



GRADO EN INGENIERÍA
INFORMÁTICA



VNIVERSITAT
DE VALÈNCIA

TRABAJO FIN DE GRADO

DESARROLLO DE UNA TIENDA ONLINE PARA UNA
DROGUERÍA

AUTOR: CARLOS JAVIER PIQUER MAÑEZ

TUTOR: ANDRES ROCAFULL RODRIGO

MARZO 2026



UNIVERSITAT
DE VALÈNCIA



Escola Tècnica Superior
d'Enginyeria **ETSE-UV**

TRABAJO FIN DE GRADO

DESARROLLO DE UNA TIENDA ONLINE PARA UNA DROGUERÍA

AUTOR: CARLOS JAVIER PIQUER MAÑEZ

TUTOR: ANDRES ROCAFULL RODRIGO

Declaración de autoría:

Yo, Carlos Javier Piquer Mañez, declaro la autoría del Trabajo Fin de Grado titulado “Desarrollo de una tienda online para una droguería” y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual. El material no original que figura en este trabajo ha sido atribuido a sus legítimos autores.

Valencia, 27 de octubre de 2025

Fdo: Carlos Javier Piquer Mañez

Resumen:

Este es el resumen del TFM. Debe ser corto (máximo media página) y cubrir los aspectos principales del TFM.

Abstract:

This is the abstract of the TFM. It must be short and cover the main aspects of the TFM.

Resum:

Aquest és el resum del TFM. Ha de ser curt (màxim mitja pàgina) i cobrir els aspectes principals del TFM.

Agradecimientos:

En primer lugar quiero agradecer a todos aquellos que me han apoyado durante todos estos años.

En segundo lugar...

Índice general

1. Introducción	17
1.1. Introducción	17
1.2. Motivación	17
1.3. Objetivos	17
1.4. Organización de la memoria	17
2. Estado del arte	19
2.1. Análisis de aplicaciones similares	19
2.1.1. Wix eCommerce	19
2.1.2. Squarespace	20
2.1.3. PrestaShop	20
2.1.4. Comparación con la propuesta del TFG	21
2.2. Tecnologías	21
2.2.1. Frameworks frontend	21
2.2.2. Frameworks backend	23
2.2.3. Agentes de automatización	25
2.2.4. Bases de datos	27
2.2.5. Pasarelas de pago	28
3. Requisitos, especificaciones, coste, riesgos, viabilidad	31
3.1. Requisitos	31
3.1.1. Requisitos funcionales	31
3.1.2. Requisitos no funcionales	32
3.2. Especificaciones	32
3.3. Tareas y costes	33
3.3.1. Definición de tareas	33
3.3.2. Estimación temporal	33
3.3.3. Diagrama de Gantt	35
3.3.4. Estimación de costes	37

3.4. Riesgos	38
3.5. Viabilidad	39
3.5.1. Viabilidad legal	39
3.5.2. Viabilidad técnica	39
3.5.3. Viabilidad económica	39
4. Análisis	41
4.1. Diagramas de análisis	41
4.1.1. Diagrama de casos de uso	41
4.1.2. Diagrama de clases de análisis	42
4.1.3. Diagrama de secuencia general del sistema (DSGS)	43
5. Diseño	45
5.1. Arquitectura general del sistema	45
5.2. Diagramas UML	45
5.2.1. Diagrama de clases	46
5.2.2. Diagramas de secuencia	47
5.3. Diseño de la base de datos	53
5.3.1. Modelo entidad–relación	53
5.3.2. Modelo relacional y prototipo inicial	53
6. Implementación y pruebas	55
6.1. Implementación	55
6.2. Pruebas funcionales	55
6.3. Pruebas de rendimiento	55
6.4. Pruebas de usabilidad	55
7. Conclusiones	57
7.1. Revisión de costes	57
7.2. Conclusiones	57
7.3. Trabajo futuro	57
A. Apéndice	59
A.1. Ejemplos del lenguaje de marcado Latex	59
Bibliografía	60

Capítulo 1

Introducción

1.1. Introducción

1.2. Motivación

1.3. Objetivos

1.4. Organización de la memoria

Capítulo 2

Estado del arte

2.1. Análisis de aplicaciones similares

En el mercado actual la gran variedad de plataformas que nos ofrecen servicios para crear nuestra propia tienda online sin necesidad de tener algún tipo de conocimiento sobre programar es muy extenso. Algunas de las más representativas son:

2.1.1. Wix eCommerce

Wix proporciona un sistema de creación de páginas basado en el uso de herramientas de arrastrar y soltar, lo que facilita a cualquier usuario construir una tienda en poco tiempo [1]. Sin embargo, esta simplicidad también conlleva limitaciones, especialmente en lo que respecta a la personalización avanzada y al control sobre el backend, aspectos que resultan esenciales en proyectos con requisitos específicos.

Ventajas:

- Interfaz intuitiva y sistema de diseño drag-and-drop.
- Plantillas atractivas y actualizadas.
- Incluye hosting y mantenimiento automático.
- Integración sencilla con herramientas externas.

Desventajas:

- Escasa flexibilidad para personalizaciones complejas.
- Limitaciones en el acceso al código fuente.
- Dificultades para escalar proyectos grandes.
- Dependencia total del ecosistema de Wix.

2.1.2. Squarespace

Squarespace está orientado a usuarios que priorizan el diseño visual y la facilidad de uso [2]. Sus plantillas resultan intuitivas y permiten crear páginas atractivas sin dificultad. No obstante, su ecosistema de integraciones es más limitado, lo que restringe su aplicabilidad en proyectos que requieren funcionalidades técnicas más avanzadas.

