean
12   DirectExchange exchange() {
13     return new DirectExchange(EXCHANGE_NAME);
14   }
15
16   //se crea una cola para los métodos publish/subscribe
17   @Bean
18   Queue primaryQueue() {
19     return QueueBuilder.durable(QUEUE)
20       .deadLetterExchange(DEAD_LETTER_EXCHANGE)
21       .deadLetterRoutingKey(PARKINGLOT_QUEUE)
22       .build();
23   }
24
25   //se crea una cola para los métodos request/response asíncronos
26   @Bean
27   public Queue queueRequest() {
28     return new Queue(QUEUE_REQUEST);
29   }
30
31   //Subscripción al evento CreateProductEvent
32   @Bean
33   Binding createProductEventBinding() {
34     subscriptions.addSubscription(CreateProductEvent.class);
35     return BindingBuilder.bind(primaryQueue())
36       .to(exchange())
37       .with(CreateProductEvent.class.getSimpleName());
38   }
39
40   //Subscripción al evento DeleteProductEvent
41   @Bean
42   Binding deleteProductEventBinding() {
43     subscriptions.addSubscription(DeleteProductEvent.class);
44     return BindingBuilder.bind(primaryQueue()).to(exchange())
45       .with(DeleteProductEvent.class.getSimpleName());
46   }
47
48   //Subscripción al evento UpdateProductEvent
49   @Bean
50   Binding updateProductEventBinding() {
51     subscriptions.addSubscription(UpdateProductEvent.class);
52     return BindingBuilder.bind(primaryQueue()).to(exchange())
53       .with(UpdateProductEvent.class.getSimpleName());
54   }
55
56   //Subscripción al evento ValidateInventoryEvent
57   @Bean
58   public Binding bindingRequest() {
59     return BindingBuilder.bind(queueRequest())
60     .to(exchange()).with(ValidateInventoryEvent.class.getSimpleName());
61   }
62
63   //otro código...
64 }
```

- **Implementación en .Net Core**

En .Net Core la implementación de RabbitMQ ha sido con la librería *RabbitMQ.Client*

En el siguiente método se realizan las subscripciones a los eventos.

Listado 6.7: Implementación y configuración de RabbitMQ en .NetCore

```
private void ConfigureEventBus(IApplicationBuilder app)
{
  var eventBus = ↩
      ↪ app.ApplicationServices.GetRequiredService<IEventBus>();

  //catalog microservice events
  eventBus.Subscribe<CreateProductEvent, ProductCreatedHandler>();
  eventBus.Subscribe<UpdateProductEvent, ProductUpdatedHandler>();
  eventBus.Subscribe<DeleteProductEvent, ProductDeletedHandler>();
  //inventory microservice events
  eventBus.Subscribe<UpdateStockEvent, StockUpdatedHandler>();
}
```

6.2.3. Implementación del paquete api

A continuación se muestra un ejemplo de cómo se ha implementado un controlador de API en Spring Boot y en .Net Core.

- **Implementación en Spring Boot**

Listado 6.8: Implementación de un controlador de API REST con Spring Boot

```
@RestController
@RequestMapping("/warehouses") //Ruta raíz
public class WarehouseController {

  @Autowired  //Inyección de dependencias
  private IWarehouseService warehouseService;

  @PostMapping()
  public ResponseEntity<WarehouseDto> create(@Valid ↩
      ↪ @RequestBody WarehouseDto warehouseDto)
  //otro código...

  @GetMapping("/{id}")
  public ResponseEntity<WarehouseWithStockDto> ↩
      ↪ load(@PathVariable Long id)
  //otro código...

  @PutMapping
  public ResponseEntity<WarehouseDto> update(@Valid ↩
      ↪ @RequestBody WarehouseDto warehouseDto)
  //otro código...

  @DeleteMapping("/{id}")
  public ResponseEntity<?> delete(@PathVariable Long id)
  //otro código...
}
```

- **Implementación en ASP.Net Core**

Listado 6.9: Implementación de un controlador de API REST con ASP.Net Core

```
1  [Route("catalog_products")] //Ruta raíz
2  public class CatalogProductController : ControllerBase
3  {
4      private readonly ICatalogProductService _catalogProductService;
5      //Inyección de dependencias
6      public CatalogProductController(ICatalogProductService ↩
           ↪ catalogProductService)
7      {
8          _catalogProductService = catalogProductService;
9      }
10
11     [HttpGet("novelties")]
12     [AllowAnonymous] //Control de autorización. Permite acceso anónimo.
13     public async ↩
           ↪ Task<ActionResult<ICollection<CatalogProductDto>>> ↩
           ↪ GetNovelties()
14     //otro código...
15
16     [HttpPost]
17     [Authorize(Roles = Role.Admin)] //Control de autorización, sólo usuarios
           con el rol Admin.
18     public async Task<ActionResult<CatalogProductDto>> ↩
           ↪ Post([FromBody] CatalogProductDto catalogProductDto)
19     //otro código...
20
21     [HttpDelete("{productId}")]
22     [Authorize(Roles = Role.Admin)]
23     public async Task Delete(long productId)
24     //otro código...
25 }
```

6.2.4. Implementación del paquete service facade

- **Implementación en Spring Boot**

Listado 6.10: Ejemplo de implementación de la capa servicios en Spring Boot

```
// Todas las clases service heredan de una clase base con todos los métodos CRUD comunes con su correspondiente control de errores
@Service
public class WarehouseService extends BaseService<Warehouse, Long,
  IWarehouseRepository> implements IWarehouseService {
{
  //Inyección de dependencias
  @Autowired
  private IntegrationEventLogService integrationEventService;

  @Autowired
  private RabbitTemplate rabbitTemplate;
  //otro código...

  //Implementa atomicidad al realizar las operaciones de la base de datos en una única transacción
  @Transactional
  public WarehouseDto update(WarehouseDto warehouseDto) {

    //el dto es validado
    updateValidationActions(warehouseDto);

    Warehouse warehouse = super.updateBase(warehouseDto);

    //el evento es creado
    UpdateWarehouseEvent event = map(warehouse, ↩
        ↪ UpdateWarehouseEvent.class);

    //el evento es almacenado en la tabla IntegrationEventLogContext
    integrationEventService.saveEvent(event);

    //el evento es publicado
    rabbitTemplate.convertAndSend(RabbitMqConfig.EXCHANGE_NAME,
      UpdateWarehouseEvent.class.getSimpleName(), event);

    return map(warehouse, WarehouseDto.class);
  }
    //otro código...
}
```

- **Implementación en ASP.Net Core**

Listado 6.11: Ejemplo de implementación de la capa servicios en .Net Core

```
1  // Todas las clases service heredan de una clase base con todos los métodos CRUD comunes
       con su correspondiente control de errores
2  public class CatalogProductService : ↩
       ↪ CatalogBaseService<CatalogProduct, long, ↩
       ↪ CatalogProductDto, ICatalogProductDao>, ↩
       ↪ ICatalogProductService
3  {
4    private readonly ICatalogIntegrationEventService ↩
         ↪ _catalogIntegrationEventService;
5
6    public CatalogProductService(
7      ICatalogProductDao catalogProductDao,
8      IMapper mapper,
9      ICatalogIntegrationEventService catalogIntegrationEventService
10     ) : base(catalogProductDao, mapper)
11   {
12     _catalogIntegrationEventService = ↩
           ↪ catalogIntegrationEventService;
13   }
14
15   //otro código...
16
17   public override async Task<CatalogProductDto> ↩
         ↪ CreateAsync(CatalogProductDto catalogProductDto)
18   {
19     var productCreated = await base.CreateAsync(catalogProductDto);
20
21     // Se crea el evento a partir del producto creado
22     var productCreatedEvent = ↩
           ↪ _mapper.Map<CreateProductEvent>(productCreated);
23
24     // El evento se almacena en la tabla IntegrationEventLogContext con el fin de asegurar
           la consistencia de datos del sistema
25     await _catalogIntegrationEventService
26       .SaveEventAndCatalogContextChangesAsync(productCreatedEvent);
27
28     // Se publica el evento para ser propagado por otros microservicios
29     await _catalogIntegrationEventService
30       .PublishThroughEventBusAsync(productCreatedEvent);
31
32     return productCreated;
33   }
34
35   //otro código...
36 }
```

Es recomendable que los datos del dominio y el evento se almacenen en la base de datos en la misma transacción con el fin de conseguir atomicidad, ya que se puede dar el caso de que se produzca un fallo en la transacción que actualiza el evento y el sistema quede en un estado inconsistente.

Listado 6.12: Implementando atomicidad y resistencia a fallos

```
public class CatalogIntegrationEventService : ↩
    ↪ ICatalogIntegrationEventService
{
  //otro código...

  public async Task ↩
      ↪ SaveEventAndCatalogContextChangesAsync(IntegrationEvent ↩
      ↪ evt)
  {
      //Estrategia de recuperación de errores, se reintentará en caso de fallo cada llamada a
          SaveChangesAsync() y a SaveEventAsync()
      await ResilientTransaction
        .New(_catalogContext).ExecuteAsync(async () =>
      {
          // Se implementa atomicidad al realizar las dos operaciones en la misma
              transacción.
          await _catalogContext.SaveChangesAsync();
          await _eventLogService.SaveEventAsync(evt, ↩
              ↪ _catalogContext.Database.CurrentTransaction);
      });
  }

  //otro código...
}
```

Más información sobre la recuperación de errores en las transacciones en el siguiente enlace: https://docs.microsoft.com/en-us/ef/core/miscellaneous/connection-resiliency

6.2.5. Implementación del paquete event handlers

A continuación se presenta un ejemplo de un event handler en Spring Boot y otro en .Net Core. Ambos siguen el patrón de diseño publish/subscribe, por lo que estos event handlers no generan una respuesta.

- **Implementación en Spring Boot**

Listado 6.13: Implementando de un event handler en Spring Boot

```
@Service
public class ProductUpdatedHandler implements ↩
    ↪ IIntegrationEventHandler<UpdateProductEvent> {
  private final Logger logger =
    LoggerFactory.getLogger(ProductUpdatedHandler.class);

  @Autowired
  private IProductService productService;

  @Override
  @Async
  @EventListener
  public void handle(UpdateProductEvent event) {
   logger.info("UpdateProductEvent␣received");
   productService.update(event);
  }
}
```

- **Implementación en ASP.Net Core**

Listado 6.14: Implementando de un event handler en .Net Core

```
public class ProductCreatedHandler :
  IIntegrationEventHandler<CreateProductEvent>
{
  private readonly IProductService _productService;

  public ProductCreatedHandler(IProductService productService)
  {
    _productService = productService;
  }

  public async Task HandleAsync(CreateProductEvent @event)
  {
    await _productService.CreateAsync(@event);
  }
}
```

Ejemplo de un event handler que implementa el método request/response asíncrono y si que devuelve una respuesta de tipo *ValidateInventoryEvent* a través del message broker.

Listado 6.15: Implementando de request/response asíncrono

```
@Service
public class ValidateInventoryRequest {

  @Autowired
  private IWarehouseStockService warehouseStockService;

  //API de mensajería con retorno de mensaje
  @RabbitListener(queues =
    RabbitMqConfig.QUEUE_REQUEST, concurrency = "10")
  public ValidateInventoryEvent validateInventory(
    ValidateInventoryEvent event) {

    return warehouseStockService.validateInventory(event);
  }
}
```

6.2.6. Implementación del paquete repository

El framework usado para el acceso a la base de datos en los microservicios en Spring Boot ha sido Hibernate + Jpa Repository.

Los métodos de consulta comunes son heredados de interface JPA Repository donde están predefinidos. Las consultas más complejas se han definido mediante queries utilizando la anotación @Query de Spring Data JPA. Con el fin de abstraer el lenguaje de consultas y hacer las consultas independientes a la tecnología de la base de datos, se ha utilizado JPQL.

A continuación se muestra un ejemplo en el que se utiliza @Query.

Listado 6.16: Ejemplo de implementando del paquete repository

```
1
2  public interface IWarehouseStockRepository extends ↩
       ↪ JpaRepository<WarehouseStock, Long> {
3
4    @Query("select new ↩
         ↪ clothingstorefranchise.spring.inventory.dtos.StockCountDto
5        (w.id.productId, w.id.size, sum(w.stock))"
6        + "from WarehouseStock w where w.id.productId in ?1 group ↩
             ↪ by w.id.size")
7    List<StockCountDto> countTotalProductStocks(List<Long> ↩
         ↪ productIds);
8
9    @Query("select new ↩
         ↪ clothingstorefranchise.spring.inventory.dtos.StockCountDto
10       (w.id.productId, w.id.size, sum(w.stock))"
11       + "from WarehouseStock w where w.id.productId = ?1 and ↩
             ↪ w.id.size = ?2 group by w.id.size")
12   StockCountDto countTotalProductStockByProductIdAndSize(Long ↩
         ↪ productId, int size);
13
14   @Query("select w from WarehouseStock w where w.id.productId = ↩
         ↪ ?1 and w.id.size = ?2 and w.stock >= ?3 order by ↩
         ↪ w.stock asc")
15   List<WarehouseStock> ↩
         ↪ findByProductIdAndSizeWhereStockIsGreaterThanQuantityAndStockIsMax
16     (Long productId, int size, Long quantity);
17
18   @Transactional
19   @Modifying //permite ejecutar INSERT, UPDATE, DELETE
20   @Query("delete from WarehouseStock w WHERE w.id.productId = ?1")
21   void deleteStockByProductId(Long productId);
22 }
```

6.2.7. Implementación del paquete dao

El framework usado para el acceso a la base de datos en los microservicios en .Net Core ha sido Entity Framework Core (EF). Este framework ofrece una abstracción de la tecnología de la base de datos.

Los métodos de consulta comunes son heredados de la clase base BaseAbstractEntitiesDao donde están predefinidos. Las consultas más complejas se realizan mediante el mecanismo de funciones lambda de EF.

A continuación se muestra un ejemplo del uso de funciones lambda en EF.

Listado 6.17: Ejemplo de implementando del paquete dao

```
public class SizeStockDao : BaseAbstractEntitiesDao<SizeStock, long, CustomersContext>, ISizeStockDao
{
  public SizeStockDao(CustomersContext contextContainer) : base(contextContainer) {}

  public async Task<SizeStock> FindByProductIdAndSizeWithEnoughStock(long productId, int size, long quantity)
  {
    //Uso de una función lambda para hacer la consulta
    return await FindSingleWhereAsync(stock => stock.Size == size && quantity  <= stock.Stock && stock.Product.Id == productId);
  }

  public async Task<SizeStock> FindByProductIdAndSize(long productId, int size)
  {
    //Uso de una función lambda para hacer la consulta
    return await FindSingleWhereAsync(stock => stock.Size == size && stock.Product.Id == productId);
  }

  //Carga de la entidad relacionada de forma EAGER
  protected override IQueryable<SizeStock> QueryTemplate()
  {
    return base.QueryTemplate()
      .Include(o => o.Product);
  }
}
```

6.2.8. Implementación de las bases de datos

Los microservicios en Spring Boot utilizan bases de datos MySql. El diseño de las bases de datos se ha realizado mediante una estrategia Code First. Al arrancar el microservicio el esquema de la base de datos será creado o actualizado automáticamente si no existe, tanto en entornos de desarrollo como en entornos de producción.

Los microservicios en .Net Core utilizan bases de datos Microsoft SQL Server. El diseño de las bases de datos se ha realizado mediante una estrategia Code First. Las migraciones de la base de datos son generadas mediante linea de comandos, a continuación se muestra un ejemplo en la consola de visual studio:

```
$ Add-Migration InitialCreate -Context DatabaseContext -OutputDir Migrations\CustomerMigrations
```

Se actualiza la base de datos con el siguiente comando:

```
$ Update-Database -Context DatabaseContext
```

Si se desea generar el fichero sql para revisarlo y ejecutarlo en producción ejecutar el siguiente comando:

```
$ Script-Migration -Context IntegrationEventLogContext -Output ↩
    ↪ ./integration_event_log_context.sql
```

6.3. APLICACIÓN WEB - FRONT END

En esta sección se presentan partes de la implementación front-end que se han considerado interesantes.

- **Angular material**

El diseño de la aplicación se ha realizado utilizando el módulo angula Angular Material. Angular Material es un módulo construido por y para Angular que permite implementar componentes Angular con un diseño basado en Material Design.

Para instalar Angular Material, el Kit de Desarrollo de Componentes y Angular Animations se ha ejecutado el siguiente comando en la carpeta raíz del código de la aplicación web:

```
$ ng add @angular/material
```

Este módulo facilita realizar el diseño de la IU mediante el uso de componentes prediseñados.

- **Local Storage**

La información básica de la sesión del usuario es almacenada en el navegador mediante un local storage service.

Listado 6.18: Implementación de LocalStorageService

```
1  @Injectable({ providedIn: 'root' })
2  export class LocalStorageService {
3    localStorage: Storage;
4    changes$ = new Subject();
5    constructor() {
6      this.localStorage   = window.localStorage;
7    }
8
9    //otro código
10 }
```

- **Http Service**

Se ha implementado un servicio común donde se implementan todos los métodos de HttpClient y su control de errores. Cada microservicio tiene su propio servicio en angular y en cada uno de estos servicios se ha inyectado este servicio común.

Listado 6.19: Servicio http común

```
@Injectable({ providedIn: 'root' })
export class HttpMethodsService {

  constructor(private http: HttpClient) { }

  get<T>(url: string, params?: any): Observable<T> {
        return this.http.get(`${environment.apiUrl}`+url, params)
        .pipe(
          tap((res: any) => {
            return res;
        }));
    }

  doPost<T>(url: string, data: T, needId: boolean, params?: any): Observable<T> {
        return this.http.post<T>(`${environment.apiUrl}`+url, data, params)
            .pipe(
              tap((res: any) => {
                return res;
            }));
    }

    delete(url: string, params?: any) {
        this.http.delete(`${environment.apiUrl}`+url, params)
            .subscribe((res) => {console.log('deleted');
        });
    }

    doPut<T>(url: string, data: any, params?: any): Observable<T> {
        return this.http.put<T>(`${environment.apiUrl}`+url, data, params)
        .pipe(
          tap((res: any) => {
            return res;
        }));
    }

  //otro código
}
```

- **Http Interceptor Service**

Este servicio intercepta todas las peticiones http que se producen o se reciben. Entre otras funciones, se encarga de inserta en cada petición el JWT en la cabecera junto a otros parámetros. También se encarga de la gestión de errores de las peticiones http.

Listado 6.20: Servicio http común

```
@Injectable()
export class HttpInterceptorService implements HttpInterceptor {

  constructor(private localStorageService: LocalStorageService) { }

  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

    // añade a la cabecera de la petición el jwt del usuario si está identificado.
    const token = this.localStorageService.get('jwt');
    const isApiUrl = request.url.startsWith(environment.apiUrl);
    if (token != null && isApiUrl) {

        request = request.clone({
            setHeaders: {
                Authorization: token
            }
        });
      }
        return next.handle(request)
        .pipe(
          catchError((error: HttpErrorResponse) => {
            console.error(error);
            return throwError(error);
          })
        )
  }
}
```

- **Gestión de la autorización**

El acceso a las URLs de la aplicación web ha sido controlada mediante autorización según el rol del usuario con Guards.

Listado 6.21: Gestión de la autorización, fichero *app-routing.module.ts*