Ventajas:

- Excelente diseño visual y acabados profesionales.
- Plataforma todo en uno (hosting, dominio y soporte).
- Adaptación automática a dispositivos móviles.
- Seguridad gestionada por la propia plataforma.

Desventajas:

- Pocas integraciones con APIs o sistemas externos.
- Costes mensuales relativamente altos.
- Escasa libertad para modificar la estructura del sitio.
- Dependencia del entorno cerrado de Squarespace.

2.1.3. PrestaShop

PrestaShop es una solución de código abierto muy popular para la creación de tiendas online [3]. Ofrece una gran capacidad de personalización mediante módulos y plantillas, lo que la convierte en una opción atractiva para pequeñas y medianas empresas. No obstante, requiere ciertos conocimientos técnicos para su instalación y mantenimiento, y en proyectos de mayor tamaño puede presentar dificultades de rendimiento si no se configura correctamente.

Ventajas:

- Open source y altamente personalizable.
- Gran comunidad de desarrolladores.
- Amplio catálogo de módulos y temas.
- Permite control total sobre el servidor y la base de datos.

Desventajas:

- Necesita conocimientos técnicos para instalación y soporte.
- Requiere mantenimiento y actualizaciones manuales.
- Puede presentar problemas de rendimiento en tiendas grandes.
- Algunos módulos son de pago y elevan los costes.

2.1.4. Comparación con la propuesta del TFG

Las plataformas analizadas muestran distintas formas de crear una tienda online sin necesidad de conocimientos de programación. Wix y Squarespace destacan por su sencillez y diseño intuitivo, aunque sacrifican flexibilidad y control sobre la aplicación. Por su parte, PrestaShop ofrece mayor capacidad de personalización, pero exige un nivel técnico más elevado y puede presentar limitaciones de rendimiento en proyectos de gran escala. En contraste, la propuesta de este TFG plantea el desarrollo de una solución a medida utilizando tecnologías modernas como Angular, Spring Boot, SQL y n8n, lo que permitirá un control total sobre la arquitectura, la seguridad y las integraciones externas.

Criterio	Wix	Squarespace	PrestaShop
Facilidad de uso	Muy alta	Alta	Media
Personalización	Baja	Media	Alta
Escalabilidad	Limitada	Media	Alta
Requiere conocimientos técnicos	No	No	Sí
Coste	Medio	Medio	Variable

Cuadro 2.1: Comparativa de plataformas de creación de tiendas online

2.2. Tecnologías

2.2.1. Frameworks frontend

Angular

Angular es un framework desarrollado por Google y basado en TypeScript [4]. Ofrece una solución completa para el desarrollo de aplicaciones web a gran escala, con herramientas integradas como enrutamiento y gestión de formularios. Su principal desventaja es la necesidad de contar con conocimientos previos más avanzados, lo que incrementa la curva de aprendizaje.

Ventajas:

- Framework completo con soporte oficial de Google.
- Tipado fuerte con TypeScript, que mejora la robustez del código.
- Soporte integrado para testing, formularios y enrutamiento.
- Comunidad amplia y documentación exhaustiva.

Desventajas:

- Curva de aprendizaje pronunciada.
- Tamaño inicial del proyecto elevado.
- Requiere experiencia previa en TypeScript.
- Sintaxis más compleja frente a otras alternativas.

React

React es una tecnología desarrollada por Meta [5]. Se trata de una biblioteca para la construcción de interfaces de usuario, que destaca por su flexibilidad y por el rendimiento que ofrece gracias al uso del Virtual DOM. Sin embargo, al no ser un framework completo, depende de librerías de la comunidad para cubrir funcionalidades adicionales o más avanzadas.

Ventajas:

- Excelente rendimiento gracias al Virtual DOM.
- Gran comunidad y recursos de aprendizaje.
- Compatible con múltiples librerías y frameworks.
- Curva de aprendizaje accesible para principiantes.

Desventajas:

- Necesidad de integrar herramientas externas (routing, estado).
- Menor estructura definida que Angular.
- Frecuentes actualizaciones y cambios en librerías.
- Requiere experiencia para mantener proyectos grandes.

Vue

Vue es un framework progresivo creado por Evan You, exingeniero de Google [6]. Combina características de Angular y React, con un enfoque en la simplicidad y en la rapidez de aprendizaje. No obstante, su comunidad y ecosistema siguen siendo más reducidos en comparación con otras tecnologías consolidadas.

Ventajas:

- Ligero y de fácil adopción.
- Documentación clara y organizada.
- Buena integración con proyectos existentes.
- Sintaxis sencilla y curva de aprendizaje rápida.

Desventajas:

- Ecosistema más limitado que Angular o React.
- Menor presencia en grandes empresas.
- Menos recursos de formación avanzada.
- Riesgo de menor soporte a largo plazo.

Comparación y elección

Los tres frameworks permiten crear aplicaciones modernas, pero Angular ofrece una estructura más completa desde el inicio. Para este TFG se ha optado por Angular, ya que es la tecnología utilizada en Capgemini, empresa en la que voy a realizar las prácticas. De esta forma, el proyecto servirá también como una preparación práctica, complementada con cursos de formación, para no comenzar esa experiencia sin conocimientos previos.