```
1  // La carga de otros módulos se realiza de un modo LAZY
2  const salesModule = () => import('./sales/sales.module').then(x ↩
       ↪ => x.SalesModule);
3  const inventoryModule = () => ↩
       ↪ import('./inventory/inventory.module').then(x => ↩
       ↪ x.InventoryModule);
4  const catalogModule = () => ↩
       ↪ import('./catalog/catalog.module').then(x => ↩
       ↪ x.CatalogModule);
5  const employeesModule = () => ↩
       ↪ import('./employees/employees.module').then(x => ↩
       ↪ x.EmployeesModule);
6  const customersModule = () => ↩
       ↪ import('./customers/customers.module').then(x => ↩
       ↪ x.CustomersModule);
7  const accountModule = () => ↩
       ↪ import('./account/account.module').then(x => ↩
       ↪ x.AccountModule);
8
9  // Rutas raíz de los diferentes módulos que forman la aplicación
10 const appRoutes: Routes = [
11
12   {path: 'inventory', loadChildren: inventoryModule,
13     canActivate: [RoleGuard], data: {roles: `[${ROLES.Admin}, ↩
         ↪ ${ROLES.WarehouseEmployee}, ${ROLES.ShopEmployee}]`}},
14
15   {path: 'catalog', loadChildren: catalogModule,
16     canActivate: [RoleGuard], data: {roles: `[${ROLES.Admin}, ↩
         ↪ ${ROLES.Customer}, ${ROLES.Anonymous}]`}},
17
18   {path: 'account', loadChildren: accountModule},
19
20   {path: 'customers', loadChildren: customersModule,
21     canActivate: [RoleGuard], data: {roles: ↩
         ↪ `[${ROLES.Customer}, ${ROLES.Anonymous}]`}},
22
23   {path: 'employees', loadChildren: employeesModule,
24     canActivate: [RoleGuard], data: {roles: `[${ROLES.Admin}]`}},
25
26   {path: 'sales', loadChildren: salesModule,
27     canActivate: [RoleGuard], data: {roles: `[${ROLES.Admin}, ↩
         ↪ ${ROLES.Customer}, ${ROLES.WarehouseEmployee}]`}},
28
29   {path: '', canActivate: [DefaultRouteGuard], redirectTo: '', ↩
       ↪ pathMatch: 'full'},
30
31   {path: '**', canActivate: [DefaultRouteGuard], redirectTo: ↩
       ↪ '', pathMatch: 'full'}
32 ];
33
34 @NgModule({
35   imports: [RouterModule.forRoot(appRoutes)],
36   exports: [RouterModule]
37 })
38 export class AppRoutingModule {}
```

CAPÍTULO 7

PRUEBAS

7.1. PRUEBAS DE LOS END-POINTS

En la tabla siguiente se puede comprobar el código que tiene cada rol. Este código se utilizará en las tablas que se presentan en esta sección, para indicar los roles que tienen autorización a acceder a los servicios que se ofrecen.

Tabla 7.1: Esquema de código de roles

Rol	Código
Anonymous	0
Admin	1
Customer	2
Shop Employee	3
Warehouse Employee	4

7.1.1. Pruebas microservicio Authentication

La URL base a la API de este microservicio es */gateway/auth*

Tabla 7.2: Pruebas de Autenticación y Autorización

Descripción	**URL**	**Roles**	**Operación**	**Cuerpo Pet.**	**Código Resp.**	**Cuerpo Resp.**	**Resultado**
Realizar petición con token a URL permitida para el usuario	/gateway/**	1,2,3,4	*	*	Distinto a 401 o a 403	*	OK
Realizar petición con token a URL no permitida para el usuario	/gateway/**	1,2,3,4	*	*	403	“Forbidden”	OK
Realizar petición sin token a URL que requiere autenticación	/gateway/**	0	*	*	401	“No autorizado”	OK

Tabla 7.3: Pruebas realizadas a la API del microservicio Authentication

Descripción	**URL**	**Roles**	**Operación**	**Cuerpo Pet.**	**Código Resp.**	**Cuerpo Resp.**	**Resultado**
Registrar usuario	/user	0	POST	Usuario	201	Usuario	OK
Registrar usuario con campo obligatorio nulo	/user	0	POST	Usuario	400	Mensaje de error	OK
Registrar usuario con nombre de usuario existente	/user	0	POST	Usuario	409	Mensaje de error	OK
Consultar usuario por nombre de usuario	/user/ {username}	1,2,3,4	GET	Vacío	200	Usuario	OK
Consultar usuario por nombre de usuario que no existe	/user/ {username}	1,2,3,4	GET	Vacío	404	Mensaje de error	OK
Eliminar Empleado Almacén	/user/{id}	1	DELETE	Vacío	204	Mensaje de error	OK
Eliminar Empleado Almacén que no existe	/user/{id}	1	DELETE	Vacío	404	Mensaje de error	OK
Eliminar Empleado Tienda	/user/{id}	1	DELETE	Vacío	204	Mensaje de error	OK
Eliminar Empleado Tienda que no existe	/user/{id}	1	DELETE	Vacío	404	Mensaje de error	OK
Login usuario	/login	0	POST	Nombre de usuario y contraseña	201	Token	OK
Login usuario o contraseña incorrecta	/login	0	POST	Nombre de usuario y contraseña	401	Mensaje de error	OK

7.1.2. Pruebas microservicio Catalog

La URL base a la API de este microservicio es */gateway/catalog*

Tabla 7.4: Pruebas realizadas al controlador con la URL: */category*

Descripción	**URL**	**Roles**	**Operación**	**Cuerpo Pet.**	**Código Resp.**	**Cuerpo Resp.**	**Resultado**
Crear categoría	/	1	POST	Categoría	201	Categoría	OK
Crear subcategoría	/	1	POST	Subcategoría	201	Subcategoría	OK
Crear categoría existente	/	1	POST	Categoría	409	Mensaje de error	OK
Ver todas las categorías	/	0	GET	Vacío	200	Array de Categorías con sus subcategorías	OK
Ver productos de una subcategoría	/subcategory/{id}	0	GET	Vacío	200	Array productos	OK
Crear categoría con campo obligatorio nulo	/	0	GET	Categoría	400	Mensaje de error	OK
Ver productos de una subcategoría que no existe	/	0	GET	Vacío	Inesperado, debería ser 404	Mensaje de error	FALLO

Tabla 7.5: Pruebas realizadas al controlador con la URL: */catalog_products*

Descripción	**URL**	**Roles**	**Operación**	**Cuerpo Pet.**	**Código Resp.**	**Cuerpo Resp.**	**Resultado**
Crear producto	/	1	POST	Producto	201	Producto	OK
Crear producto con campo obligatorio nulo	/	1	POST	Producto	400	Mensaje de error	OK
Crear Producto existente	/	1	POST	Producto	409	Mensaje de error	OK
Actualizar producto	/	1	PUT	Producto	200	Producto	OK
Actualizar producto con campo obligatorio nulo	/	1	PUT	Producto	400	Producto	OK
Actualizar producto que no existe	/	1	PUT	Producto	404	Producto	OK
Consultar novedades	/	0	GET	Vacío	200	Array productos	OK
Obtener todos los productos	/	0	GET	Vacío	200	Array productos	OK

7.1.3. Pruebas microservicio Customers

La URL base a la API de este microservicio es */gateway/customers*

Tabla 7.6: Pruebas realizadas al controlador con la URL: */customers*

Descripción	URL	Roles	Operación	Cuerpo Pet.	Código Resp.	Cuerpo Resp.	Resultado
Crear cliente	/	0	POST	Cliente	201	Producto	OK
Crear cliente con campo nulo	/	0	POST	Cliente	400	Mensaje de error	OK
Crear cliente existente	/	0	POST	Cliente	409	Mensaje de error	OK
Consultar cliente por nombre de usuario	/	2	GET	Vacío	200	Cliente	OK

Tabla 7.7: Pruebas realizadas al controlador con la URL: */cart*

Descripción	URL	Roles	Operación	Cuerpo Pet.	Código Resp.	Cuerpo Resp.	Resultado
Actualizar cantidad producto del carro	/	2	PUT	Producto del carro	200	Producto del carro	OK
Actualizar cantidad negativa de producto	/	2	PUT	Producto del carro	400	Mensaje de error	OK
Obtener productos del carro cliente no identificado	/anonymous	0	PUT	Array productos del carro	200	Array productos del carro	OK
Modificar carro de la compra	/customer/{id}	2	PUT	Array productos del carro	200	Array productos del carro	OK
Consultar carro cliente identificado	/customer/{id}	2	GET	Vacío	200	Array productos del carro	OK
Obtener carro cliente identificado que no existe	/customer/{id}	2	DELETE	Vacío	Inesperado, debería ser 404	Mensaje de error	FALLO
Eliminar carro cliente	/customer/{id}	2	DELETE	Vacío	204	Vacío	OK
Eliminar producto del carro	/{productId}	2	DELETE	Vacío	204	Vacío	OK
Eliminar producto que no existe del carro	/{productId}	2	DELETE	Vacío	404	Mensaje de error	OK

7.1.4. Pruebas microservicio Employees

La URL base a la API de este microservicio es */gateway/employees*

Tabla 7.8: Pruebas realizadas a la API del microservicio Customers

Descripción	URL	Roles	Operación	Cuerpo Pet.	Código Resp.	Cuerpo Resp.	Resultado
Crear Empleado Tienda	/shop-employees	1	POST	Empleado	201	Empleado	OK
Crear Empleado tienda campo obligatorio nulo	/shop-employees	1	POST	Empleado	400	Mensaje de error	OK
Crear Empleado Tienda existente	/shop-employees	1	POST	Empleado	409	Mensaje de error	OK
Consultar Empleado tienda	/shop-employees/{id}	1, 3	GET	Vacío	200	Empleado	OK
Crear Empleado Almacén	/warehouse-employees	1	POST	Empleado	201	Empleado	OK
Crear Empleado almacén campo obligatorio nulo	/warehouse-employees	1	POST	Empleado	400	Mensaje de error	OK
Crear Empleado almacén existente	/warehouse-employees	1	POST	Empleado	409	Mensaje de error	OK
Consultar Empleado almacén	/warehouse-employees/{id}	1, 4	GET	Vacío	200	Empleado	OK
Obtener todos los almacenes	/warehouses	1	GET	Vacío	200	Array almacenes	OK
Obtener almacén	/warehouses/{id}	1	GET	Vacío	200	Almacén con empleados	OK
Obtener almacén que no existe	/warehouses/{id}	1	GET	Vacío	404	Mensaje de error	OK
Obtener todas las tiendas	/shops	1	GET	Vacío	200	Array shops	OK
Obtener tienda	/shops/{id}	1	GET	Vacío	200	Tienda con empleados	OK
Obtener tienda que no existe	/shops/{id}	1	GET	Vacío	404	Mensaje de error	OK

7.1.5. Pruebas microservicio Inventory

La URL base a la API de este microservicio es */gateway/inventory*

Tabla 7.9: Pruebas realizadas al controlador con la URL: */shops*

Descripción	URL	Roles	Operación	Cuerpo Pet.	Código Resp.	Cuerpo Resp.	Resultado
Crear tienda		1	POST	Tienda	201	Tienda	OK
Crear tienda campo obligatorio nulo		1	POST	Tienda	400	Mensaje de error	OK
Actualizar tienda			PUT	Tienda	200	Tienda	OK
Actualizar tienda campo obligatorio nulo		1	POST	Tienda	400	Mensaje de error	OK
Actualizar tienda que no existe		1	PUT	Tienda	404	Mensaje de error	OK
Consultar todas las tiendas		1	GET	Vacío	200	Array tienda	OK
Consultar tienda	/{id}	1,3	GET	Vacío	200	Tienda	OK
Consultar tienda que no existe	/{id}	1,3	GET	Vacío	404	Mensaje de error	OK
Consultar las tiendas que no tienen un producto	/without _product/ {productId}/	1	GET	Vacío	200	Array de tiendas	OK
Consultar las tiendas que no tienen un producto y el producto no existe	/without _product/ {productId}	1	GET	Vacío	Inesperado, debería ser 404	Vacío	FALLO
Asignar producto a tiendas	/product-allocation/ {productId}	1	PUT	Array tiendas	200	Array tiendas	OK
Asignar producto a tiendas y el producto no existe	/product-allocation/ {productId}	1	PUT	Array tiendas	404	Mensaje de error	OK

Tabla 7.10: Pruebas realizadas al controlador de api con la URL: */warehouses*

Descripción	**URL**	**Roles**	**Operación**	**Cuerpo Pet.**	**Código Resp.**	**Cuerpo Resp.**	**Resultado**
Crear almacén	/	1	POST	Almacén	201	Almacén	OK
Crear almacén campo obligatorio nulo	/	1	POST	Almacén	400	Mensaje de error	OK
Actualizar almacén	/	1	PUT	Almacén	200	Almacén	OK
Actualizar almacén campo obligatorio nulo	/	1	POST	Almacén	400	Mensaje de error	OK
Actualizar almacén que no existe	/	1	PUT	Almacén	404	Mensaje de error	OK
Consultar todos los almacenes	/	1	GET	Vacío	200	Array almacenes	OK
Consultar almacén	/{id}	1,4	GET	Vacío	200	Almacén	OK
Consultar almacén que no existe	/{id}	1,4	GET	Vacío	404	Mensaje de error	OK
Consultar los almacenes que no tienen un producto	/without _product/ {productId}	1	GET	Vacío	200	Array almacenes	OK
Consultar los almacenes que no tienen un producto y el producto no existe	/without _product/ {productId}	1	GET	Vacío	Inesperado, debería ser 404	Vacío	FALLO
Asignar producto a almacenes	/product-allocation/ {productId}	1	PUT	Array almacenes	200	Array almacenes	OK
Asignar producto a almacenes y el producto no existe	/product-allocation/ {productId}	1	PUT	Array almacenes	404	Mensaje de error	OK

Tabla 7.11: Pruebas realizadas al controlador de api con la URL: */products*

Descripción	**URL**	**Roles**	**Operación**	**Cuerpo Pet.**	**Código Resp.**	**Cuerpo Resp.**	**Resultado**
Consultar todos los productos	/	1	GET	Vacío	200	Array productos	OK
Consultar todo el stock de un producto	/{productId}/stocks	1	GET	Vacío	200	Stock	OK
Consultar todo el stock de un producto que no existe	/{productId}/stocks	1	GET	Vacío	Inesperado, debería ser 404	Vacío	FALLO
Consultar todo el stock de un producto en almacenes	/{productId}/stocks-without-warehouses	0	GET	Vacío	200	Stock	OK

7.1.6. Pruebas microservicio Sales

La URL base a la API de este microservicio es */gateway/sales*

Tabla 7.12: Pruebas realizadas a la API del microservicio Sales

Descripción	URL	Roles	Operación	Cuerpo Pet.	Código Resp.	Cuerpo Resp.	Resultado
Crear pedido	/orders	2	POST	Pedido	201	Pedido	OK
Crear pedido con campo obligatorio nulo	/orders	2	POST	Pedido	400	Mensaje de error	OK
Consultar todos los pedidos	/orders	1	GET	Vacío	200	Array de pedidos con productos	OK
Consultar pedidos de un cliente	/orders /customers /{customerId}	2	GET	Vacío	200	Array de pedidos con productos	OK
Actualizar un pedido producto	/order-products	1,4	PUT	Pedido Producto	200	Pedido Producto	OK
Actualizar un pedido producto que no existe	/order-products	1,4	PUT	Pedido Producto	404	Mensaje de error	OK
Actualizar un pedido producto con campo obligatorio nulo	/order-products	1,4	PUT	Pedido	400	Mensaje de error	OK
Consultar perdidos producto por estado	/state /{stateId}	1,4	GET	Vacío	200	Array de pedidos productos	OK
Consultar perdidos producto por almacén	/warehouse /{warehouseId}	1,4	GET	Vacío	200	Array de pedidos productos	OK
Consultar perdidos producto por estado y almacén	/warehouse /{warehouseId} /state /{stateId}	1,4	GET	Vacío	200	Array de pedidos productos	OK

7.2. PRUEBAS DE INTEGRACIÓN

Tabla 7.13: Pruebas de integración 1

Caso de uso	**Prueba realizada**	**Resultado esperado**	**Prueba superada**
Identificar usuario	Login credenciales correctas	Se carga la página por defecto según el rol del usuario	OK
	Login credenciales incorrectas	Se muestra un mensaje de usuario o contraseña incorrectos	OK
	Campo contraseña o username vacío	Mensaje indicando los campos requeridos	OK
	Fallo al conectar con el servidor	Mensaje de error	OK
Consultar categorías	El usuario selecciona la opción catálogo	El sistema muestra todas las categorías del catálogo	OK
Consultar subcategorías	El usuario selecciona una categoría	El sistema muestra sus subcategorías	OK
Consultar productos subcategoría	El usuario selecciona una subcategoría con productos	El sistema muestra los productos de la subcategoría	OK
	El usuario selecciona una subcategoría con productos	El sistema indica que la subcategoría no tiene productos	OK
Consultar existencias de productos en su almacén	El usuario selecciona consultar existencias	El sistema muestra todas las existencias de productos en su almacén	OK
	No hay existencias	El sistema indica que no hay existencias en su almacén	OK
Consultar pedidos de productos en su almacén	El usuario selecciona consultar pedidos de productos	El sistema muestra todos los pedidos de productos en su almacén	OK
	No hay pedidos de productos	El sistema indica que no hay pedidos de productos	OK
Cambiar estado pedido	El usuario selecciona cambiar estado a preparado	El pedido cambia del estado confirmado a preparado	OK
Consultar existencias de productos en su tienda	El usuario selecciona consultar existencias	El sistema muestra todas las existencias de productos en su tienda	OK
	No hay existencias	El sistema indica que no hay existencias en su tienda	OK

Tabla 7.14: Pruebas de integración 2

Caso de uso	**Prueba realizada**	**Resultado esperado**	**Prueba superada**
Crear categoría	El usuario selecciona crear categoría e introduce el nombre de la categoría	La categoría se muestra en la lista de categorías	OK
	No introduce el nombre de la categoría	Indica que el valor es requerido	OK
	La categoría ya existe	Un mensaje de advertencia	No implementado
	El usuario cancela	El caso de uso termina	OK
Crear subcategoría	El usuario selecciona crear categoría e introduce el nombre de la subcategoría y el tipo talla	La categoría se muestra en la lista de subcategorías	OK
	No introduce el nombre de la categoría o el tipo de talla	Indica que los valores son requeridos	OK
	La subcategoría ya existe	Un mensaje de advertencia	No implementado
	El usuario cancela	El caso de uso termina	OK
Crear producto	El usuario selecciona añadir producto y crea un nuevo producto	El sistema muestra el producto en la lista de productos de la subcategoría y se propaga a los microservicios Inventory y Customers	OK
	Campos obligatorios nulos o entrada incorrecta	El sistema indica el error en los campos implicados	OK
	El producto ya existe	Un mensaje de advertencia	No implementado
	El usuario cancela	El caso de uso queda sin efecto	OK
Modificar producto	El usuario selecciona editar producto y modifica un nuevo producto	El sistema muestra el producto modificado y propaga los cambios a los microservicios Inventory y Customers	OK
	Campos obligatorios nulos o entrada incorrecta	El sistema indica el error en los campos implicados	OK
	El usuario cancela	El caso de uso queda sin efecto	OK

Tabla 7.15: Pruebas de integración 3

Caso de uso	**Prueba realizada**	**Resultado esperado**	**Prueba superada**
Eliminar producto	No implementado	No implementado	-
Registrar empleado	El usuario registra un empleado correctamente	El empleado se muestra en la lista de empleados de la tienda o almacén	OK
	Campos obligatorios nulos o entrada incorrecta	El sistema indica el error en los campos implicados	OK
	El nombre de usuario ya existe	El sistema muestra un mensaje	OK
	El usuario cancela	El caso de uso queda sin efecto	OK
Consultar Empleados	El sistema selecciona consultar los empleados de un almacén o tienda	El sistema muestra la lista de empleados	OK
	No hay empleados	El sistema indica que no hay empleados en el almacén	OK
Eliminar empleado	El usuario selecciona eliminar un empleado	El empleado es eliminado de la lista y de los microservicios Authentication y Employees	OK
	El usuario cancela	El caso de uso queda sin efecto	OK
Actualizar empleado	El usuario actualiza un empleado	El sistema muestra el empleado actualizado	OK
	Campos obligatorios nulos o entrada incorrecta	El sistema indica el error en los campos implicados	OK
	El usuario cancela	El caso de uso queda sin efecto	OK
Crear tienda	El usuario selecciona crear una tienda y crea una tienda	El sistema crea la tienda y muestra la tienda en la lista de tiendas	OK
	Campos obligatorios nulos o entrada incorrecta	El sistema indica el error en los campos implicados	OK
	El usuario cancela	El caso de uso queda sin efecto	OK

Tabla 7.16: Pruebas de integración 4

Caso de uso	**Prueba realizada**	**Resultado esperado**	**Prueba superada**
Modificar información tienda	El usuario actualiza una tienda	El sistema actualiza la tienda y muestra la tienda actualizada en la lista de tiendas. Los datos son propagados al microservicio Employees	OK
	Campos obligatorios nulos o entrada incorrecta	El sistema indica el error en los campos implicados	OK
	El usuario cancela	El caso de uso queda sin efecto	OK
Consultar tiendas	El usuario hace clic en Tiendas	El sistema muestra una lista con todas las tiendas del sistema	OK
	No hay tiendas	El sistema indica que no hay tiendas en el sistema	OK
Eliminar tienda	No implementado	No implementado	-
Consultar stock tiendas	El usuario selecciona una tienda en la lista de tiendas	El sistema muestra el stock de productos en esa tienda	OK
	No hay stock	El sistema indica que no hay stock en la tienda seleccionada	OK
Consultar stock almacenes	El usuario selecciona un almacén en la lista de almacenes	El sistema muestra el stock de productos en ese almacén	OK
	No hay stock	El sistema indica que no hay stock en el almacén seleccionado	OK
Asignar productos a almacenes	El usuario selecciona un producto y asigna uno o más almacenes en los que no esté asignado el producto seleccionado	El sistema asigna el producto a los almacenes e inicializa su stock. Los datos se propagan al microservicio Customers	OK
	El usuario cancela	El caso de uso queda sin efecto	OK
Asignar productos a tiendas	El usuario selecciona un producto y lo asigna a una o más tiendas en las que no esté asignado el producto seleccionado	El sistema asigna el producto a las tiendas e inicializa su stock	OK
	El usuario cancela	El caso de uso queda sin efecto	OK
Consultar almacenes	El usuario hace clic en Almacenes	El sistema muestra una lista con todos los almacenes del sistema	OK
	No hay almacenes	El sistema indica que no hay almacenes en el sistema	OK

Tabla 7.17: Pruebas de integración 5

Caso de uso	**Prueba realizada**	**Resultado esperado**	**Prueba superada**
Crear almacén	El usuario crea un almacén	El sistema crea el almacén y muestra el almacén en la lista de almacenes	OK
	Campos obligatorios nulos o entrada incorrecta	El sistema indica el error en los campos implicados	OK
	El usuario cancela	El caso de uso queda sin efecto	OK
Actualizar datos almacén	El usuario actualiza un almacén	El sistema actualiza el almacén y muestra el almacén actualizado en la lista de almacenes. Los datos son propagados al microservicio Employees	OK
	Campos obligatorios nulos o entrada incorrecta	El sistema indica el error en los campos implicados	OK
	El usuario cancela	El caso de uso queda sin efecto	OK
Consultar pedidos por estado	El usuario hace click en consultar pedidos y selecciona el estado del pedido	El usuario muestra una lista con todos los pedidos de productos en el estado seleccionado	OK
	No hay pedidos de productos en el estado seleccionado	El sistema indica que no hay pedidos de productos en el estado seleccionado	OK
Registrar cliente	El usuario se registra como cliente correctamente	El sistema muestra un mensaje de éxito y redirecciona al usuario a la ventana de login indicando que debe iniciar sesión	OK
	Campos obligatorios nulos o entrada incorrecta	El sistema indica el error en los campos implicados	OK
	El nombre de usuario ya existe	El sistema muestra un mensaje de error indicando que ya existe ese nombre de usuario	OK
	El usuario cancela	El caso de uso queda sin efecto	OK
Consultar novedades	El usuario hace click en novedades	El sistema muestra los productos que son novedad	OK
	No hay productos novedad	El sistema indica que no hay novedades	OK
Consultar carro	El usuario selecciona el carro de la compra	El sistema muestra los productos en el carro de la compra	OK
	No hay productos en el carro de la compra	El sistema indica que el carro de la compra está vacío	OK
Modificar cantidad de producto en el carro	El usuario modifica la cantidad de un producto en el carro seleccionando una cantidad posible	El sistema actualiza la cantidad del producto en el carro, actualiza el numero total de productos en el carro y el precio	OK
Eliminar producto del carro	El usuario elimina un producto del carro	El sistema elimina el producto del carro, actualiza el número total de productos y el precio	OK

Tabla 7.18: Pruebas de integración 6

Caso de uso	**Prueba realizada**	**Resultado esperado**	**Prueba superada**
Añadir producto al carro	El usuario selecciona la talla de un producto y la cantidad, hace clic en "Añadir al carro"	El sistema añade el producto al carro y actualiza el carro. El indicador numérico de productos en el carro se incrementa.	OK
Obtener detalles stock talla producto	El usuario selecciona un producto con existencias en la tienda online	El sistema muestra los detalles de producto e indica que el producto se encuentra disponible en la tienda online	OK
	El usuario selecciona un producto sin existencias en la tienda online	El sistema muestra los detalles de producto e indica que el producto no se encuentra disponible en la tienda online	OK
	No hay existencias en la tienda online del producto para la talla seleccionada	El sistema indica que no hay existencias para la talla seleccionada. El botón "seleccionar cantidad" y el botón "añadir al carro" quedan bloqueados	OK
Consultar disponibilidad de la talla de un producto en tiendas físicas	El usuario selecciona una talla y una tienda física que tenga asignado el producto con existencias	El sistema indica que el producto se encuentra disponible en la tienda seleccionada	OK
	El usuario selecciona una talla y una tienda física que tenga asignado el producto sin existencias	El sistema indica que el producto no se encuentra disponible en la tienda seleccionada	OK
Realizar pedido	El usuario selecciona tramitar pedido y confirma el pedido con éxito	El sistema crea un pedido y disminuye el stock de ese producto en el inventario. El pedido queda en estado "Confirmado"	OK
	Un producto no tiene stock	El estado de ese pedido de producto queda en el esta "Pendiente"	OK
	El usuario modifica la tarjeta de crédito y la fecha de caducidad	El sistema actualiza la información del cliente	OK
	Campos obligatorios nulos o entrada incorrecta	El sistema indica el error en los campos implicados	OK
	El usuario cancela	El caso de uso queda sin efecto	OK
Consultar pedidos realizados	El usuario selecciona consultar pedidos	El sistema muestra una lista con todos los pedidos del usuario	OK
	No hay pedidos	El sistema indica que no hay pedidos realizados	OK

CAPÍTULO 8

DESPLIEGUE

El sistema esta formado por un message broker, una aplicación web implementada en Angular, tres microservicios implementados con la tecnología ASP.Net Core, tres microservicios implementados con la tecnología Spring Boot, y una Gateway con un Service Discovery externo que suponen otros dos microservicios implementados en Spring Boot.

Como se puede apreciar, el sistema consta de una amplia variedad de tecnologías, esto puede causar varios problemas, como por ejemplo problemas de compatibilidad a la hora de ejecutar los servicios en el entorno de ejecución o una implementación más laboriosa.

La tecnología de contenedores Docker es perfecta para una sistema de estas características, ya que proporcionan una gran portabilidad al poder ser ejecutados en cualquier lugar que tenga instalado un host de Docker. Estos contenedores permiten la ejecución de los microservicios de forma simple y aislada, haciendo un uso más eficiente de la infraestructura del entorno de ejecución.

Desplegar microservicios en contenedores es un proceso simple y directo que facilita el versionado. El código, sus dependencias, y el ejecutable se empaquetan en un archivo binario llamado *imagen docker*. Estas imágenes son almacenadas en un registro de contenedores, que actúa como un repositorio de librerías para imágenes. Un registro de contenedores puede encontrarse en un entorno de desarrollo personal, un centro de datos o en una cloud pública como Docker Hub.

Cuando sea necesario, la imagen docker se puede transformar en una o varias instancia de un contenedor ejecutable. Estos contenedores son inmutables y pueden ser recreados exactamente de la misma manera. Si alguna parte del sistema sufre algún cambio y necesita ser desplegada de nuevo, no será necesario reiniciar el despliegue del sistema completo, bastará con reiniciar el contenedor correspondiente.

Los servicios basados en contenedores pueden aprovechar los beneficios de escalado proporcionados por herramientas como Kubernetes. Por diseño, los contenedores solo se conocen a ellos mismos. Organizar un gran número de contenedores y sus dependencias compartidas, como las configuraciones de red puede ser complejo.

Kubernetes es una plataforma de orquestación de contenedores diseñada para automatizar el despliegue, escalar, y gestionar el despliegue de aplicaciones en contenedores. Crea una capa de abstracción por encima de los grupos de contenedores y los organiza en *pods*. Estos pods se ejecutan en maquinas llamadas *nodes*. El conjunto de grupos organizados es llamado *cluster*.

Kubernetes permite escalar la aplicación bajo demanda. Combinado esto con los microservicios en contenedores Docker, proporciona a las aplicaciones cloud la capacidad de responder de forma rápida y eficiente a las variaciones de demanda de recursos, asignando o reduciendo recursos a los distintos componentes del cluster según sea necesario.

8.1. DESPLIEGUE EN UN ENTORNO DE DESARROLLO

8.1.1. Paso 1 - Descargar el proyecto

El primer paso para desplegar la aplicación en un entorno de desarrollo local es descargar dentro de un mismo directorio, todos los repositorios de la siguiente organización GitHub:

https://github.com/ClothingStoreFranchise

8.1.2. Paso 2 - Crear imágenes Docker

La forma más rápida y sencilla de desplegar el sistema en un entorno de desarrollo local es mediante Docker. Para ello, es necesario tener instalado en el entorno de desarrollo un host de Docker.

Cada servicio contiene su propio Dockerfile. Un Dockerfile es un archivo de texto plano que contiene una serie de instrucciones necesarias para crear una imagen Docker. En el repositorio GitHub de cada microservicio se puede ver su respectivo Dockerfile.

Si se quiere generar la imagen a partir del Dockerfile, se debe ejecutar en el directorio donde se sitúa el Dockerfile el siguiente comando:

```
$docker build -t valdearados/clothing_store_franchise_customers .
```

Este comando genera en el repositorio de contenedores local una imagen docker para un microservicio. En el caso de los microservicios en Spring Boot no olvidar generar el archivo ejecutable .jar previamente.

Ejecutar el comando anterior en el directorio de los microservicios no es necesario a no ser que se quieran realizar modificaciones en algún microservicio. La imagen docker de cada uno de los microservicios han sido subidas a un repositorio público en el registro de contenedores cloud Docker Hub, por lo que pueden ser descargadas mediante el comando *docker pull [nombre-repositorio]* al repositorio de contenedores local.

8.1.3. Paso 3 - Crear y ejecutar contenedores Docker

El despliegue queda totalmente automatizado gracias a la herramienta Docker Compose. Docker Compose es una herramienta que permite simplificar el uso de Docker facilitando el proceso de crear contenedores, conectarlos, habilitar puertos, volúmenes, etc. En el archivo *docker-compose.yml* que se puede encontrar en la carpeta */deploy* del repositorio GitHub *settings* descargado anteriormente, se encuentran las instrucciones para desplegar el sistema completo en el host docker del entorno de desarrollo. Para desplegar el sistema con este fichero, se tiene que ejecutar el siguiente comando en la carpeta donde se encuentre el archivo:

```
$docker-compose -f docker-compose.yml up
```

8.2. DESPLIEGUE EN UN ENTORNO DE PRODUCCIÓN

Los entornos Cloud son ideales para las arquitecturas de microservicios debido a que ofrecen una alta escalabilidad. El proveedor de servicios de computación Cloud elegido para el despliegue del sistema en producción ha sido Google Cloud Platform (GCP). Al abrir una cuenta nueva en esta plataforma, se dispone de 300$ iniciales para probar sus productos, esto será suficientes para el despliegue temporal de este sistema.

Se ha utilizada la herramienta Google Kubernetes Engine (GKE) para realizar el despliegue. GKE permite crear y configurar los componentes mediante UI o mediante consola de comandos SDK de

Google Cloud. A continuación se explica como se ha desplegado y configurado el cluster Kubernetes en GKE. Se utilizarán ambos métodos para crear y configurar el cluster Kubernetes. Será necesario iniciar sesión en Google Cloud Platform y en la consola SDK.

8.2.1. Paso 1 - Crear y configurar un cluster Kubernetes

Lo primero es crear el cluster Kubernetes en el que se desplegará el sistema mediante el siguiente comando dentro de la consola SDK.

```
$ gcloud container clusters create [CLUSTER_NAME] ↩
    ↪ --release-channel --zone [COMPUTE_ZONE] --NODE-LOCATIONS ↩
    ↪ [COMPUTE_ZONE]
```

A continuación mediante la UI configuramos el cluster como se muestra en la Figura 8.1:

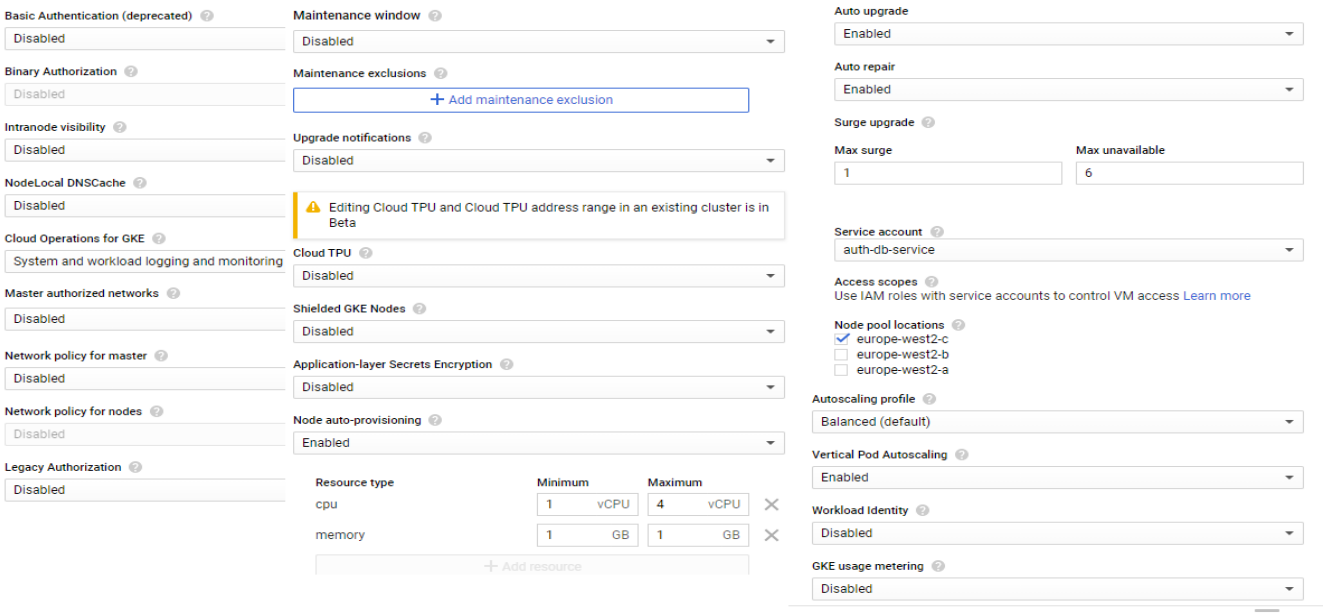

Figura 8.1: Configuración del cluster Kubernetes.

8.2.2. Paso 2 - Configurar el pool de nodos

El pool de nodos que tiene por defecto el cluster también debe ser configurado como se muestra en la Figura 8.2:

Como se puede observar, el cluster se ha configurado para permitir el auto escalado, de esta forma el propio cluster añade o quita recursos bajo demanda. Es importante para desplegar el sistema con éxito, aumentar los valores máximos de los recursos, ya que la configuración que trae el cluster por defecto, no proporciona recursos suficientes para desplegar la cantidad de servicios que se necesitan desplegar en este proyecto.

8.2.3. Paso 3 - Crear instancias SQL Cloud

A continuación se crean una instancia de SQL Cloud para cada uno de los microservicios. Para los microservicios implementados en Spring Boot el tipo de base de datos utilizado es MySql y para los implementados en .Net Core el tipo de base de datos es SQL Server. Para reducir latencias se recomienda que las bases de datos se desplieguen en la misma zona que el cluster Kubernetes. En la Figura 8.3 se pueden ver las instancias SQL desplegadas:

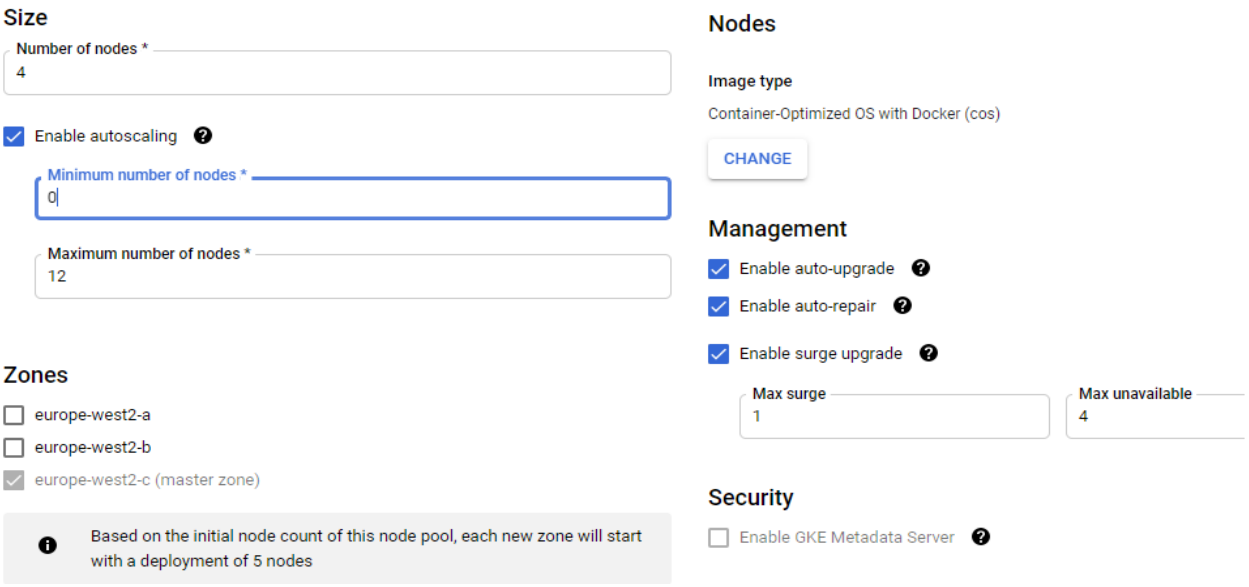

Figura 8.2: Configuración del pool de nodos del cluster Kubernetes.

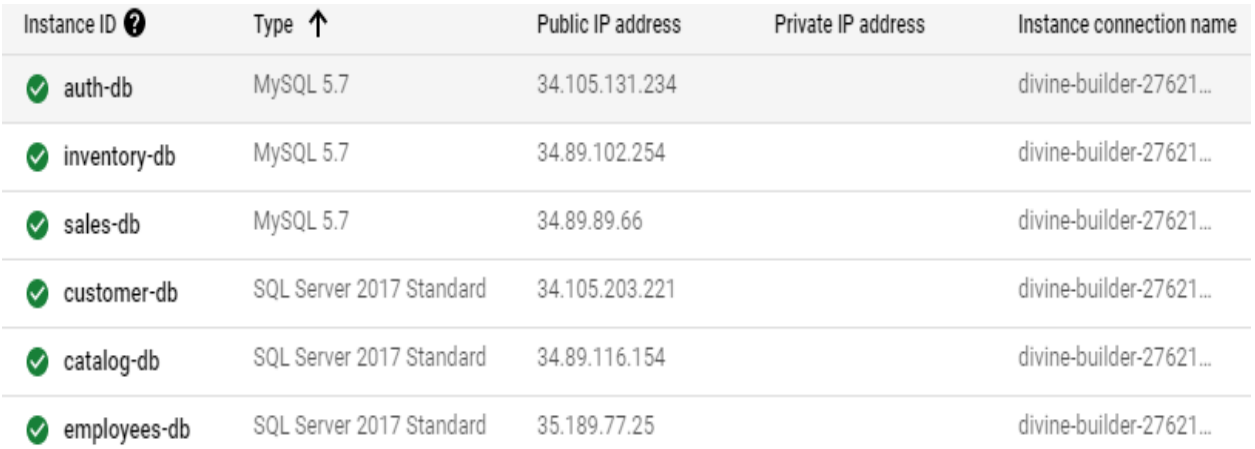

Figura 8.3: Bases de datos desplegadas en producción.

8.2.4. Paso 4 - Gestionar los usuarios de las bases de datos

Cada instancia de base de datos tiene un usuario por defecto. En cada una de las instancias se pueden consultar los usuarios que tienen acceso. Si se desea añadir un nuevo usuario a la base de datos, ejecutar el siguiente comando:

```
$ gcloud sql users create [NOMBRE_USUARIO] --host=% ↩
    ↪ --instance=[NOMBRE_INSTANCIA] --password=[PASSWORD]
```

8.2.5. Paso 5 - Conexión y autenticación a las bases de datos

Los microservicios se despliegan en pods. La conexión entre los pods y las bases de datos se realiza mediante el proxy de Cloud SQL, tal y como se recomienda en la documentación de Google Cloud. El proxy es un contenedor que es añadido dentro del pod en el que se encuentra el microservicio usando el patrón de contenedores *Sidecar*. El hecho de que el contenedor proxy este en el mismo pod que el microservicio, permite al microservicio conectarse al proxy mediante *localhost*, incrementando así la seguridad y el rendimiento.

Para autorizar la conexión entre los microservicios y las bases de datos, es necesario crear una cuenta de servicio en Google Cloud con el siguiente comando:

```
$ gcloud iam service-accounts create some-account-name
```

En la sección IAM en Google Cloud se podrá descargar un fichero JSON con las claves y la información de la cuenta de servicio que ha sido creada.

Una vez se han creado la cuenta de servicio y los usuarios en la base de datos, añadir los correspondiente *secrets* al entorno GKE:

```
$ kubectl create secret generic cloudsql-service-credentials ↩
    ↪ --from-file=credentials.json=[PATH-FICHERO-CUENTA-SERVICIO]
```

```
$ kubectl create secret generic cloudsql-db-credentials ↩
    ↪ --from-literal=username=[DATABASE-USERNAME] ↩
    ↪ --from-literal=password=[PASSWORD]
```

8.2.6. Paso 6 - Configuración y despliegue de los pods

El despliegue de los servicios del sistema se realizará mediante los ficheros *yaml* que se encuentran en el repositorio *settings* en el directorio */deploy*. Estos ficheros deben ser ejecutados en la consola SDK de la cuenta del propietario dentro del cluster de Kubernetes con el siguiente comando:

```
$ kubectl apply -f "${NOMBRE_ARCHIVO.yaml}"
```

En estos ficheros se encuentran las instrucciones y configuraciones para desplegar los microservicios en un entorno GKE. Configuraciones como el número de réplicas del pod, los puertos, la conexión a la base de datos mediante el proxy Cloud SQL, las conexiones internas entre pods mediante nombres de host, las conexiones externas mediante load blancer e IP pública. También, en estos ficheros se despliegan los contenedores que ejecutan los microservicios, las imágenes docker son descargadas del repositorio de contenedores Docker Hub.

La Gateway es el único microservicio que necesita ser accedido desde el exterior junto a la aplicación web Angular, por lo que son los únicos que necesitan una IP Pública. En el caso de que se quiera activar la IP Pública en los servicios como el message broker RabbitMQ o el service discovery Eureka Server con el fin de monitorizar su funcionamiento, hay que ejecutar el siguiente comando:

```
$ kubectl patch svc [NOMBRE_DEL_SERVICIO] --patch '{"spec": ↩
    ↪ {"type": "LoadBalancer"}}'
```

Para encontrar información más detallada sobre el despliegue en GKE recomiendo visitar la documentación de Google Cloud para GKE en el siguiente enlace https://cloud.google.com/kubernetes-engine/docs/.

En la Figura 8.4 se puede ver el diagrama de despliegue del sistema.

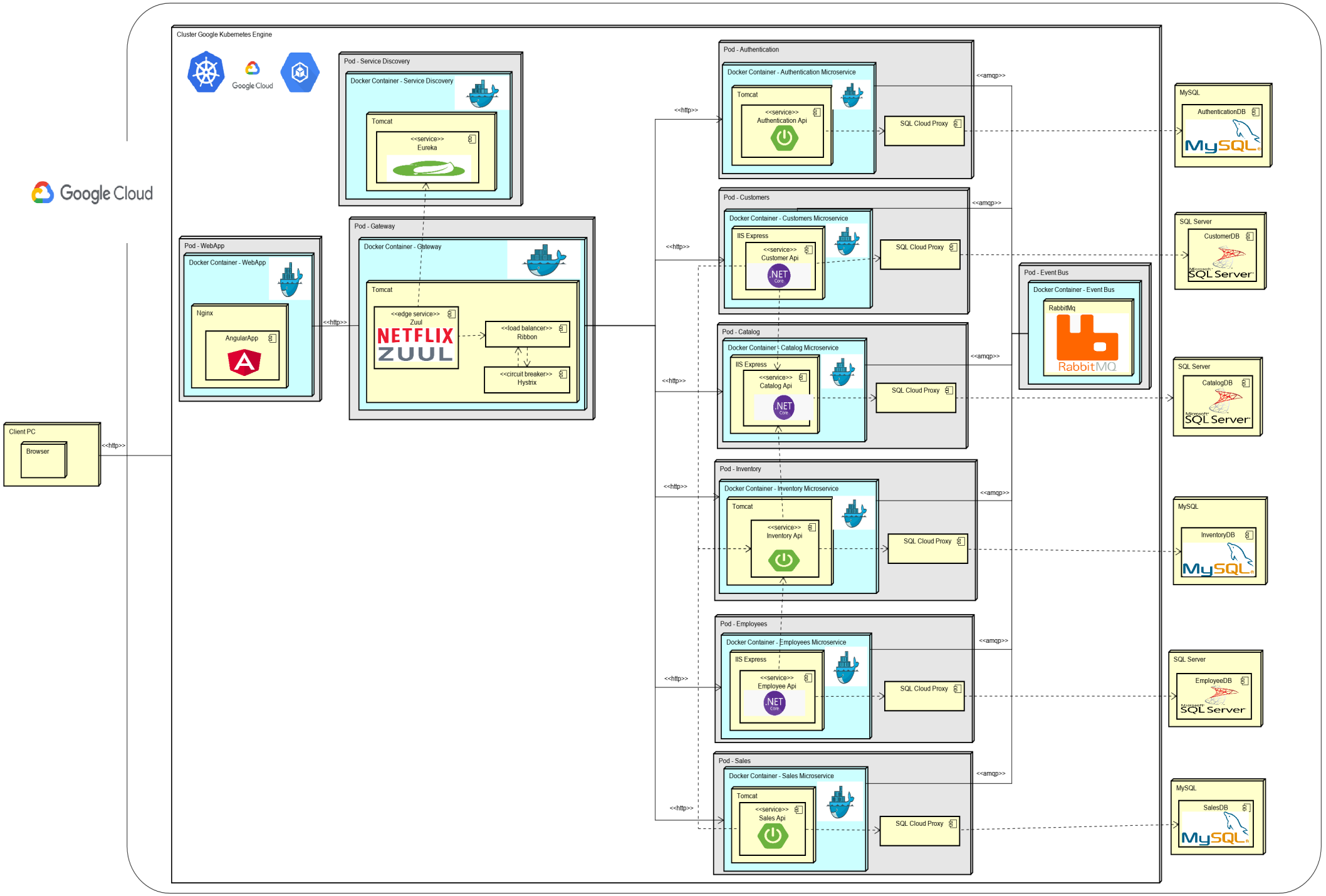

Figura 8.4: Diagrama de despliegue del sistema.

CAPÍTULO 9

CONCLUSIONES

9.1. CONCLUSIONES

Durante la realización de este proyecto se ha hecho un estudio avanzado de las arquitecturas de microservicios y se ha puesto en práctica los conocimientos aprendidos, realizando con éxito la implementación de una aplicación cloud políglota para una franquicia de tiendas de ropa, en la que se han seguido los principales patrones de diseño recomendados para este tipo de arquitecturas. Todo esto ha sido organizado en un proyecto de ingeniería de software en el que se han obtenido los siguientes resultados al final del proyecto:

- Se ha desarrollado una aplicación cloud políglota basada en una arquitectura de microservicios que ofrece una alta disponibilidad, una alta escalabilidad y tolerancia a fallos.
- Se han implementado los microservicios de forma políglota en .Net Core y en Spring boot, con el objetivo de poner en práctica las ventajas que ofrece la arquitectura y aprender tecnologías muy utilizadas en este tipo de arquitectura.
- Se ha implementado un mecanismo de autenticación propio basado en el microservicio *Athentication*, el cual se encarga de la gestión de la autorización y autenticación del usuario en el sistema mediante JWT.
- Se ha implementado una API Gateway propia que hace de punto de entrada a back-end, permitiendo la gestión del tráfico de peticiones http y facilitando junto al microservicio Authentication, la autenticación del usuario en todo el sistema distribuido.
- Se ha implementadao la comunicación entre el cliente web y back-end mediante RESTful.
- Se ha implementado una comunicación entre microservicios basada el protocolo AMQP mediante el message broker RabbitMQ, con el que se ha implementado un sistema Publish/Subscribe y el método de comunicación request/response asíncrono.
- Se ha conseguido que todos los microservicios se encuentren aislados e independientes.
- Se han seguido los patrones de diseño recomendados para que se conserve la consistencia de datos entre los microservicios distribuidos, sin sacrificar los beneficios que ofrece la arquitectura.
- Se ha implementado una aplicación web en Angular adaptada a la arquitectura de microservicios backend.
- Se ha desplegado el sistema de forma profesional en el entorno de producción Google Kubernetes Engine, consiguiendo así el despliegue de un sistema con una alta disponibilidad y

escalabilidad.

9.2. TRABAJO FUTURO

El sistema diseñado e implementado en este proyecto tiene unas dimensiones considerables. Esto lo hace un sistema basado en arquitectura de microservicios muy interesante para seguir estudiando y practicando la arquitectura, debido a las posibilidades de escalabilidad que ofrece. Por otro lado, al haber sido realizado por una única persona hay buenas prácticas, bugs y tests que no han sido implementados o solucionados. A continuación se menciona parte del trabajo que queda por hacer:

9.2.1. Mejoras técnicas

1. Realizar pruebas de rendimiento.

2. Implementar integración continua.

3. Implementar técnicas real-time de envío de notificaciones asíncrona desde el servidor al cliente web.

4. Completar los test de los microservicios implementando tests unitarios.

5. Completar la configuración del control de fallos en el sistema de colas del massage broker RabbitMQ, tal y como se recomienda en la documentación y realizar pruebas más profesionales en la comunicación entre microservicios.

6. Completar el patrón Event Sourcing, de tal modo que en cada microservicio haya procesos en segundo plano gestionando el estado de los eventos. Si se dispone de más tiempo, sería interesante realizar la implementación de tal modo que estos procesos en segundo plano fueran realizados por algún *Worker* externo, tal y como recomienda en los libros.

9.2.2. Mejoras funcionales

1. Mejorar la interfaz, mejorar el control de la entrada de datos del usuario y aumentar la cantidad de mensajes de información que notifican al usuario si la operación se ha completado con éxito o no.

2. Flexibilizar la forma en la que se ha implementado la relación entre empleado y tienda o almacén, permitiendo cambios de tiendas o almacenes.

3. Permitir al cliente y a los empleados consultar o modificar información de su cuenta, consultar pedidos por estado, valorar su compra o gestionar la venta en las tiendas físicas.

4. Ampliar el sistema con los microservicios de gestión de reparto o proveedores.

BIBLIOGRAFÍA

[1] Angular. (2021). "Angular documentation", [Online]. Available: https://dotnet.microsoft.com/apps/aspnet/microservices (visited on 01/04/2021).

[2] Docker. (2020). "Docker documentation", [Online]. Available: https://docs.docker.com/ (visited on 01/02/2021).

[3] EcuRed. (2020). «Proceso unificado de desarrollo», dirección: https://www.ecured.cu/Proceso_unificado_de_desarrollo (visitado 12-02-2021).

[4] Google. (2021). "Google cloud documentation", [Online]. Available: https://cloud.google.com/docs/ (visited on 02/05/2021).

[5] Indeed. (2021). «Salario programador Junior en España», dirección: https://es.indeed.com/career/programador-junior/salaries (visitado 13-02-2021).

[6] K. Indrasiri and P. Siriwardena, *Microservices for the Enterprise*. Apress Media LLC, 2018, ISBN: 978-1-4842-3858-5.

[7] Kubernetes. (2020). "Kubernetes documentation", [Online]. Available: https://kubernetes.io/docs/home/ (visited on 02/01/2021).

[8] C. Larman, *UML y Patrones: una introducción al análisis y diseño orientado a objetos y al proceso unificado*. PRENTICE HALL, 2002.

[9] Microsoft. (2020). "Asp.net documentation", [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-3.1 (visited on 02/15/2021).

[10] ——, (2020). "Microservices with .net", Microsoft, [Online]. Available: https://dotnet.microsoft.com/apps/aspnet/microservices (visited on 01/26/2020).

[11] RabbitMQ. (2020). "Rabbitmq documentation", [Online]. Available: https://www.rabbitmq.com/documentation.html (visited on 01/10/2021).

[12] C. Richardson, *Microservices Patterns*. Manning Publications Co., 2019, ISBN: 9781617294549.

[13] ——, (2021). "Microservice architecture", [Online]. Available: https://microservices.io/ (visited on 02/15/2021).

[14] J. Salido. (2010). «Curso: LaTeX esencial para preparación de TFG, Tesis y otros documentos académicos», Universidad de Castilla-La Mancha, dirección: http://visilab.etsii.uclm.es/?page_id=1468 (visitado 20-11-2020).

[15] Spring.io. (2020). "Spring projects", [Online]. Available: https://spring.io/projects (visited on 11/12/2020).

[16] C. de la Torre with Bill Wagner and M. Rousos. (2020). ".net microservices: Architecture for containerized .net applications", [Online]. Available: https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf.

ANEXOS

ANEXO A

MANUAL DE USUARIO

A.1. BANCO DE IMÁGENES

Todas las imágenes que aparecen en este manual y se han utilizadas en la aplicación web, han sido sacadas del banco de imágenes gratuito *Pexels* sin copyright. En el siguiente enlace se accede a *Pexels*, donde se puede encontrar más información sobre la licencia:

https://www.pexels.com/es-es/license/

A.2. OPERACIONES DE UN USUARIO ANÓNIMO

A.2.1. Página principal

Al acceder a la URL de la aplicación web lo primero que ve un usuario anónimo son los productos de la franquicia de ropa que son novedad. Si el usuario anónimo introduce una URL incorrecta dentro de la aplicación web o no está autorizado, será direccionado a la página principal.

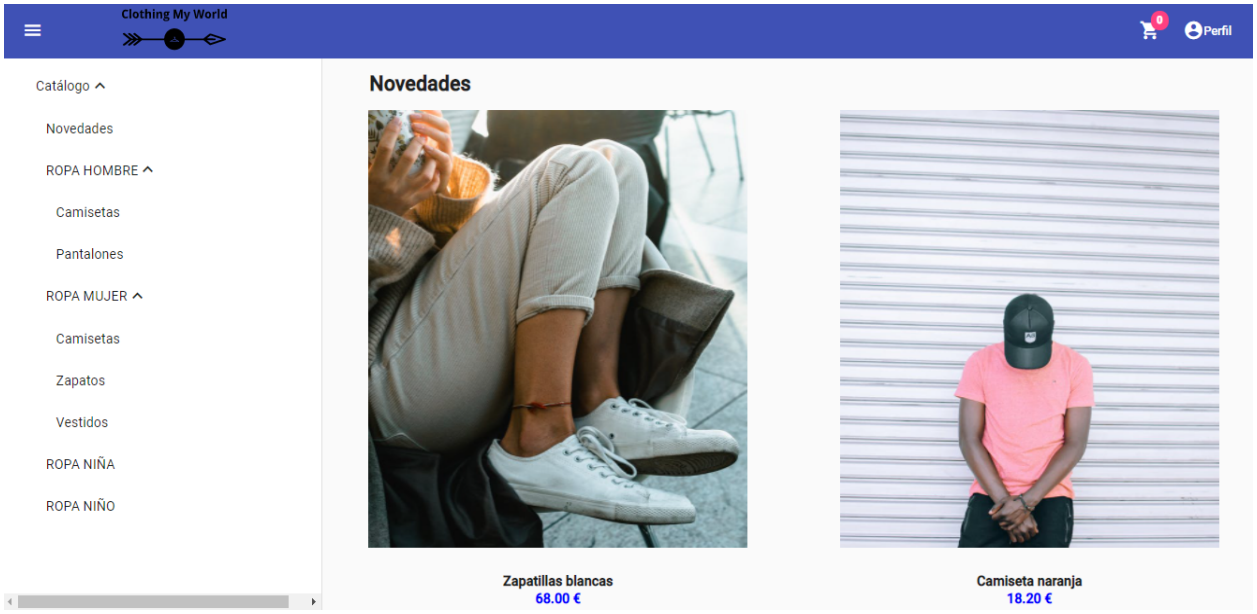

Figura A.1: Página principal usuario anónimo.

En la Figura A.1 podemos ver como se distribuyen los elementos de la interfaz. Un usuario anónimo puede navegar por las diferentes categorías de productos mediante el menú de navegación en la parte izquierda de la interfaz. En la parte superior de la UI, se encuentra una Toolbar estática

que da la opción al usuario de consultar su cuenta de usuario, el carro de la compra o mostrar/ocultar la barra de navegación izquierda.

La forma mediante la que el usuario puede regresar a la página principal, es haciendo clic en la opción *Novedades* en el menú de navegación o pulsando el logo de la franquicia de ropa que se encuentra en la Toolbar.

A.2.2. Consultar productos subcategoría

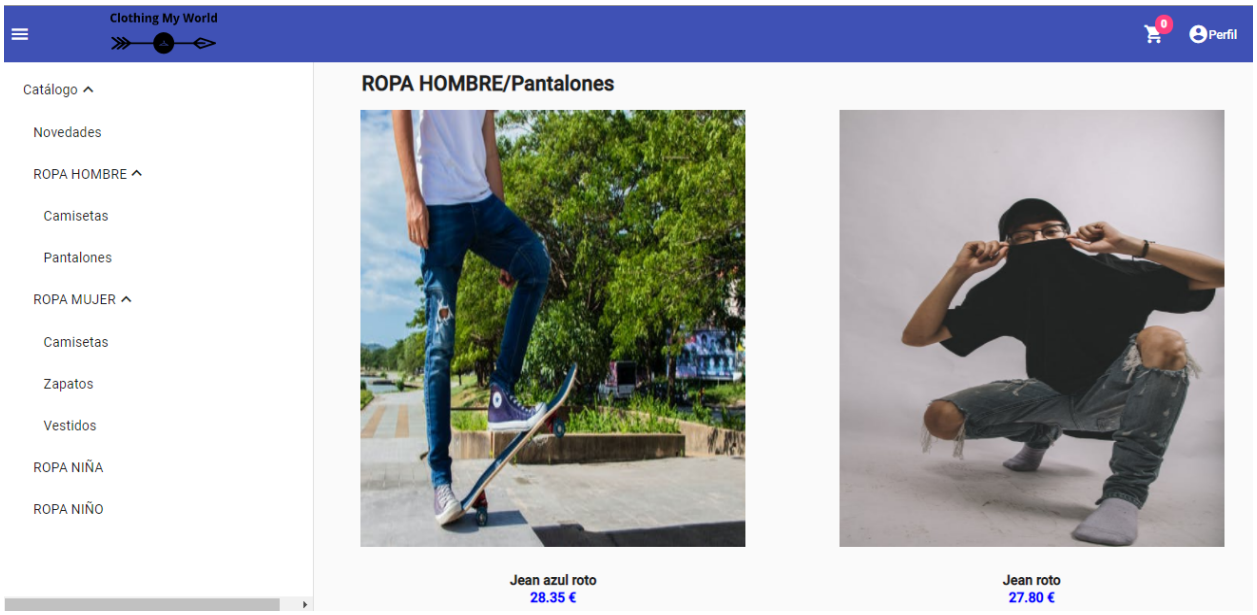

Figura A.2: Consultar productos subcategoría.

Para consultar los productos de la subcategoría deseada, el usuario debe buscar la subcategoría en el menú de navegación a la izquierda y hacer clic.

A.2.3. Consultar detalles y stock de un producto

Para consultar los detalles y el stock de un producto, hay que hacer clic en la imagen de algún producto de una subcategoría del catálogo. El resultado de hacer clic será el siguiente.