Criterio	Angular	React	Vue
Lenguaje principal	TypeScript	JavaScript	JavaScript
Complejidad inicial	Alta	Media	Baja
Ecosistema	Muy amplio	Amplio	Moderado
Soporte empresarial	Alto	Alto	Medio
Rendimiento general	Alto	Muy alto	Alto

Cuadro 2.2: Comparativa de frameworks frontend

2.2.2. Frameworks backend

Spring Boot

Spring Boot es un framework del ecosistema Java que facilita la creación de aplicaciones backend robustas y escalables [7]. Entre sus principales ventajas se encuentran su amplia comunidad, la integración con múltiples librerías y la facilidad para configurar proyectos complejos. Como inconvenientes, requiere conocimientos previos en Java y, según el tamaño de la aplicación, puede resultar más pesado que otras alternativas.

Ventajas:

- Framework maduro con gran soporte empresarial.
- Integración nativa con bases de datos y APIs REST.
- Facilita pruebas y despliegues automáticos.
- Gran estabilidad y seguridad.

Desventajas:

- Requiere conocimientos avanzados en Java.
- Consumo de recursos más alto que Node.js.
- Curva de aprendizaje larga.
- Configuraciones iniciales complejas.

Node.js

Node.js es un entorno de ejecución de JavaScript que permite utilizar este lenguaje en la configuración y desarrollo de servidores [8]. Destaca por su rendimiento en aplicaciones

en tiempo real y por la ventaja de compartir el mismo lenguaje en frontend y backend, lo que agiliza el desarrollo. Su principal limitación es que no está tan orientado a aplicaciones de gran escala como otros frameworks más consolidados.

Ventajas:

- Excelente rendimiento en tiempo real.
- Usa el mismo lenguaje en cliente y servidor.
- Amplia comunidad y soporte.
- Ecosistema extenso con NPM.

Desventajas:

- Menor rendimiento en tareas muy pesadas.
- Dificultad para proyectos altamente estructurados.
- Depende mucho de librerías externas.
- Escalabilidad más limitada que Spring Boot.

Django

Django es un framework de Python que sigue el principio "batteries included", lo que significa que incorpora numerosas funcionalidades listas para usar [9]. Es adecuado para el desarrollo rápido de prototipos y proyectos de tamaño medio, con una curva de aprendizaje moderada. Sin embargo, su rendimiento puede verse limitado en aplicaciones muy exigentes.

Ventajas:

- Gran cantidad de herramientas integradas.
- Excelente para desarrollo rápido.
- Seguridad y control de autenticación robusto.
- Documentación muy completa.

Desventajas:

- Menor rendimiento en aplicaciones grandes.
- Menor integración con sistemas Java.
- No tan extendido en entornos empresariales.
- Difícil de optimizar para proyectos complejos.

Comparación y elección

Spring Boot, Node.js y Django son alternativas viables para el desarrollo de un backend. Node.js destaca por su rapidez en tiempo real y Django por la simplicidad de Python, pero para este TFG se ha optado por Spring Boot. Esta elección responde a que es la tecnología utilizada en Capgemini, empresa en la que realizaré las prácticas, lo que me permite formarme previamente mediante cursos y adquirir los conceptos básicos necesarios. Además, Spring Boot destaca por su robustez, su madurez en entornos empresariales y porque complementa de forma óptima la arquitectura planteada con Angular y SQL.

Criterio	Spring Boot	Node.js	Django
Lenguaje base	Java	JavaScript	Python
Escalabilidad	Alta	Media	Media
Curva de aprendizaje	Alta	Media	Baja
Rendimiento general	Alto	Alto	Medio
Soporte empresarial	Muy alto	Medio	Medio

Cuadro 2.3: Comparativa de frameworks backend

2.2.3. Agentes de automatización

Zapier

Zapier es una herramienta para automatizar tareas mediante la conexión de aplicaciones. Destaca por su simplicidad y facilidad de uso, aunque sus funcionalidades avanzadas requieren una suscripción de pago[10].

Ventajas:

- Muy fácil de usar.
- Gran catálogo de aplicaciones compatibles.
- Ideal para tareas simples y rápidas.
- Interfaz visual clara y moderna.

Desventajas:

- Las funciones avanzadas requieren suscripción.
- Limitado control sobre la configuración interna.
- Escasa flexibilidad para integraciones complejas.
- No permite autoalojamiento.

Make

Make, anteriormente conocido como Integromat, es una herramienta que ofrece un sistema visual para crear flujos de trabajo más complejos que en Zapier. Es potente y flexible, aunque su uso intensivo también depende de planes de pago y de la integración con servicios externos[11].

Ventajas:

- Muy flexible para automatizaciones avanzadas.
- Interfaz visual potente y configurable.
- Permite múltiples pasos por flujo.
- Buen equilibrio entre facilidad y personalización.

Desventajas:

- Algunas funciones requieren plan de pago.
- Requiere conexión constante a la nube.
- Mayor complejidad de uso frente a Zapier.
- Menor documentación en español.

n8n

n8n es una plataforma de automatización de código abierto que permite crear flujos personalizados e instalarlos en servidores propios, sin necesidad de planes de pago. Ofrece un gran control sobre la configuración y resulta una herramienta intuitiva y fácil de utilizar [12].

Ventajas:

- Totalmente open source y gratuita.
- Puede instalarse en servidores propios.
- Gran control sobre la configuración.
- Compatible con múltiples servicios y APIs.

Desventajas:

- Requiere configuración inicial manual.
- Menor comunidad que Zapier.
- No tan intuitiva para principiantes.
- Necesita recursos del servidor donde se aloje.

Comparación y elección

Zapier y Make destacan por su facilidad de uso y popularidad, pero dependen de planes de suscripción y de servicios externos, además de no ofrecer un control total sobre la configuración. En este TFG se ha optado por n8n por ser una solución open source, gratuita y altamente configurable, lo que facilita su integración con la arquitectura propuesta.

Criterio	Zapier	Make	n8n
Tipo de licencia	De pago	Freemium	Open source
Facilidad de uso	Muy alta	Alta	Media
Control y personalización	Bajo	Medio	Alto
Coste	Elevado	Medio	Gratis
Soporte técnico	Alto	Medio	Medio

Cuadro 2.4: Comparativa de agentes de automatización

2.2.4. Bases de datos

SQL

Las bases de datos relacionales, como MySQL o PostgreSQL, utilizan un modelo basado en tablas y relaciones entre ellas [13]. Destacan por su consistencia, su alto nivel de seguridad y por el uso del lenguaje SQL, que es un estándar ampliamente adoptado en el entorno empresarial.

NoSQL

Las bases de datos NoSQL, como MongoDB, almacenan la información en estructuras más flexibles, como documentos o grafos. Ofrecen mayor escalabilidad y rapidez en ciertas operaciones, aunque sacrifican consistencia y mecanismos de seguridad más avanzados. [14].

Comparación y elección

Para este TFG se ha optado por utilizar **MySQL** como sistema de gestión de bases de datos. La elección se debe a su estabilidad, su integración nativa con Spring Boot y su capacidad para garantizar la seguridad de los datos. Además, a lo largo de la carrera se ha trabajado ampliamente con bases de datos SQL, mientras que NoSQL apenas se ha tratado, por lo que MySQL resulta una opción más familiar y adecuada para el desarrollo del proyecto.

Criterio	SQL (MySQL)	NoSQL (MongoDB)
Estructura	Tablas relacionales	Documentos / grafos
Consistencia	Alta	Media
Seguridad	Alta	Media
Escalabilidad	Media	Alta
Soporte empresarial	Muy alto	Medio

Cuadro 2.5: Comparativa entre bases de datos SQL y NoSQL

2.2.5. Pasarelas de pago

PayPal

PayPal es una de las plataformas de pago online más conocidas y utilizadas a nivel mundial [15]. Permite realizar transacciones de forma sencilla y segura, tanto con tarjeta como con saldo en cuenta. Es muy popular entre los usuarios por su rapidez y confianza, aunque cobra comisiones algo más altas que otras alternativas.

Ventajas:

- Gran confianza y reconocimiento internacional.
- Permite pagar sin necesidad de introducir datos bancarios en cada compra.
- Interfaz sencilla para el usuario.
- Compatible con muchas plataformas de comercio electrónico.

Desventajas:

- Comisiones elevadas por transacción.
- Poca flexibilidad para personalizar el flujo de pago.
- Integración técnica menos avanzada que otras opciones.
- Posibles retenciones temporales de fondos en algunas operaciones.

Redsys

Redsys es la pasarela de pago más extendida en España y se utiliza en la mayoría de tiendas online que trabajan con bancos nacionales [16]. Ofrece un alto nivel de seguridad y cumple con las normativas europeas, aunque su configuración inicial puede ser más compleja y su documentación menos accesible para desarrolladores.

Ventajas:

- Muy utilizada en España, con soporte por parte de los principales bancos.
- Alta seguridad gracias al uso de sistemas 3D Secure.
- Cumple con las normativas PSD2 y SCA.

- Fiable y con buena reputación en entornos comerciales.

Desventajas:

- Configuración inicial compleja.
- Documentación técnica limitada.
- Poco orientada a desarrolladores.
- Limitada flexibilidad en integraciones personalizadas.

Stripe

Stripe es una pasarela de pago moderna pensada especialmente para desarrolladores [17]. Su API es clara y bien documentada, lo que facilita mucho la integración con aplicaciones web. Permite gestionar pagos con tarjeta, transferencias y otros métodos de forma segura y rápida. Es una opción muy utilizada por startups y empresas tecnológicas por su facilidad de uso y sus amplias posibilidades de personalización.

Ventajas:

- API muy completa y fácil de integrar.
- Admite múltiples métodos de pago y divisas.
- Buen equilibrio entre seguridad y personalización.
- Excelente documentación y comunidad activa.

Desventajas:

- Requiere conocimientos técnicos básicos para la configuración.
- Comisiones similares a las de PayPal.
- Depende de una conexión estable con la API.
- No disponible en todos los países.

Comparación y elección

PayPal, Redsys y Stripe son tres opciones válidas para integrar pagos en una tienda online. PayPal destaca por su popularidad y facilidad de uso, mientras que Redsys es la alternativa más común en España y ofrece gran seguridad. Sin embargo, para este TFG se ha optado por **Stripe** debido a su flexibilidad, buena documentación y facilidad de integración con tecnologías modernas como Spring Boot y Angular. Además, su API REST simplifica la conexión con el backend y garantiza un proceso de pago seguro y fluido para el usuario.

Criterio	PayPal	Redsys	Stripe
Popularidad	Muy alta	Alta	Alta
Facilidad de integración	Media	Baja	Alta
Comisiones	Altas	Medias	Medias
Orientación a desarrolladores	Baja	Baja	Muy alta
Seguridad	Alta	Muy alta	Alta

Cuadro 2.6: Comparativa de pasarelas de pago

Capítulo 3

Requisitos, especificaciones, coste, riesgos, viabilidad

En este capítulo se presenta un estudio inicial del proyecto, analizando los requisitos, especificaciones, costes, riesgos y viabilidad del sistema desarrollado. El objetivo es definir las **funcionalidades necesarias** para el correcto funcionamiento de la aplicación, detallar sus **especificaciones técnicas**, estimar el **coste y el tiempo de desarrollo**, identificar los **principales riesgos** y, finalmente, valorar la **viabilidad global** del proyecto. A continuación, se detallan los distintos apartados que conforman este estudio.