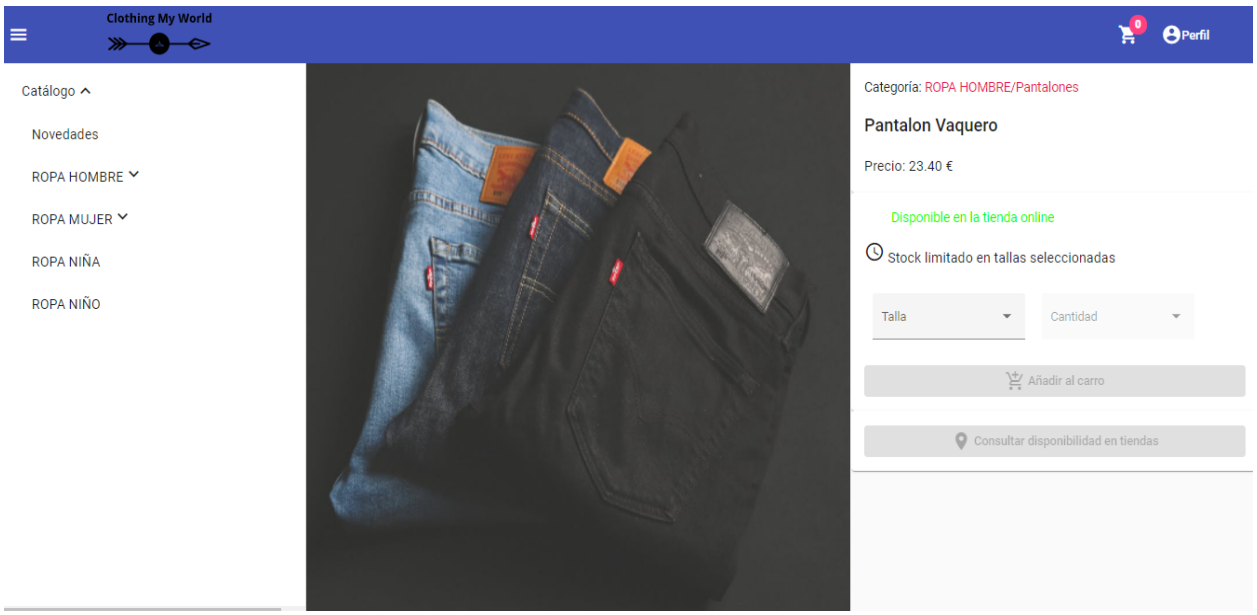

Figura A.3: Consultar detalles producto.

Como se puede ver los botones de UI están bloqueados, esto se debe a que no hay ninguna talla seleccionada.

A.2.4. Consultar disponibilidad de un producto en tiendas físicas

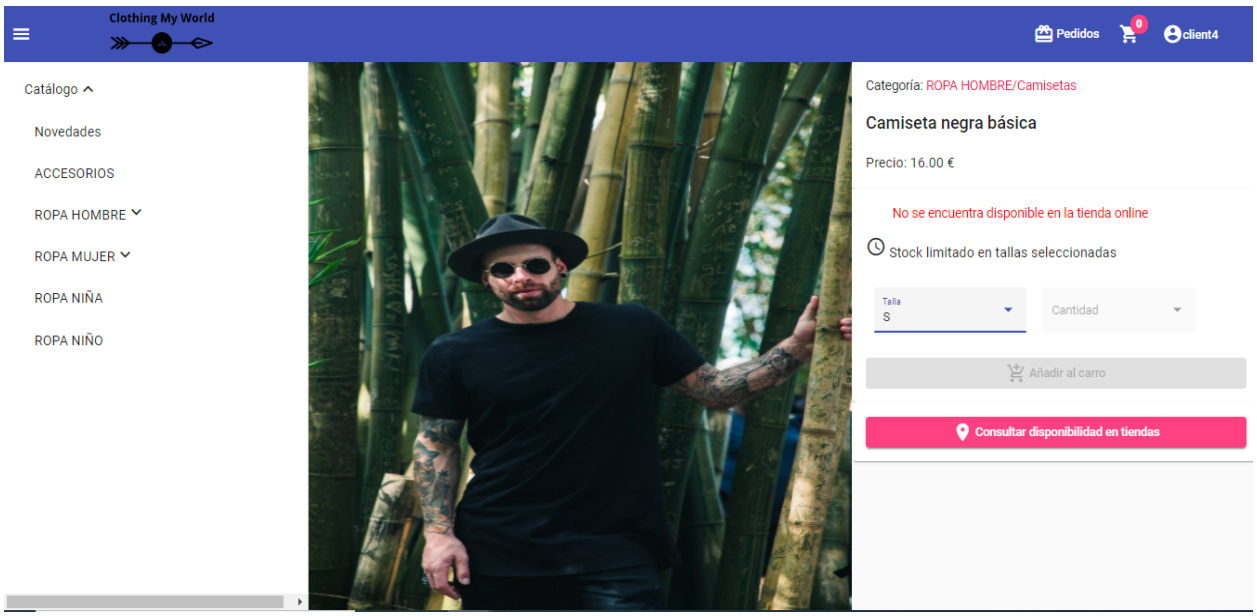

Figura A.4: Desbloquear botón *Consultar disponibilidad en tienda.*

Para consultar la disponibilidad de un producto en tiendas físicas es necesario seleccionar una talla del producto para desbloquear el botón *Consultar disponibilidad en tiendas.* Al hacer clic en el botón se despliega el siguiente pop-up.

Seleccionamos una de las tiendas que nos aparecen como opción y la aplicación indicara si se encuentra actualmente disponible.

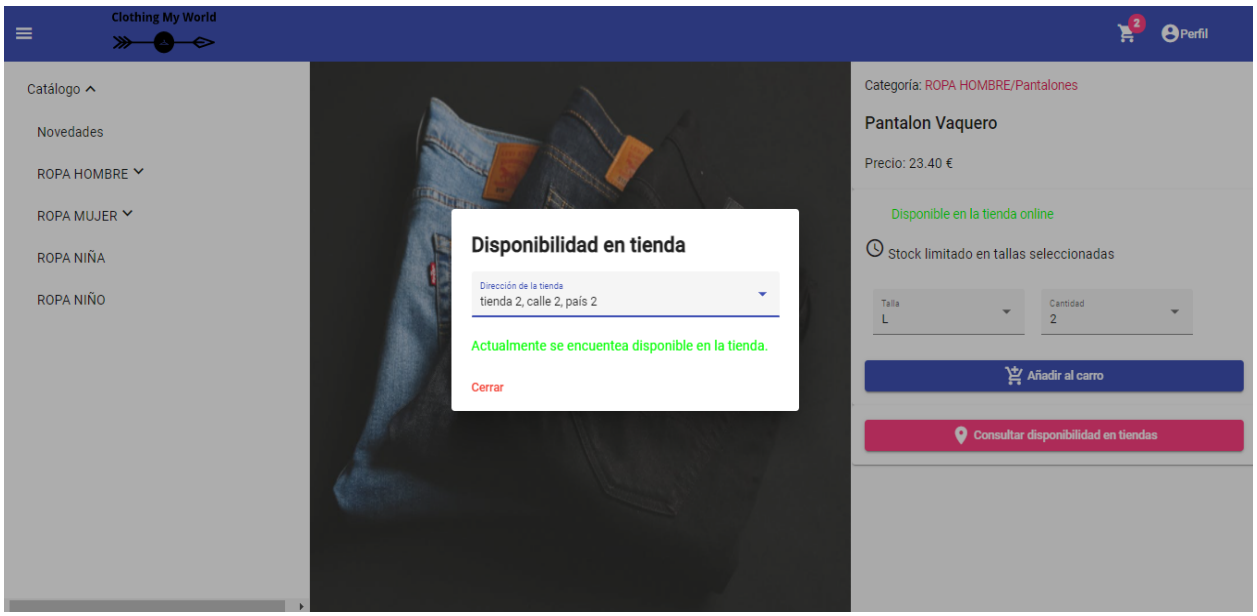

Figura A.5: Consultar la disponibilidad de un producto en tiendas físicas.

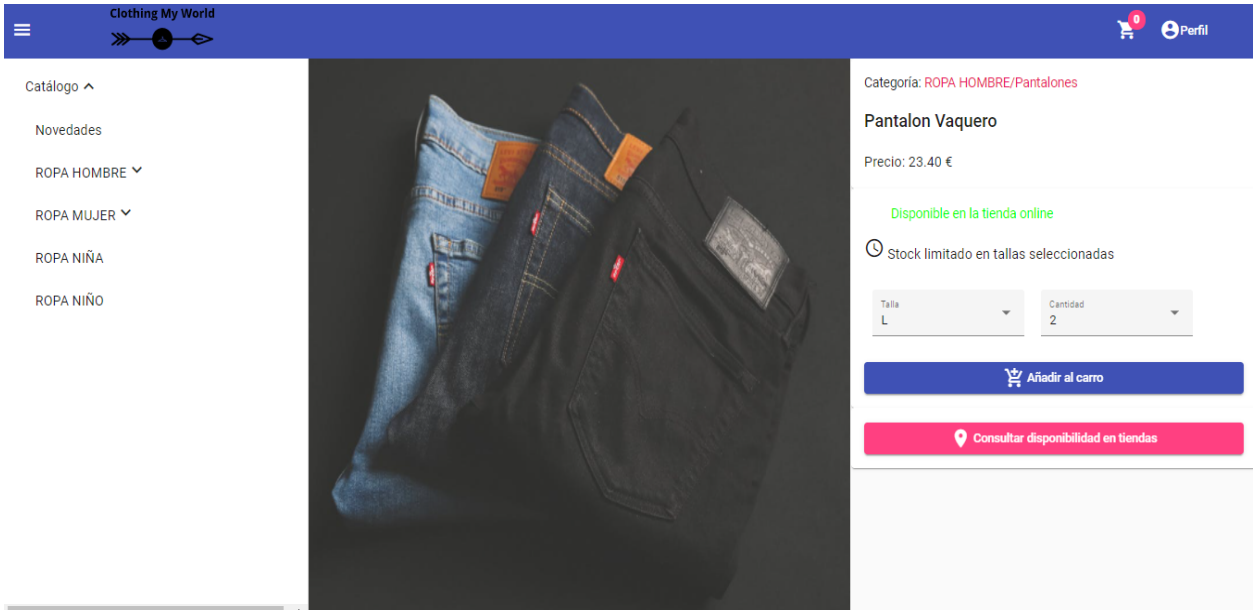

Figura A.6: Añadir producto al carro de la compra.

A.2.5. Añadir producto al carro

Para añadir un producto a carro, es necesario seleccionar una talla y una cantidad. Si el stock en la tienda online es suficiente, el botón *Añadir al carro* se desbloqueará y será posible añadir el producto al carro. Si el producto se ha añadido al carro con éxito, se mostrará una notificación y se incrementará el contador de productos en el carro.

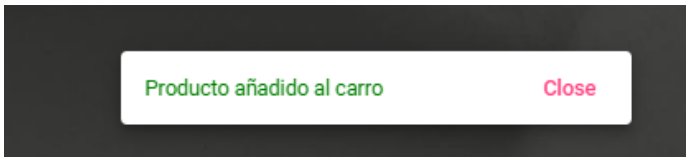

Figura A.7: Notificación producto añadido al carro.

Figura A.8: Contador productos en el carro.

A.2.6. Operaciones en el carro

Haciendo clic sobre el icono del carro de la compra en la parte derecha del Toolbar se accede al carro.

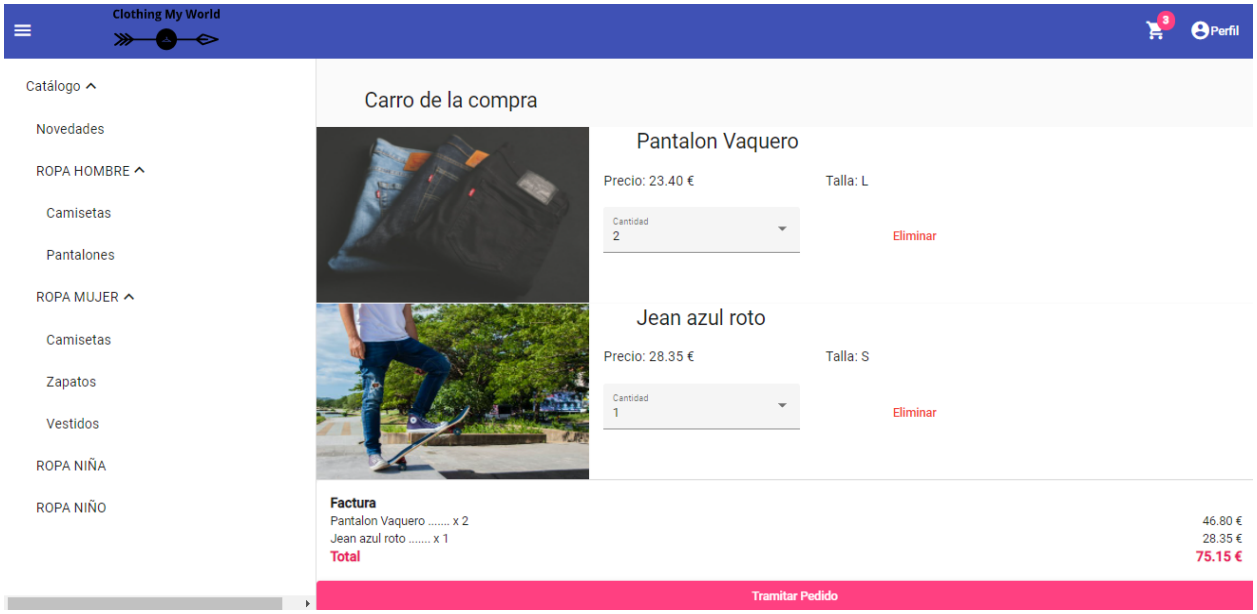

Figura A.9: Consultar el carro de la compra.

En esta ventana se puede eliminar un producto del carro o modificar la cantidad.

A.2.7. Iniciar sesión

Para iniciar sesión como usuario identificado, hay que hacer clic en el icono perfil en la parte derecha de la Toolbar y se abrirá la siguiente vista.

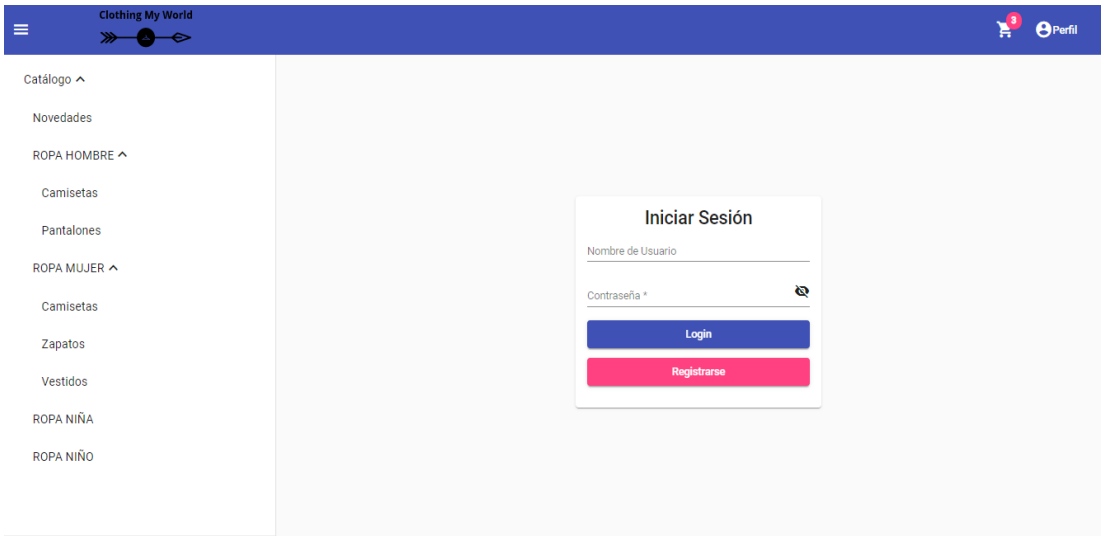

Figura A.10: Iniciar sesión.

No se permite iniciar sesión si alguno de los campos está vacío o la cantidad de caracteres de la contraseña es inferior a 4.

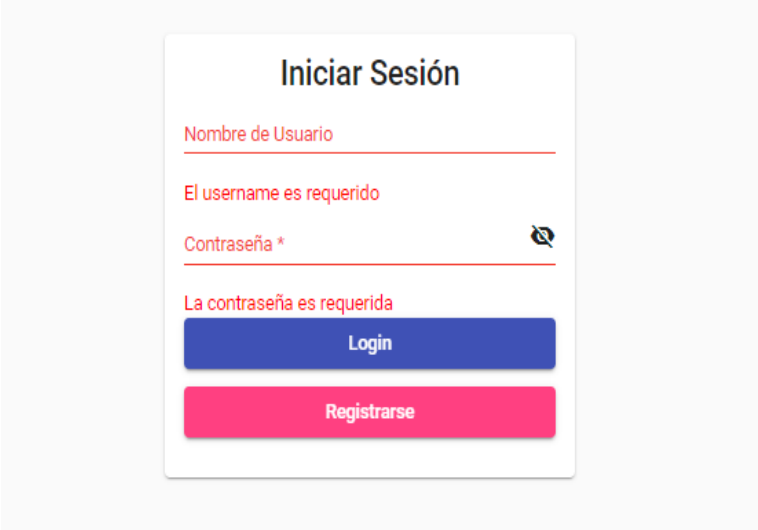

Figura A.11: Iniciar sesión campos erróneos.

Si el usuario o contraseña son incorrectos se muestra una notificación.

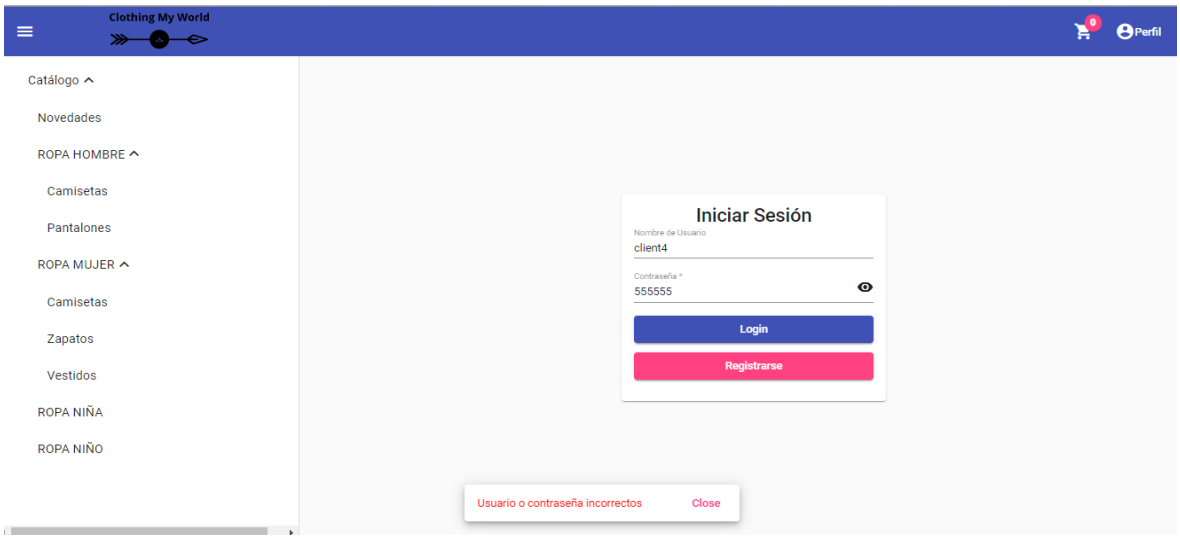

Figura A.12: Usuario o contraseña incorrectos.

A.2.8. Registrarse como cliente

Para registrase como cliente se debe pulsar el botón *Registrarse* que aparece en el vista de Iniciar Sesión, Figura A.10. Al hacer clic el botón se muestra la siguiente vista.

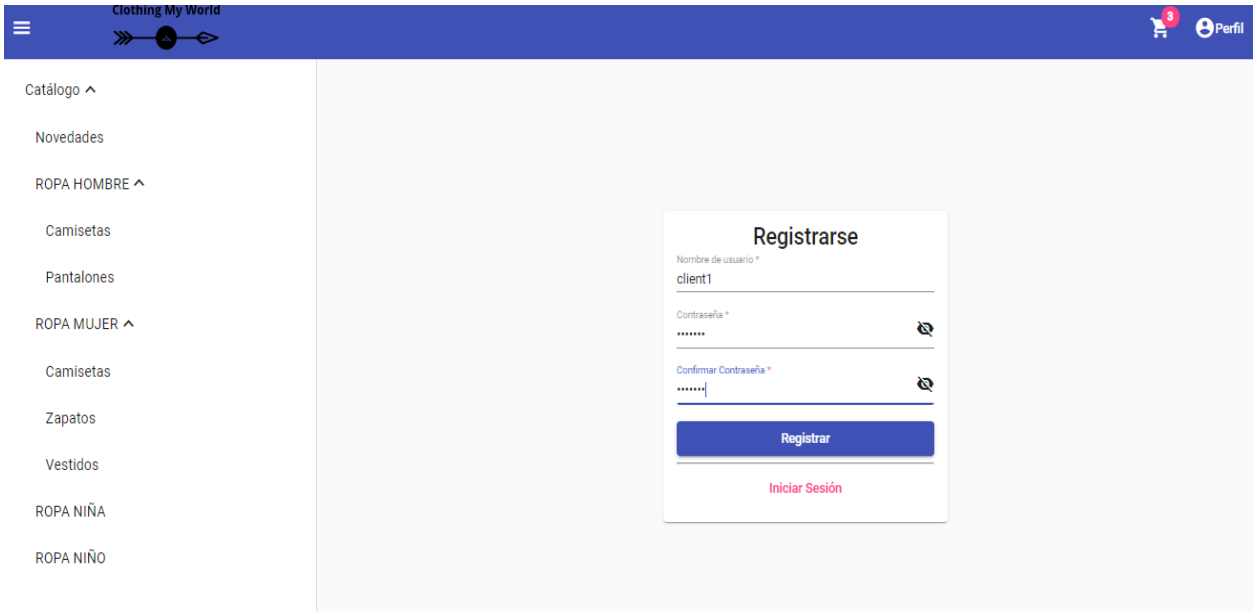

Figura A.13: Registrarse como cliente.

Si los datos introducidos son correctos y las contraseñas coinciden, se despliega el siguiente pop-up.

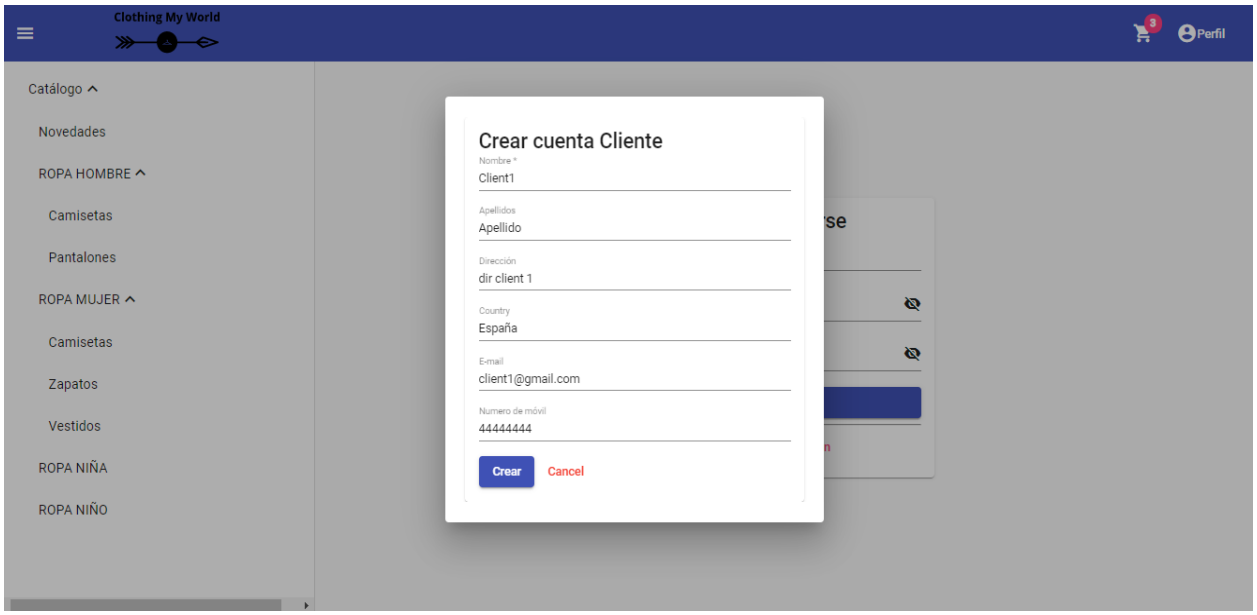

Figura A.14: Crear una cuenta cliente.

Ningún campo debe ser nulo y el formato del email debe ser correcto, en caso contrario se indicarán los errores.

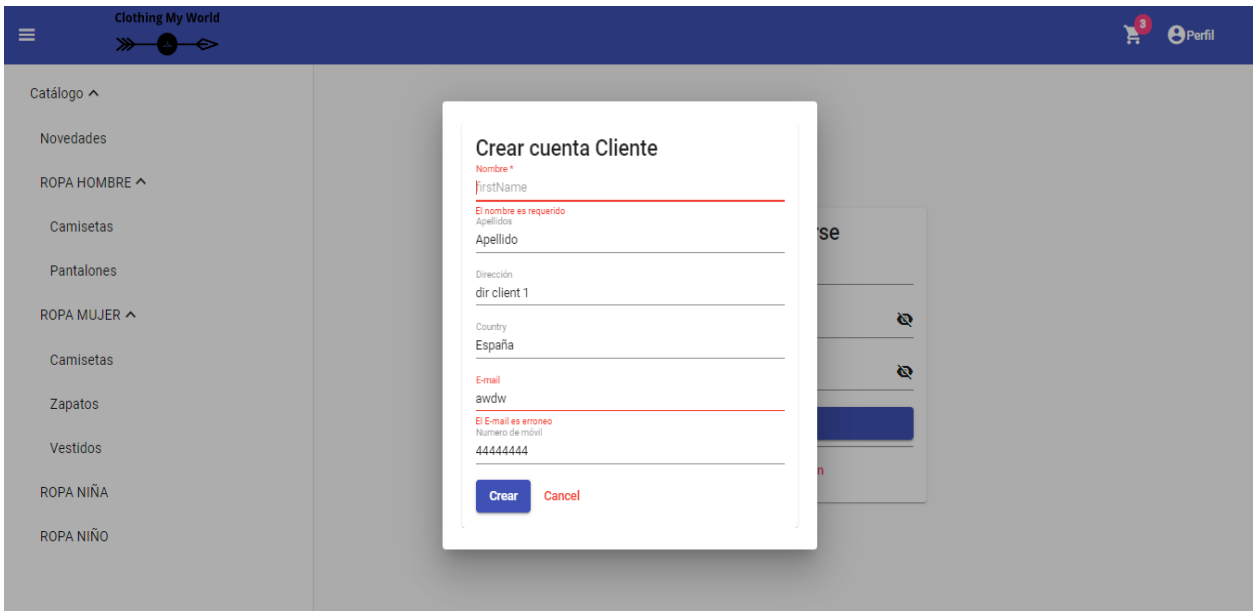

Figura A.15: Error campos crear una cuenta cliente.

Si el registro tiene éxito, el usuario será direccionado a la vista de Iniciar Sesión y será notificado. El carro de la compra será registrado.

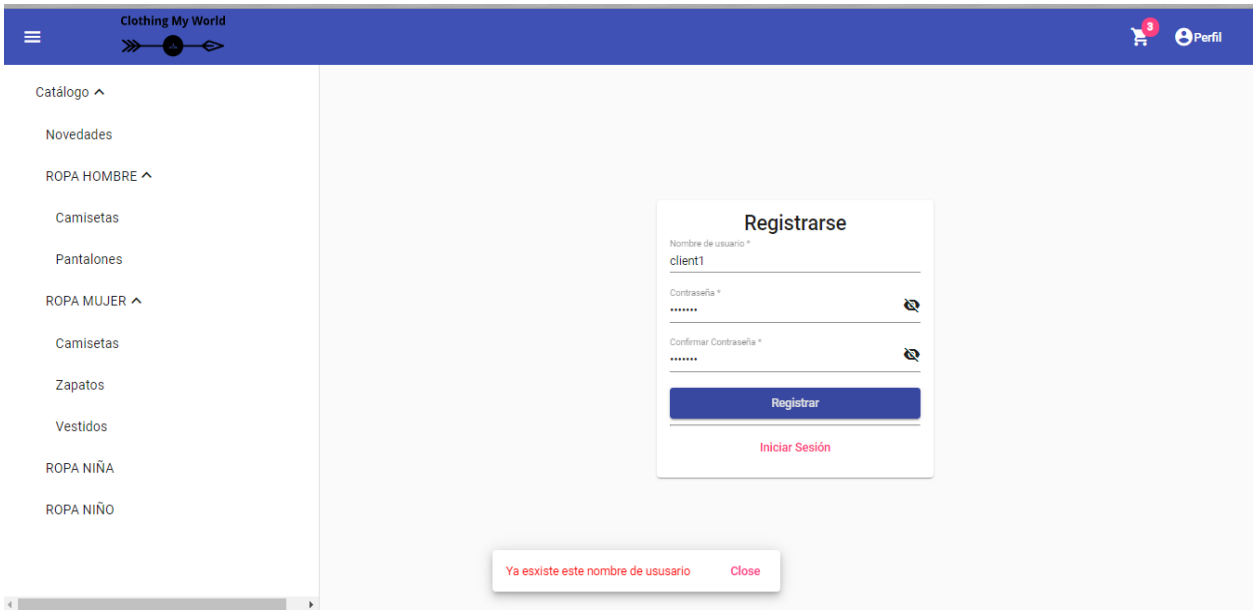

Figura A.16: El nombre de usuario ya existe.

En el caso de que el username ya exista se recibirá una notificación.

A.3. OPERACIONES COMO CLIENTE IDENTIFICADO

A.3.1. Página principal

La página principal es la misma que para el usuario anónimo, con la diferencia de que en la Toolbar hay un icono para consultar los pedidos y se muestra el username al lado del icono del perfil.

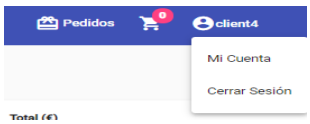

Figura A.17: Toolbar cliente identificado.

Para cerrar sesión el usuario hace clic sobre el icono perfil y posteriormente sobre *Cerrar Sesión*

A.3.2. Tramitar pedido

Para tramitar un pedido, se tiene que hacer clic en el botón *Tramitar pedido* en la vista del carro de la compra Figura A.9 y se abrirá el pop-up siguiente.

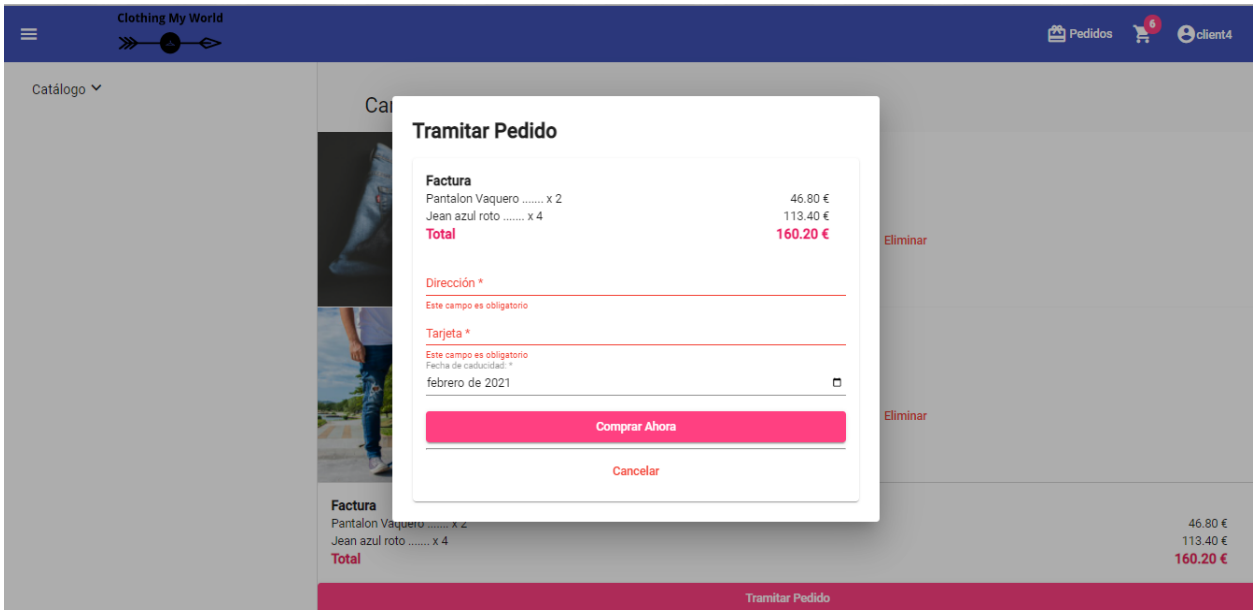

Figura A.18: Tramitar pedido.

El campo de la tarjeta y la dirección de entrega no deben ser nulos.

A.3.3. Consultar pedidos realizados

Para consultar los pedidos realizados hay que hacer clic en el icono pedidos en la Toolbar.

A.4. OPERACIONES COMO EMPLEADO

A.4.1. Página principal

La página que se muestra a un Empleado Almacén al iniciar sesión, es el inventario de su almacén. Todo intento de acceso a URL incorrecta o no autorizada será direccionada a esta vista. Pulsando el logo de la Toolbar también se puede acceder a esta vista.