3.1. Requisitos

El sistema propuesto consiste en una **plataforma web de comercio electrónico para una droguería** que permite a los usuarios registrarse, explorar productos, filtrar resultados, gestionar su carrito de compra, realizar pagos mediante un servicio externo (Stripe) y recibir notificaciones automáticas mediante el sistema n8n. Por parte del administrador, el sistema debe permitir filtrar pedidos, añadir productos, así como consultar y modificar los pedidos existentes.

A continuación, se detallan los requisitos funcionales y no funcionales definidos para el proyecto.

3.1.1. Requisitos funcionales

- **RF-01:** El sistema debe permitir el registro y autenticación de usuarios mediante correo electrónico y contraseña.
- **RF-02:** El sistema debe permitir que los usuarios consulten y filtren el catálogo de productos disponible.
- **RF-03:** El sistema debe permitir añadir, modificar o eliminar productos del carrito.
- **RF-04:** El sistema debe permitir realizar pedidos a través de una pasarela de pago (Stripe).
- **RF-05:** El sistema debe permitir al administrador gestionar el inventario de productos (crear, editar, eliminar).

- **RF-06:** El sistema debe generar y almacenar facturas de los pedidos realizados.
- **RF-07:** El sistema debe enviar notificaciones automáticas de confirmación de pedido y aviso de bajo stock mediante n8n.
- **RF-08:** El sistema debe permitir consultar el historial de pedidos y su estado.
- **RF-09:** El sistema debe eliminar automáticamente los carritos inactivos tras un periodo determinado.

3.1.2. Requisitos no funcionales

- **RNF-01:** El sistema debe desarrollarse siguiendo una arquitectura cliente-servidor (Angular + Spring Boot + MySQL).
- **RNF-02:** La interfaz debe ser intuitiva, responsive y accesible desde distintos dispositivos.
- **RNF-03:** El sistema debe garantizar la seguridad de los datos mediante cifrado de contraseñas y uso de HTTPS.
- **RNF-04:** La base de datos debe permitir transacciones seguras y mantener la integridad referencial.
- **RNF-05:** El código debe estar estructurado siguiendo buenas prácticas de desarrollo (modularización, uso de control de versiones).
- **RNF-06:** El sistema debe permitir su despliegue local o en la nube sin necesidad de licencias adicionales.
- **RNF-07:** Las automatizaciones mediante n8n deben ser ejecutadas en un entorno controlado y seguro.

3.2. Especificaciones

El sistema está diseñado como una **aplicación web moderna** basada en una arquitectura **frontend-backend**, en la que el cliente (Angular) se comunica con el servidor (Spring Boot) mediante *API REST*. La base de datos utilizada es **MySQL**, encargada de almacenar toda la información relativa a usuarios, productos, pedidos, pagos y facturas. Además, el sistema incorpora **n8n** como herramienta de automatización para gestionar tareas recurrentes, como el envío de correos postcompra o las notificaciones de bajo stock.

El sistema distingue entre dos tipos principales de usuarios:

- **Cliente:** puede registrarse, gestionar su carrito, realizar compras y consultar su historial de pedidos.
- **Administrador:** gestiona el catálogo de productos, controla los pedidos y supervisa las automatizaciones.

La aplicación está desarrollada íntegramente en **español**, es **multiplataforma** y accesible desde cualquier navegador moderno. Todos los componentes empleados son **gratuitos y de código abierto**, lo que facilita su instalación y mantenimiento sin coste adicional.

3.3. Tareas y costes

Para planificar el desarrollo del proyecto se ha elaborado una **Estructura de Descomposición del Trabajo (EDT)** que organiza las actividades en cinco fases principales: estudio, análisis, diseño, implementación y pruebas. Esta estructura permite identificar las dependencias entre tareas y establecer una secuencia lógica de ejecución.

3.3.1. Definición de tareas

ID	Nombre de la tarea	Dependencias
1	Estudio y especificación del proyecto	—
1.1	Definición de requisitos	—
1.2	Estimación de costes y planificación	1.1
1.3	Análisis de riesgos	1.2
1.4	Estudio de herramientas (Angular, Spring Boot, n8n)	1.3
2	Análisis del sistema	1
2.1	Casos de uso de la aplicación	1.4
2.2	Modelo de datos y entidades principales	2.1
3	Diseño del sistema	2
3.1	Arquitectura general (frontend-backend-n8n)	2.2
3.2	Diagramas UML y diseño de base de datos	3.1
4	Implementación	3
4.1	Desarrollo del backend en Spring Boot	3.2
4.2	Desarrollo del frontend en Angular	4.1
4.3	Integración con Stripe y MySQL	4.2
4.4	Configuración de flujos n8n	4.3
5	Pruebas y validación	4
5.1	Pruebas unitarias e integración	4.4
5.2	Validación con usuarios y optimización final	5.1

Cuadro 3.1: Estructura de Descomposición del Trabajo (EDT) y dependencias entre tareas.

3.3.2. Estimación temporal

Para estimar la duración de cada fase se ha utilizado la técnica **PERT (Program Evaluation and Review Technique)**, que permite obtener una estimación temporal basada en tres valores: el tiempo optimista, el más probable y el pesimista. Esta metodología se fundamenta en los principios clásicos de estimación temporal descritos en la gestión de proyectos [18] y calcula el tiempo estimado (TE) mediante la fórmula:

$$TE = \frac{O + 4M + P}{6}$$

donde O es el tiempo optimista, M el más probable y P el pesimista.

Fase	O	M	P	TE (días)
Estudio y especificación	3	5	7	5
Análisis	4	6	8	6
Diseño	6	8	10	8
Implementación	20	25	30	25
Pruebas	6	8	10	8
Total estimado				52 días

Cuadro 3.2: Estimación temporal de las fases del proyecto mediante el método PERT.

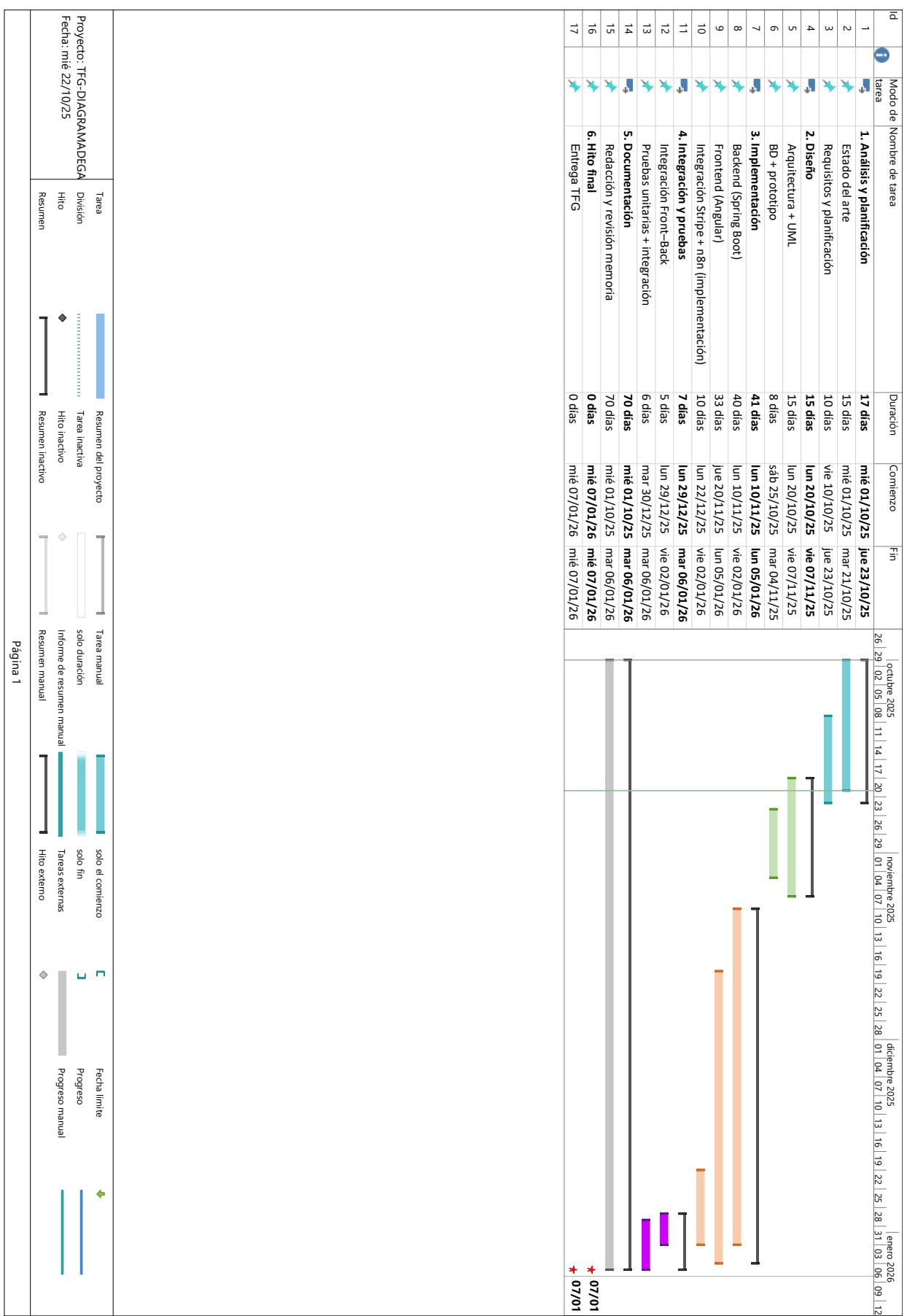
Además de la técnica **PERT**, se ha aplicado el método de **juicio de expertos** con el objetivo de validar las estimaciones obtenidas y asegurar que los tiempos calculados sean realistas. Para ello, se han considerado las opiniones de tres perfiles con diferentes niveles de experiencia en el ámbito del desarrollo de software, lo que permitió ajustar las duraciones de las tareas y obtener una planificación más equilibrada:

- **Perfil 1 — Estudiante avanzado:** estudiante de último curso del Grado en Ingeniería Informática, con experiencia en proyectos académicos y conocimientos de programación web. Estimó que el proyecto podría completarse en torno a unos **55 días**, dedicando más tiempo a las fases de diseño e implementación, al considerar que estas suponen un mayor reto técnico en un proyecto individual.
- **Perfil 2 — Desarrollador junior:** profesional con unos dos años de experiencia en el desarrollo de aplicaciones web. Según su criterio, la duración total estaría alrededor de **50 días**, al poder abordar las fases de análisis e implementación con mayor agilidad, aunque manteniendo márgenes prudentes en las fases de pruebas y documentación.
- **Perfil 3 — Desarrollador senior/jefe de proyecto:** profesional con más de cinco años de experiencia en la gestión y supervisión de proyectos informáticos. Consideró que el proyecto podría completarse en torno a **48 días**, optimizando las fases de diseño e implementación gracias a una mejor planificación y experiencia previa en proyectos similares. Su punto de vista permitió validar la coherencia general del calendario y la estimación global del esfuerzo.