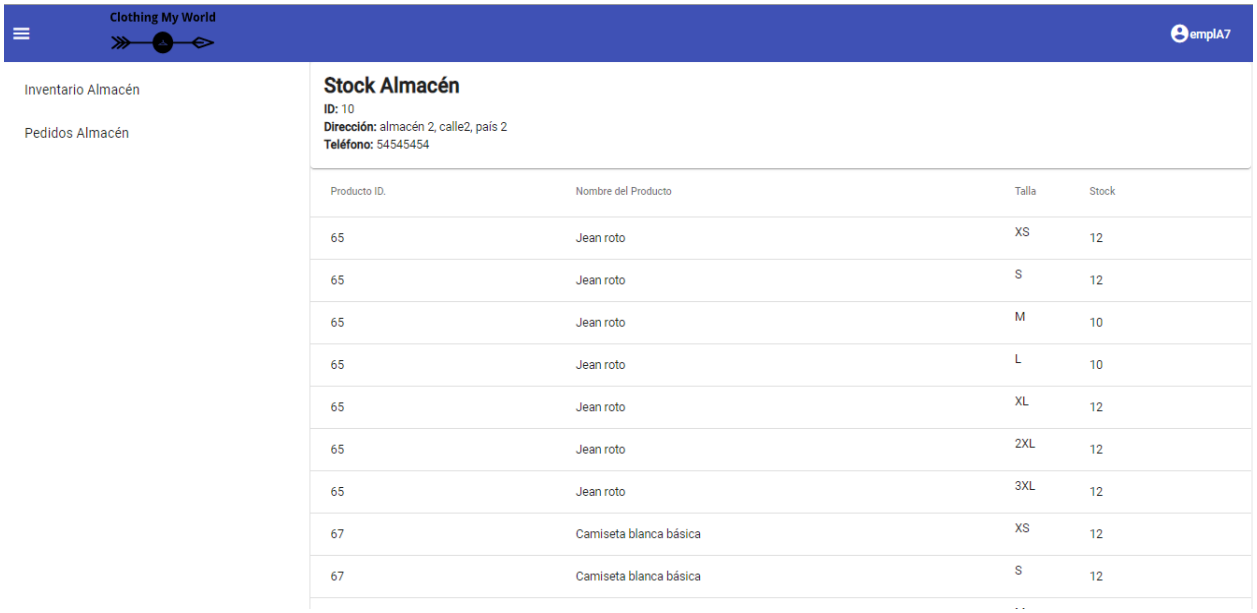

Figura A.19: Inventario en almacén.

La página que se muestra a un Empleado Almacén por defecto, es el inventario de su tienda. Todo intento de acceso a una URL incorrecta o no autorizada será direccionada a esta vista. Pulsando el logo de la Toolbar también se puede acceder a esta vista.

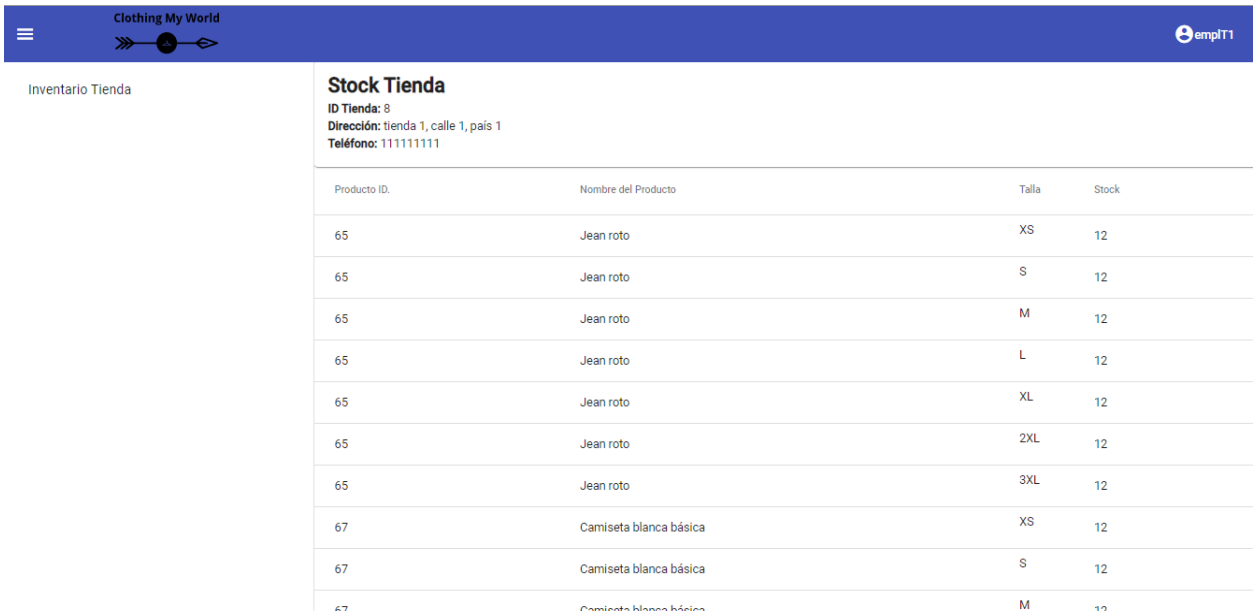

Figura A.20: Inventario en tienda.

A.4.2. Operaciones como empleado Almacén

- Consultar inventario de su almacén

Para consultar el inventario del almacén hay que hacer clic en el menú de navegación en *Inventario almacén* y se mostrará la vista de la Figura A.20.

- Consultar pedidos al almacén

Para consultar todos los pedidos a su almacén hay que hacer clic en el menú de navegación en *Pedidos almacén* y se mostrará la vista siguiente.

Figura A.21: Inventario en tienda.

- Preparar pedido

Los pedidos en estado Confirmado tienen visible el botón *Preparar pedido*, tal y como ocurre en la Figura A.20. Al hacer clic en el botón el pedido pasa ha estado Preparado.

A.4.3. Operaciones como empleado Tienda

- Consultar inventario de su tienda

Para consultar el inventario de la tienda hay que hacer clic en el menú de navegación en *Inventario Tienda* y se mostrará la vista de la Figura A.20.

A.5. OPERACIONES COMO ADMINISTRADOR

A.5.1. Página principal

La página que se muestra a un Administrador por defecto, es el inventario de productos. Todo intento de acceso a URL incorrecta o no autorizada será direccionada a esta vista.

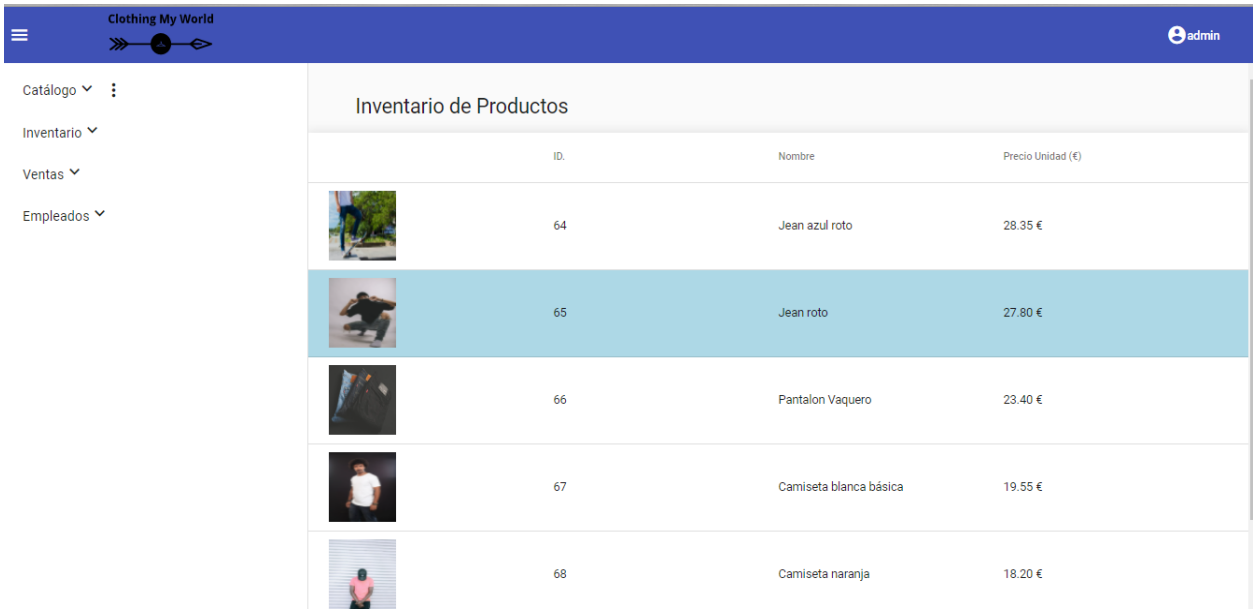

Figura A.22: Inventario de productos.

Pulsando el logo de la Toolbar también se puede acceder a esta vista.

Crear categoría y subcategoría

Para crear una categoría hay que pulsar en el botón que está en la opción *Catálogo* en el menú de navegación que se encuentra a la izquierda de la UI. Al pulsar dicho botón se muestra el siguiente menú.

Figura A.23: Menú Categorías.

Dentro del menú hacer clic en *Crear Categoría* y se mostrará la siguiente vista.

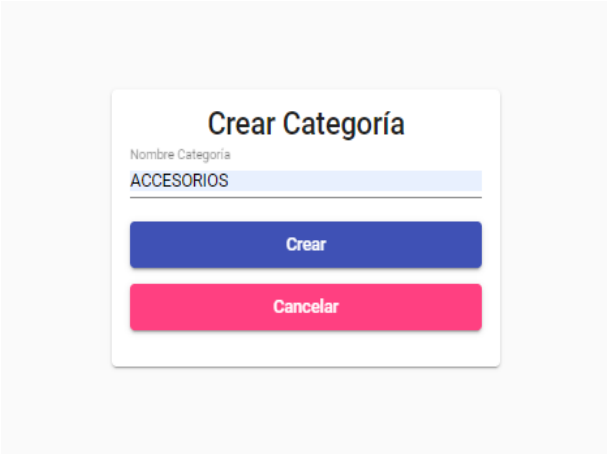

Figura A.24: Crear categoría

La categoría creada será añadida al menú de navegación en el lado izquierdo de la UI.

Figura A.25: Categoría ACCESORIOS creada

Para crear una subcategoría dentro de una categoría, hacer clic en el botón de la categoría deseada y se mostrará el siguiente menú.

Figura A.26: Menú crear subcategoría

Dentro del menú hacer clic en *Crear Subcategoría* y se mostrara la siguiente vista.

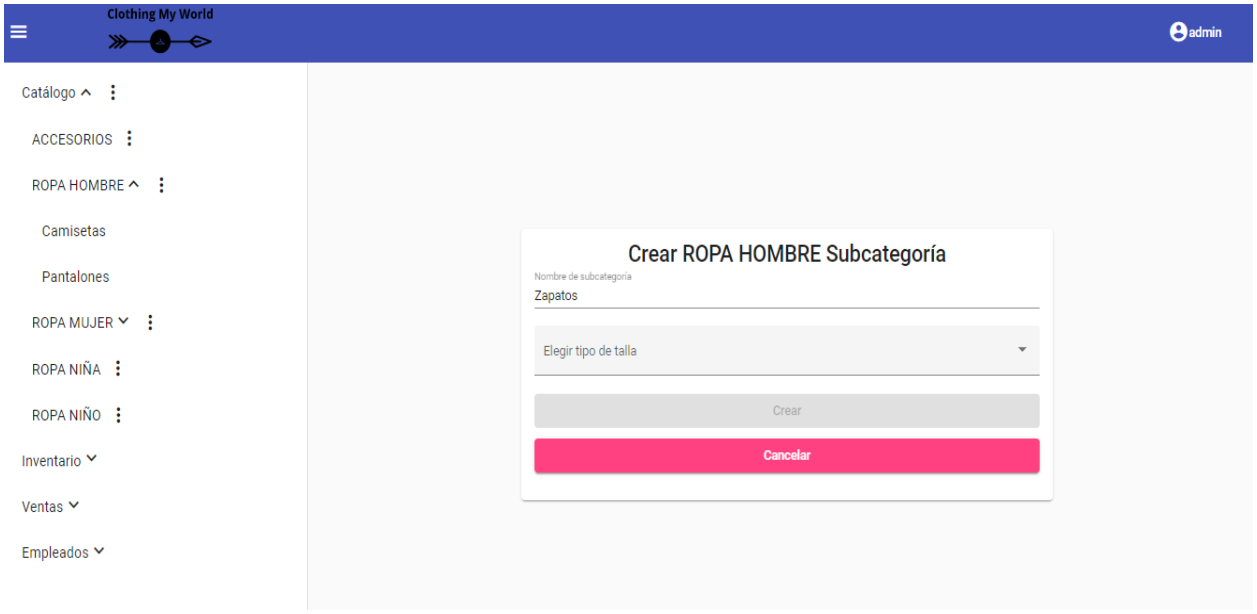

Figura A.27: Crear subcategoría.

Una vez se introduzca el nombre de la subcategoría y se seleccione el tipo de talla de los productos, se activara el botón *Crear* y se podrá crear una subcategoría.

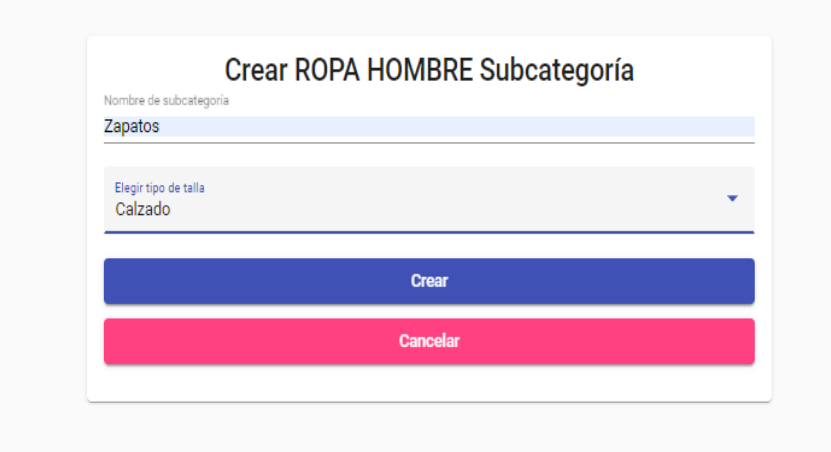

Figura A.28: Crear subcategoría botón activado.

Una vez la subcategoría ha sido creada, aparecerá dentro de la categoría a la que corresponde en el menú de navegación.

Crear y editar producto

Para crear un producto, en primer lugar hay que seleccionar la subcategoría a la que va a pertenecer el producto.

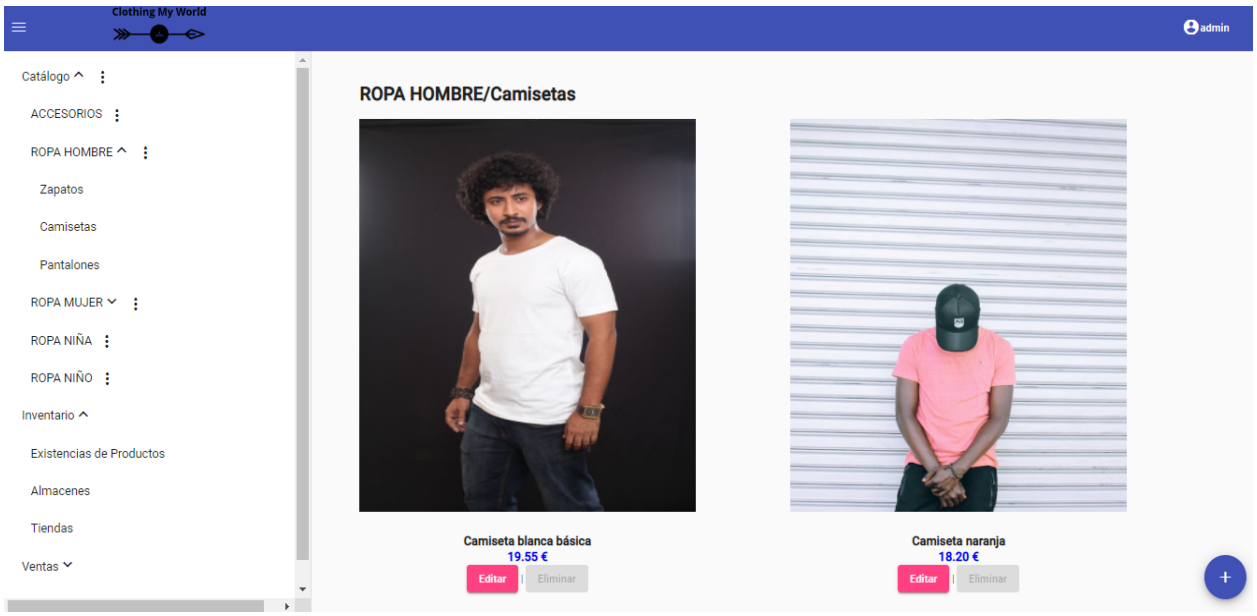

Figura A.29: Consultar subcategoría de productos.

A continuación, pulsar en el botón azul en la esquina inferior derecha, que indica *Añadir producto*, se mostrará la siguiente vista.

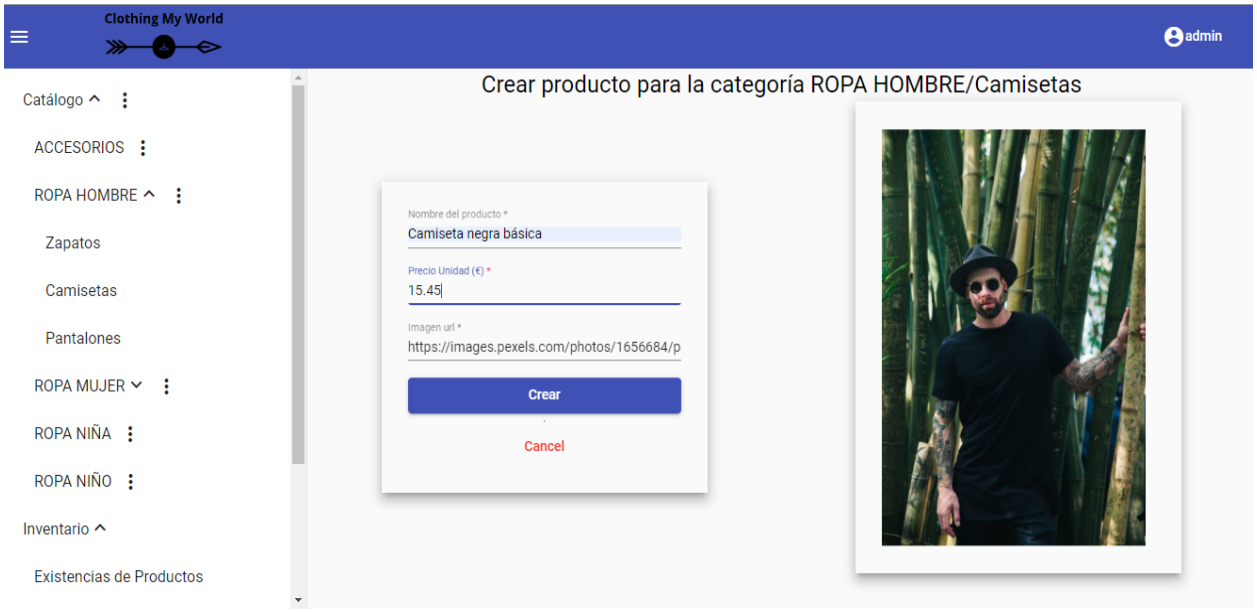

Figura A.30: Crear producto.

En el caso de que el valor de la entrada sea incorrecto o sea vacío, se indicará el error en el campo correspondiente. Al hacer clic en el botón *Crear* el producto aparecerá, en la lista de productos de la categoría.

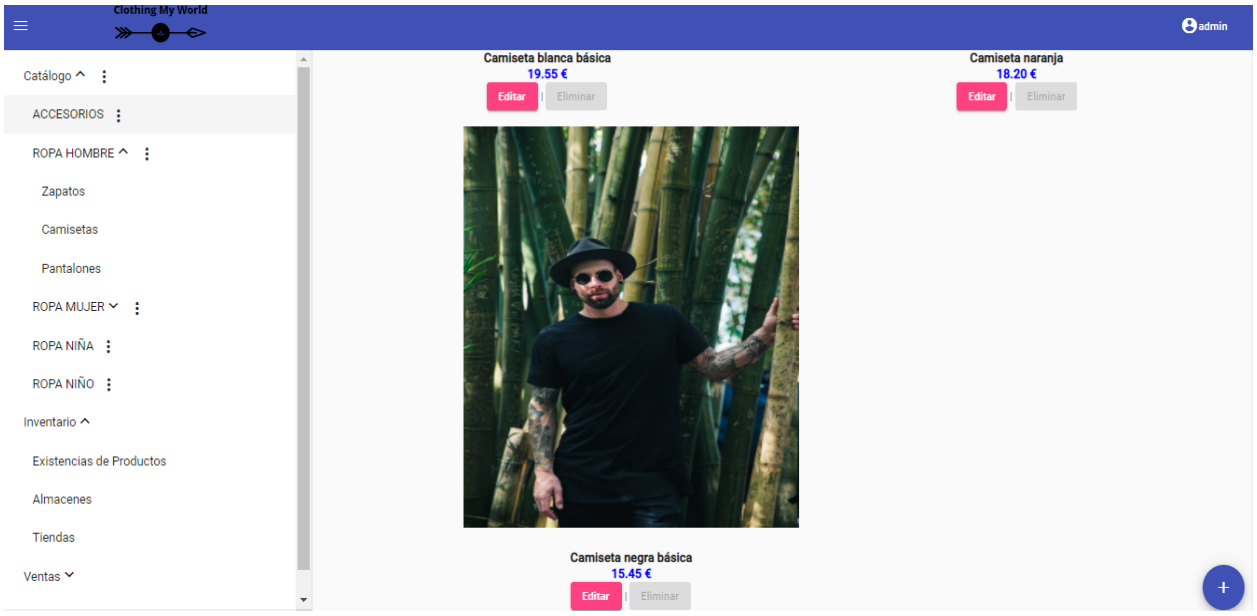

Figura A.31: Producto creado.

Si se desea modificar los atributos de un producto, hacer clic en el botón Editar debajo de la imagen de un producto y se mostrará la siguiente vista.

A.5.2. Consultar el stock de un producto en tiendas y almacenes

Pulsar en el botón *Actualizar* y los atributos del producto serán actualizados.

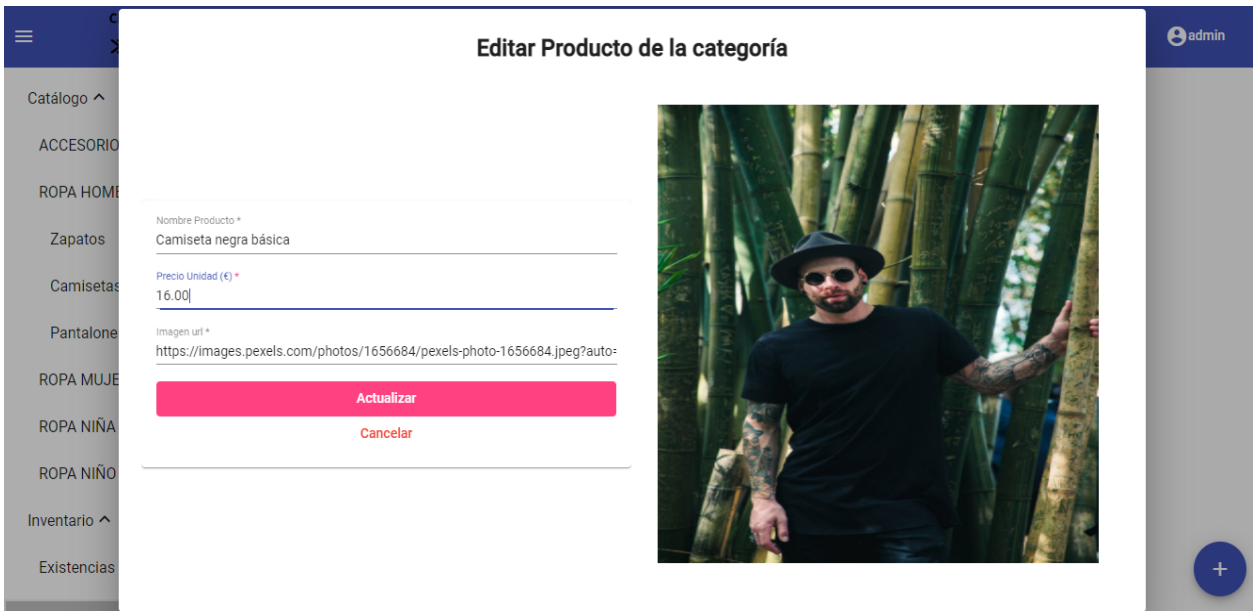

Figura A.32: Actualizar producto.

A.5.3. Asignar un producto a tiendas y almacenes

Para asignar un producto a tiendas o almacenes, lo primero que hay que hacer es hacer clic en la opción *Existencias de productos* en el menú de navegación y seleccionar un producto. Se mostrará una ventana con los almacenes y tiendas en los que está asignado.

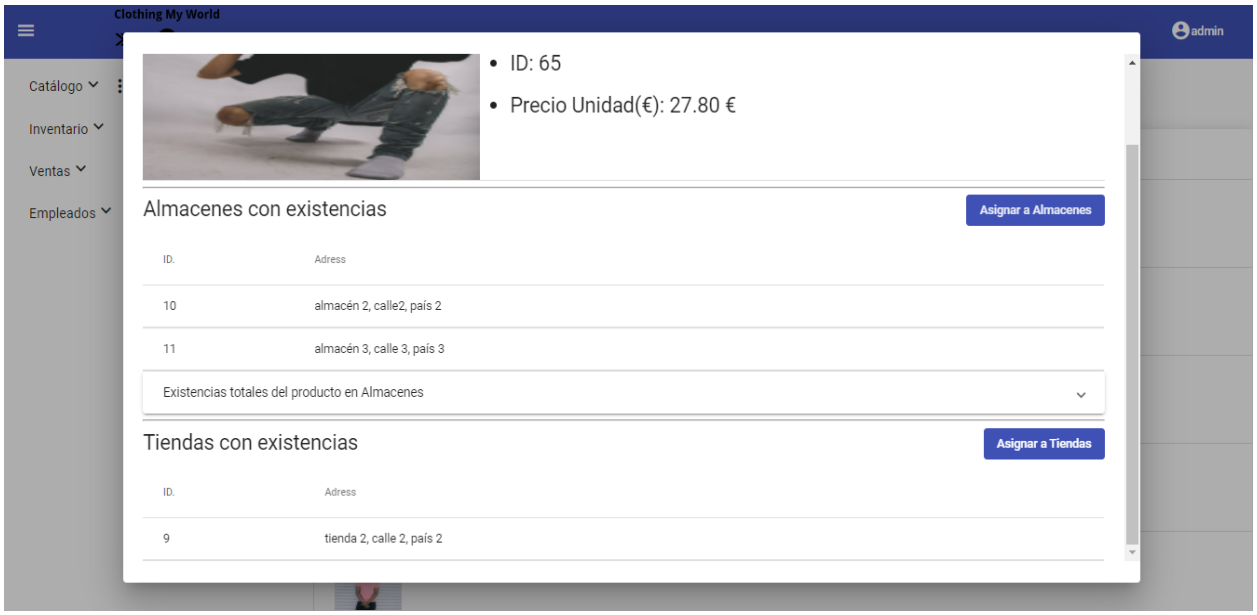

Figura A.33: Existencias del producto en tiendas y almacenes.