El contraste de opiniones entre estos perfiles permitió confirmar que las estimaciones iniciales eran adecuadas, situándose todas en un rango comprendido entre los 48 y 55 días, con una media aproximada de **50 días**. A partir de sus aportaciones, se fijó una duración final de **52 días**, incorporando un pequeño margen de contingencia que refleja un equilibrio entre las distintas perspectivas.

3.3.3. Diagrama de Gantt

El siguiente diagrama de Gantt muestra la planificación temporal del proyecto de manera visual, indicando la duración de cada fase y su relación con las tareas definidas en la estructura de descomposición del trabajo (EDT). Se ha generado a partir de las estimaciones realizadas mediante la técnica PERT y validadas con el juicio de expertos.



3.3.4. Estimación de costes

Para el cálculo de costes se han tenido en cuenta tanto los recursos humanos como los materiales y el software utilizado. Aunque el proyecto ha sido desarrollado por una sola persona, se ha desglosado el trabajo según distintos perfiles habituales en el desarrollo de software: jefe de proyecto, analista, desarrollador y tester. De esta forma se obtiene una estimación más ajustada y realista del esfuerzo requerido.

Concepto	Detalle	Coste estimado
Coste de personal	Jefe de proyecto / analista: $40 \text{ h} \times 18 \text{ €/h}$ Desarrollador backend: $140 \text{ h} \times 12 \text{ €/h}$ Desarrollador frontend: $100 \text{ h} \times 12 \text{ €/h}$ Tester / validación: $40 \text{ h} \times 10 \text{ €/h}$	720 € 1.680 € 1.200 € 400 €
Subtotal personal		4.000 €
Coste de software	Herramientas gratuitas (Angular, Spring Boot, MySQL, n8n, VSCode, Postman)	0 €
Coste de software con licencia universitaria	Visual Paradigm (licencia académica proporcionada por la universidad)	0 €
Coste de hardware (amortizado)	Portátil personal (1.200 € de valor, amortizado 4 meses sobre 4 años)	100 €
Total estimado del proyecto		4.100 €

Cuadro 3.3: Estimación de costes detallada por perfil y recursos.

Nota: el valor del hardware se ha amortizado en base a una vida útil estimada de cuatro años.

Para la planificación temporal se ha aplicado la técnica **PERT**, combinada con el **juicio de expertos**, comparando los valores calculados con estimaciones de proyectos similares y con referencias salariales obtenidas de portales como Indeed. Además, se consultaron tres perfiles con diferente nivel de experiencia en el desarrollo de software, lo que permitió contrastar las horas estimadas y ajustar los tiempos de las tareas.

El resultado confirma que el proyecto mantiene un coste razonable y acorde a su nivel de complejidad. El uso de herramientas libres y de una infraestructura propia reduce significativamente el coste total, manteniendo una buena relación entre esfuerzo, recursos y resultados obtenidos.

3.4. Riesgos

La gestión de riesgos permite anticipar posibles incidencias y definir estrategias preventivas que minimicen su impacto en el desarrollo del proyecto. En la siguiente tabla se resumen los principales riesgos identificados junto con su probabilidad, impacto y categoría general.

Riesgo identificado	Probabilidad	Impacto	Categoría
Problemas de integración entre módulos Angular–Spring Boot	Media	Alta	Técnica
Fallos en la pasarela de pago (Stripe)	Baja	Alta	Técnica
Cambios en los requisitos durante el desarrollo	Media	Media	Gestión
Sobrecarga de trabajo o retrasos en la planificación	Media	Media	Organización
Fallos de seguridad en la API o la base de datos	Baja	Alta	Seguridad
Falta de experiencia inicial con n8n	Media	Media	Técnica

Cuadro 3.4: Principales riesgos identificados durante el desarrollo del proyecto.

A continuación, se describen las medidas generales de **mitigación y contingencia** aplicadas a los riesgos identificados:

- **Problemas de integración entre módulos Angular–Spring Boot:** se intentará detectar pronto cualquier error realizando pruebas cada vez que se conecten nuevas partes del sistema. También se mantendrá una buena organización del código para facilitar la comunicación entre ambos módulos.
- **Fallos en la pasarela de pago (Stripe):** antes de activar los pagos reales, se harán pruebas en el modo de simulación que ofrece la plataforma. En caso de error, se podrá reintentar el pago o registrar el pedido para revisarlo después.
- **Cambios en los requisitos durante el desarrollo:** se dejará algo de margen en la planificación para poder adaptarse a posibles cambios sin afectar al trabajo principal. Además, se priorizarán las funciones más importantes para tener siempre una versión funcional.
- **Sobrecarga de trabajo o retrasos en la planificación:** se dividirá el trabajo en tareas pequeñas y se intentará cumplir objetivos semanales. También se reservará algo de tiempo extra para posibles imprevistos o revisiones de última hora.
- **Fallos de seguridad en la aplicación o base de datos:** se harán copias de seguridad con frecuencia y se guardarán las contraseñas de forma segura. Además, se mantendrán las herramientas actualizadas para evitar problemas.
- **Falta de experiencia con n8n:** se dedicará tiempo a aprender su funcionamiento antes de integrarlo por completo. Se probarán ejemplos sencillos para entender cómo crear los flujos y se consultará la documentación cuando sea necesario.