Al hacer clic en el botón asignar almacenes, se muestra el siguiente pop-up con las tiendas en las que no está actualmente asignado el producto.

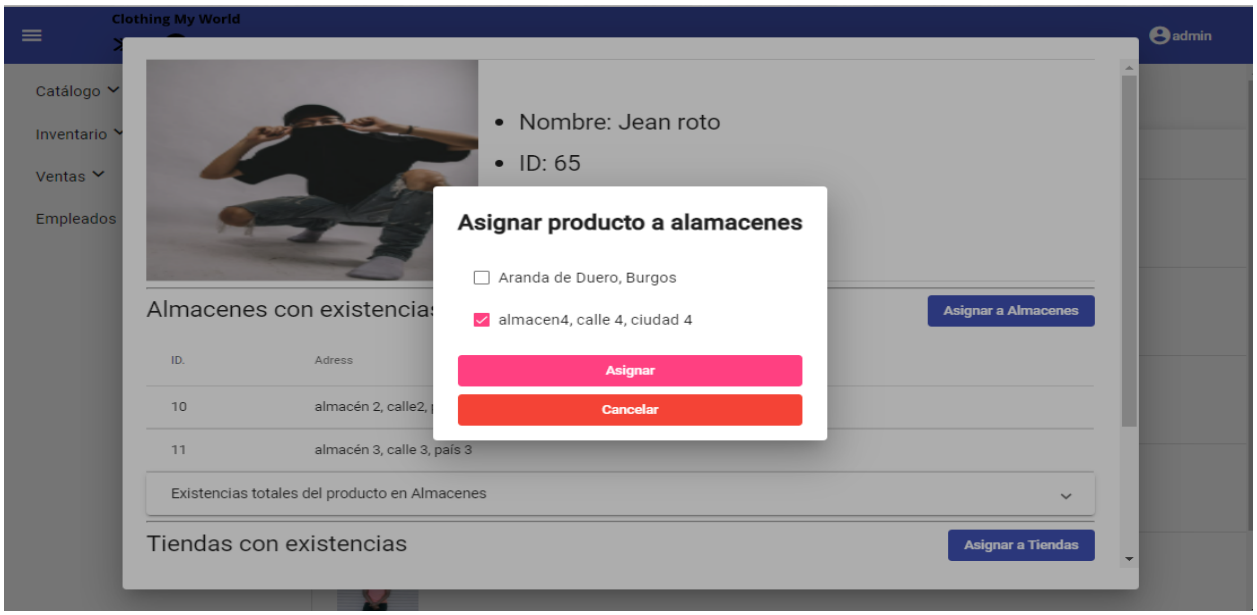

Figura A.34: Asignar producto a almacenes.

Para que se active el botón *Asignar*, es necesario que al menos haya un almacén seleccionado. Una vez se ha hecho clic en *Asignar*, el producto queda asignado a los almacenes seleccionados.

Figura A.35: Almacenes y tiendas asignados.

Para asignar el producto a tiendas, hacer clic en *Asignar a Tiendas* y seguir el mismo proceso que para almacenes.

Para ver las existencias del producto en las tiendas y almacenes asignados, hacer clic en la tiendo o almacén deseado y se mostrará el stock de cada una de sus tallas en la tienda o almacén seleccionado, Figura A.35.

A.5.4. Crear y editar tiendas o almacenes

Para crear un almacén, hacer clic en la opción *Inventario =>Almacenes* en el menú de navegación y se mostrará la siguiente vista.

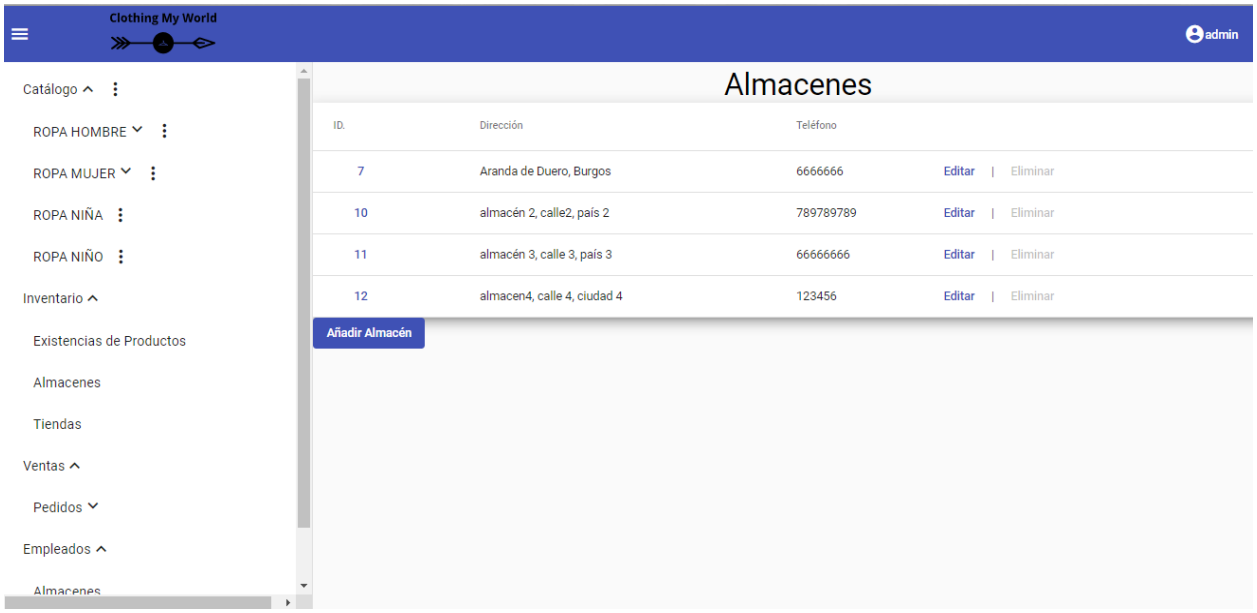

Figura A.36: Lista de almacenes.

Hacer clic en el botón *Añadir Almacén* y se mostrará la siguiente vista.

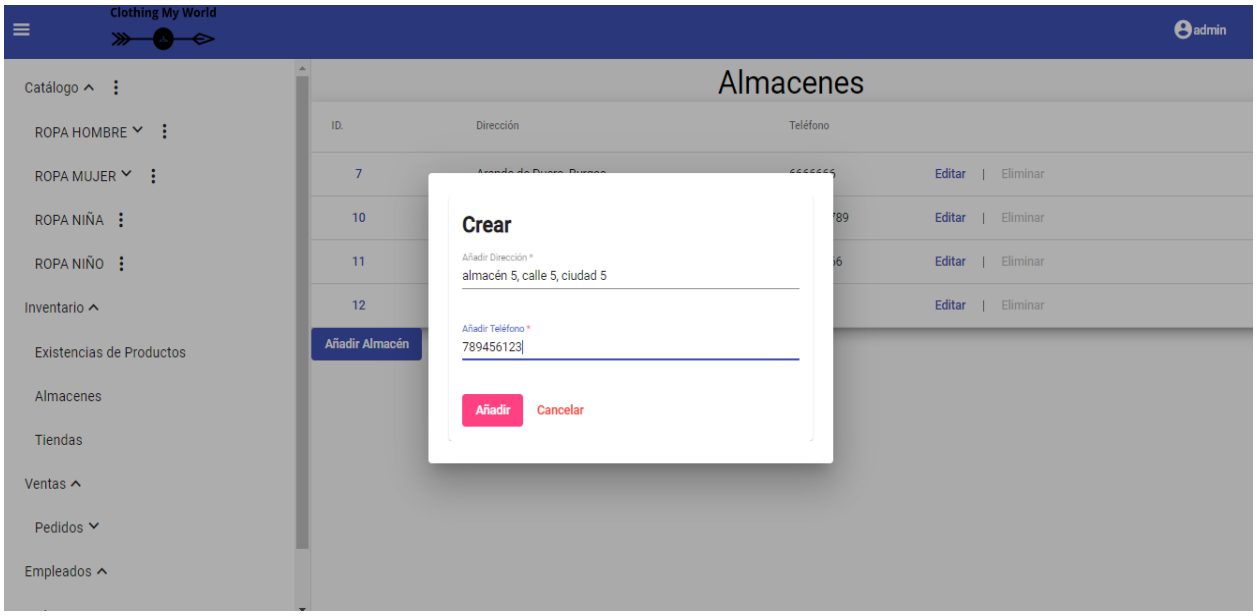

Figura A.37: Crear Almacén.

Introducir los valores de entrada, en el caso de dejar algún campo vació se indicará el error en el campo correspondiente. Hacer clic en *Añadir* y el producto aparecerá añadido a la lista de almacenes.

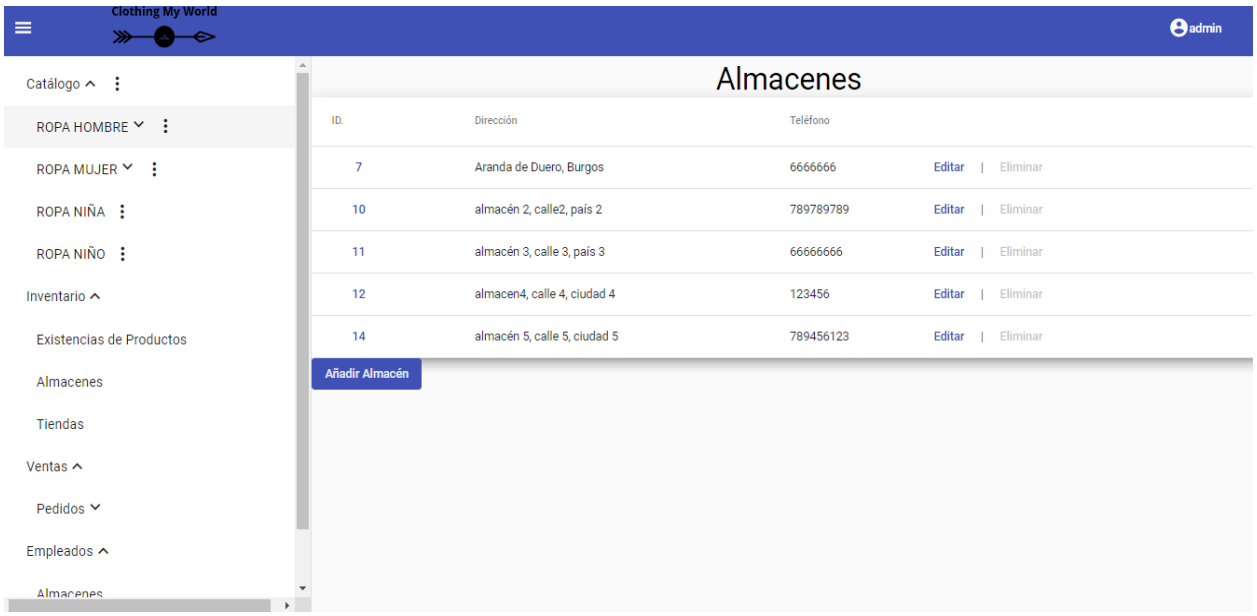

Figura A.38: Almacén creado.

Para crear una tienda, hacer clic en la opción *Inventario =>Tiendas* en el menú de navegación y continuar el proceso igual que se ha explicado para almacenes.

Para editar un almacén o una tienda, hacer clic en el botón *Editar* del almacén o tienda deseado de la lista de almacenes Figura A.38 o tiendas.

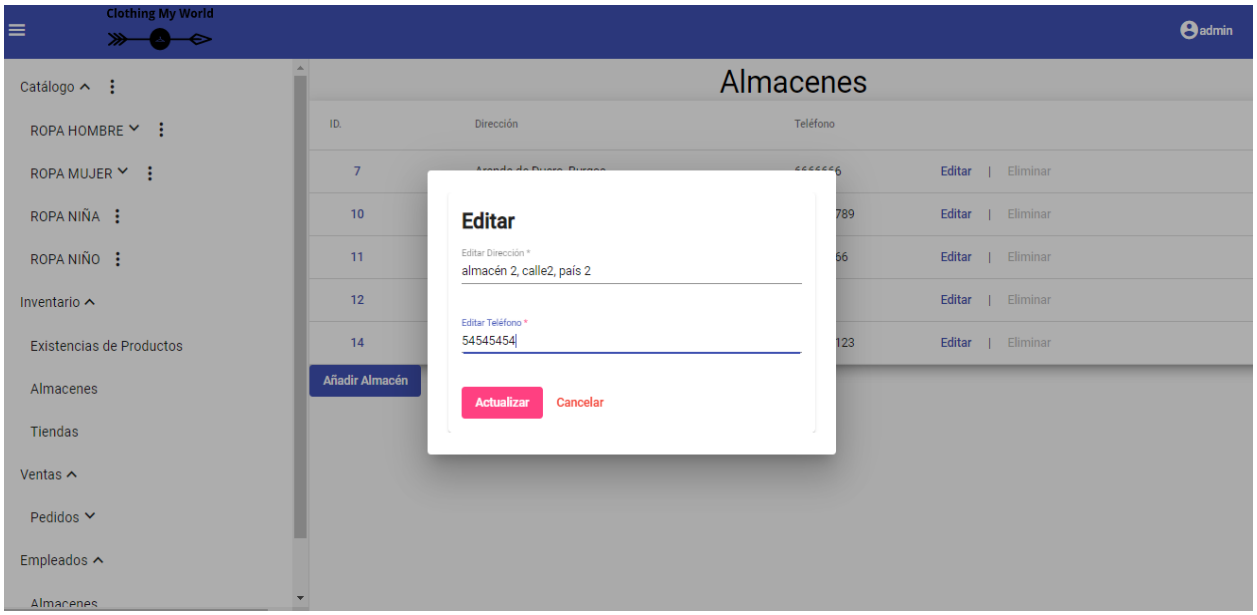

Figura A.39: Actualizar almacén.

A.5.5. Consultar el stock de una tienda o almacén

Hacer clic en el identificador de un almacén o una tienda en la lista de almacenes o tiendas, ejemplo de la lista de almacenes Figura A.38. Una vez hecho clic se mostrará el stock de la tienda o almacén seleccionado.

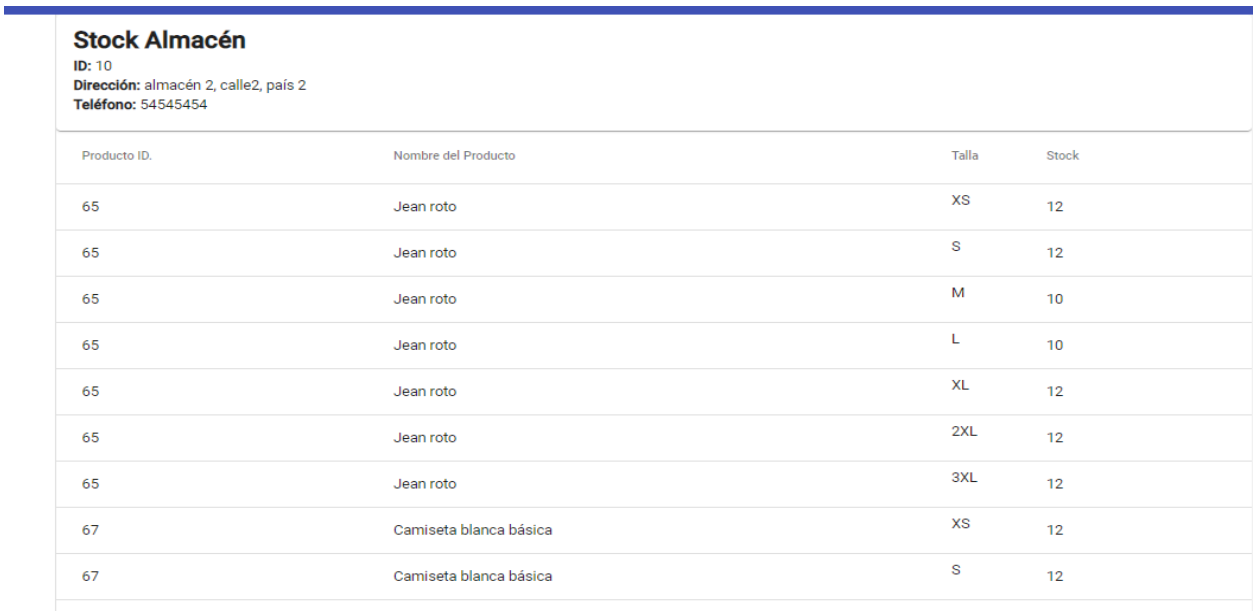

Figura A.40: Inventario de un almacén.

A.5.6. Registrar un empleado de una tienda o almacén

Para registrar el empleado de un almacén, hacer clic en *Empleados =>Almacenes* en el menú de navegación y se mostrará la lista de almacenes.

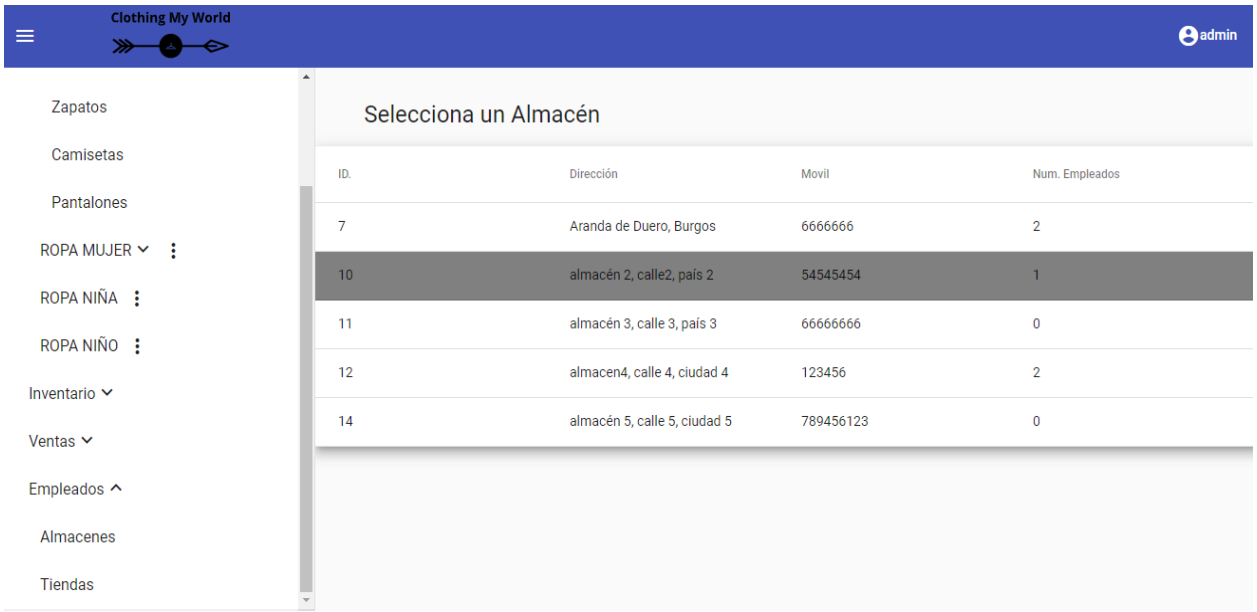

Figura A.41: Información de empleados en almacenes.

Seleccionar un almacén y se mostrará una lista con los empleados de ese almacén.

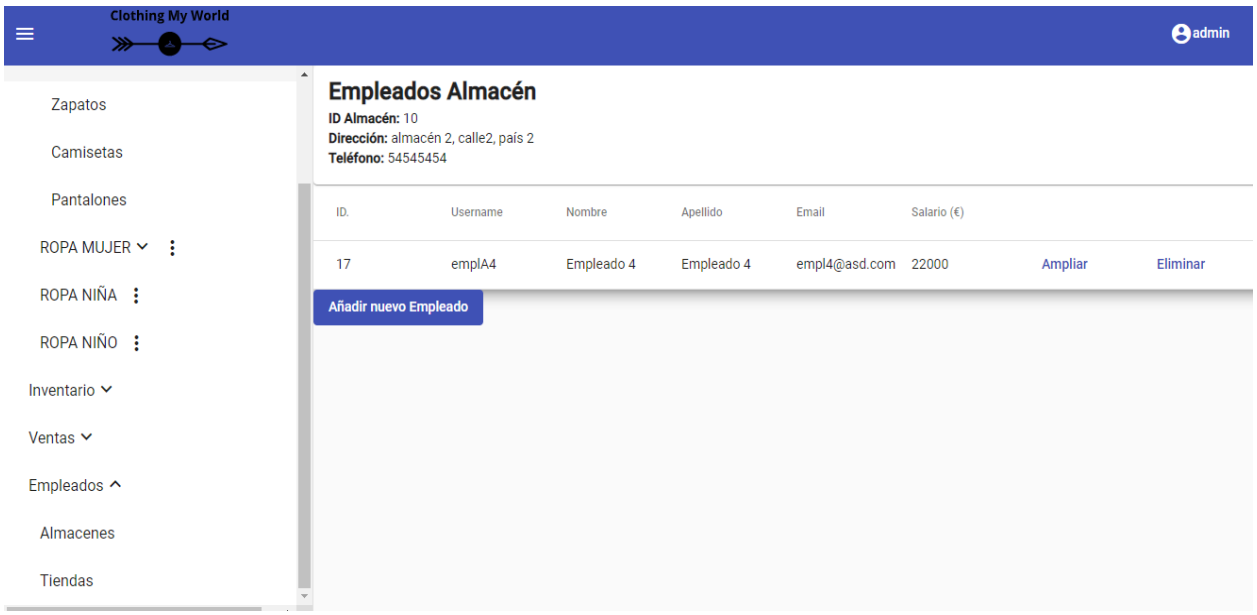

Figura A.42: Lista de empleados en almacén.

Para registrar un nuevo empleado hacer clic en *Añadir nuevo empleado* y se mostrará la siguiente vista.

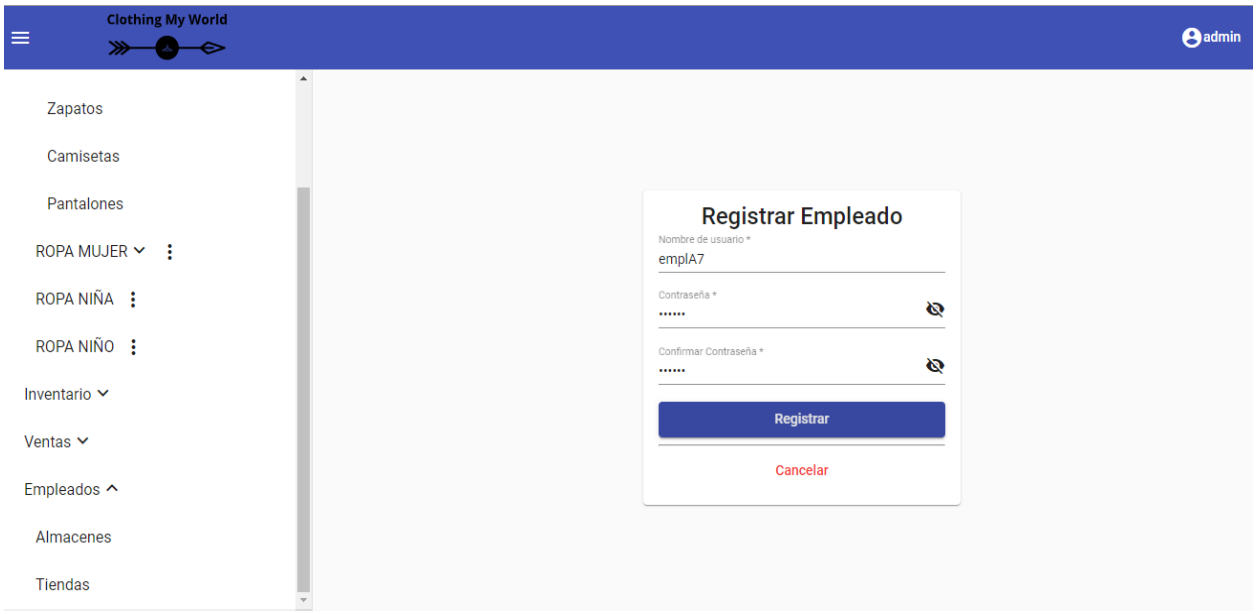

Figura A.43: Registrar empleado.

Introducir los campos de forma correcta y hacer clic en aceptar, cualquier error en la entrada será indicado.

Introducir los campos solicitados, si algún campo es nulo, el formato del email es incorrecto o el salario no es un número decimal, se indicará error en el campo correspondiente. Hacer clic en crear y el empleado creado se mostrará en la lista de empleados del almacén.

Para registrar un Empleado Tienda hacer clic en la opción *Empleados =>Tiendas* en el menú de navegación y continuar el proceso igual que se ha explicado para Empleados Almacenes.

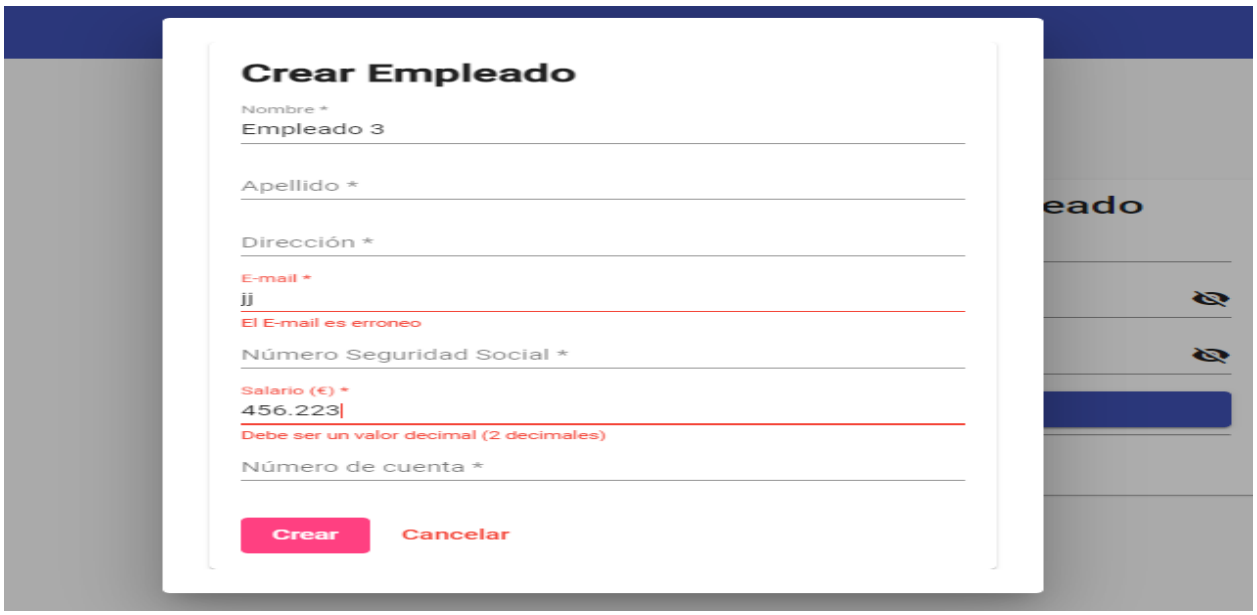

Figura A.44: Crear empleado.

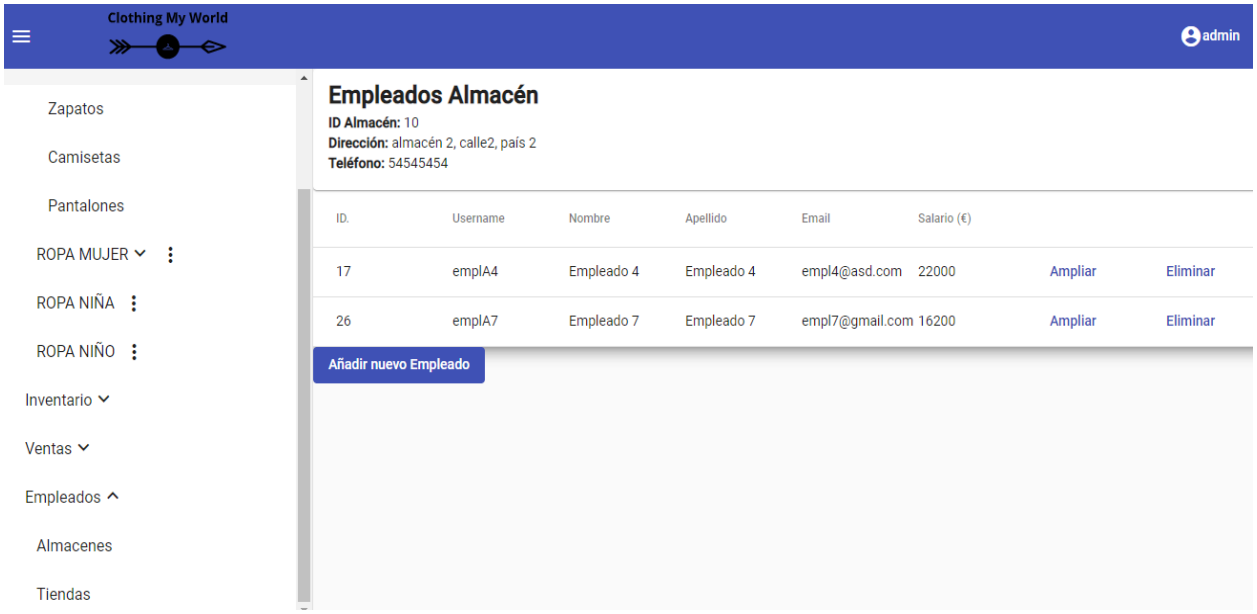

Figura A.45: Empleado almacén creado.

A.5.7. Modificar o eliminar un empleado de una tienda o almacén

Para editar o eliminar un Empleado Almacén hacer clic en el botón *Ampliar* en la lista de Empleados del almacén seleccionado Figura A.38 y se mostrará la siguiente vista.

Figura A.46: Editar empleado.

Modificar los atributos deseados y hacer clic en *Actualizar*.

Para borrar un empleado hacer clic en el botón *Eliminar* del Empleado deseado. El sistema mostrará un pop-up de confirmación.

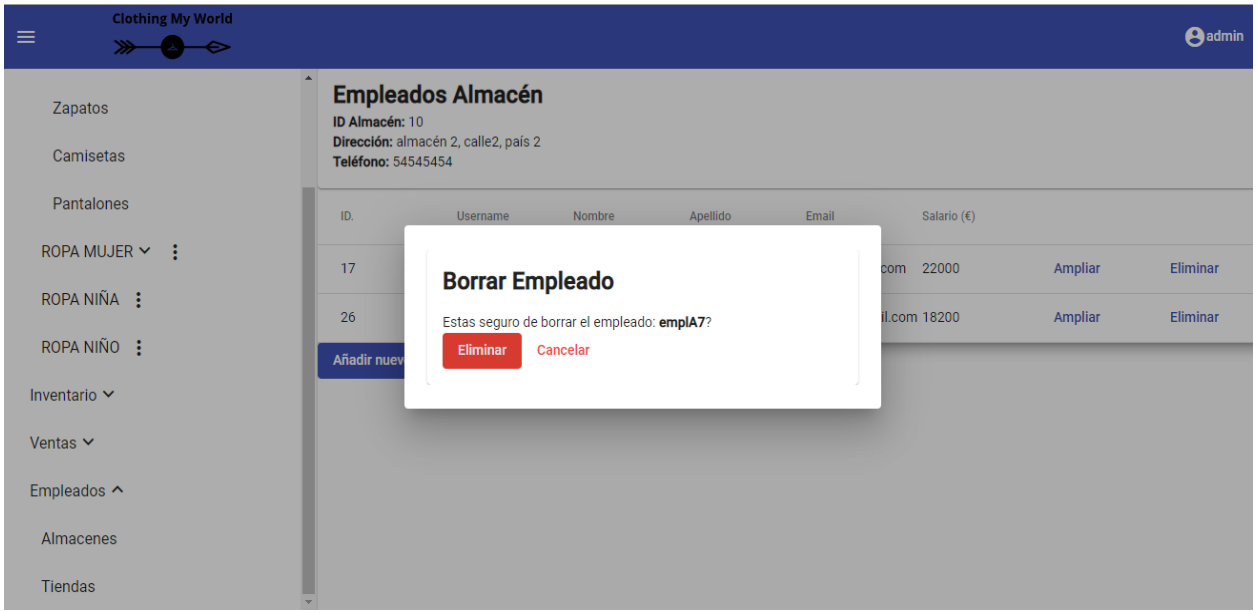

Figura A.47: Eliminar empleado.

Hacer clic en *Eliminar* y el empleado será eliminado de la lista de empleados.

Para Empleados Tienda el proceso es el mismo.

A.5.8. Consultar pedidos por estado

Para consultar los pedidos según el estado hacer clic en *Ventas =>Pedidos =>[Estado deseado]*. El sistema mostrará los pedidos de productos que se encuentran en el estado seleccionado. Ejemplo para el estado Confirmado:

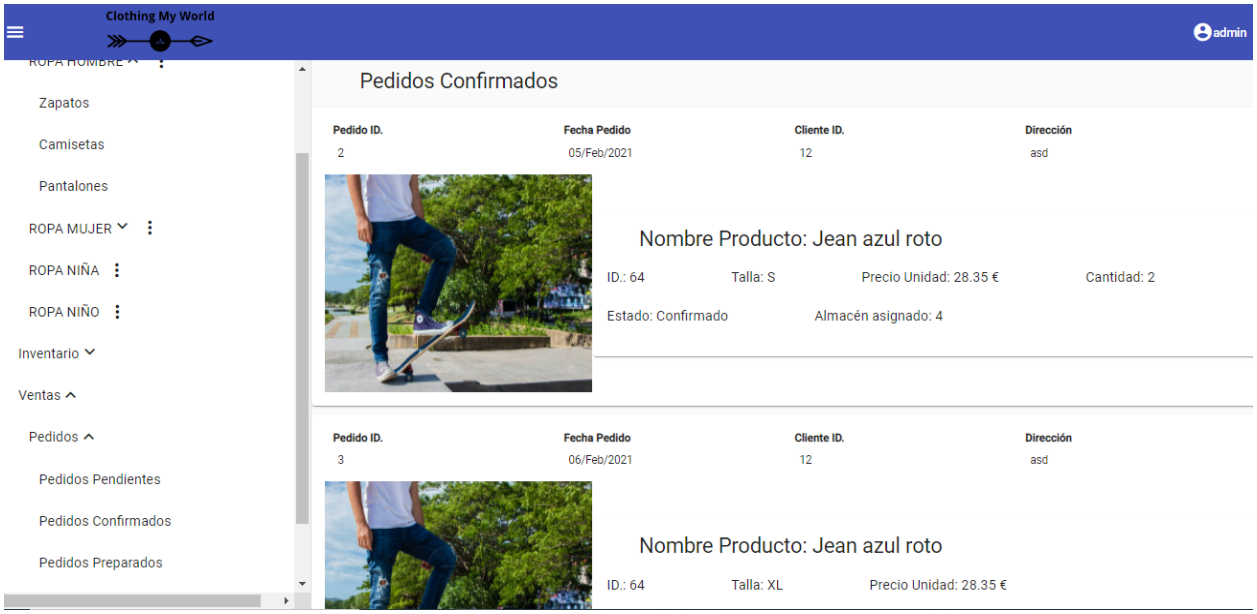

Figura A.48: Pedidos productos con estado Confirmado.