Gracias a estas medidas, se espera que los riesgos identificados tengan un impacto limitado en el desarrollo y no comprometan los objetivos principales del proyecto.

3.5. Viabilidad

3.5.1. Viabilidad legal

El proyecto no presenta impedimentos legales, ya que no maneja datos personales sensibles y cumple con la normativa vigente, especialmente con el Reglamento General de Protección de Datos (RGPD). Los pagos se realizan mediante la plataforma **Stripe**, que se encarga de procesar la información de forma segura y cifrada, evitando que el sistema tenga que almacenar datos de tarjetas. Todas las tecnologías empleadas son de **código abierto** o disponen de **licencias académicas gratuitas**, por lo que su uso es totalmente legal en el contexto de un proyecto universitario.

3.5.2. Viabilidad técnica

El uso conjunto de **Angular**, **Spring Boot**, **MySQL** y **n8n** proporciona una solución estable, escalable y fácil de mantener. Son herramientas actuales, bien documentadas y con una amplia comunidad de desarrolladores, lo que facilita la resolución de problemas y reduce el riesgo de incompatibilidades. Además, el diseño modular del sistema permite incorporar futuras mejoras, como nuevos métodos de pago o automatizaciones adicionales, sin necesidad de modificar la estructura principal del proyecto.

3.5.3. Viabilidad económica

El proyecto resulta **económicamente viable**, ya que su desarrollo no requiere inversión en licencias ni en infraestructura adicional. El software utilizado es **gratuito y de código abierto**, y el equipo empleado para el desarrollo es personal, por lo que los únicos costes reales corresponden al tiempo de dedicación y al uso del propio ordenador. Teniendo en cuenta que el coste total estimado del proyecto es de aproximadamente **4.400 €**, se puede concluir que se trata de una solución accesible, eficiente y sostenible para un entorno académico o de pequeña empresa.

Capítulo 4

Análisis

4.1. Diagramas de análisis

4.1.1. Diagrama de casos de uso

Para representar las interacciones entre los diferentes actores del sistema se ha elaborado el diagrama de casos de uso mostrado en la Figura 4.1. Este permite visualizar de forma general las funcionalidades principales y la relación entre el cliente, el administrador y el agente automatizador *n8n*.

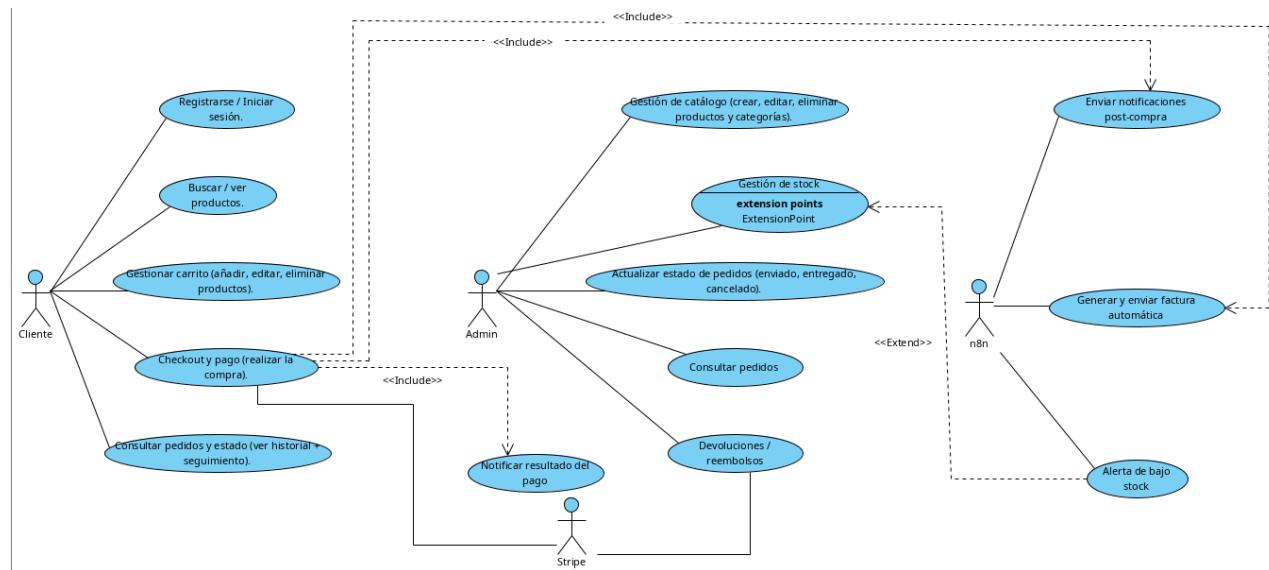


Figura 4.1: Diagrama de casos de uso del sistema.

4.1.2. Diagrama de clases de análisis

En la Figura 4.2 se muestra el **diagrama de clases de análisis** del sistema. Este modelo conceptual identifica las principales entidades que intervienen en el dominio de la aplicación, así como las relaciones que existen entre ellas. El cliente dispone de un carrito y puede realizar múltiples pedidos, cada uno de los cuales está asociado a un pago y a varios productos. Además, el cliente puede almacenar varias direcciones de envío, y los productos se agrupan dentro de categorías. El diagrama abstrae los detalles técnicos y se centra únicamente en los conceptos del negocio, sirviendo como base para el posterior diagrama de clases de diseño.

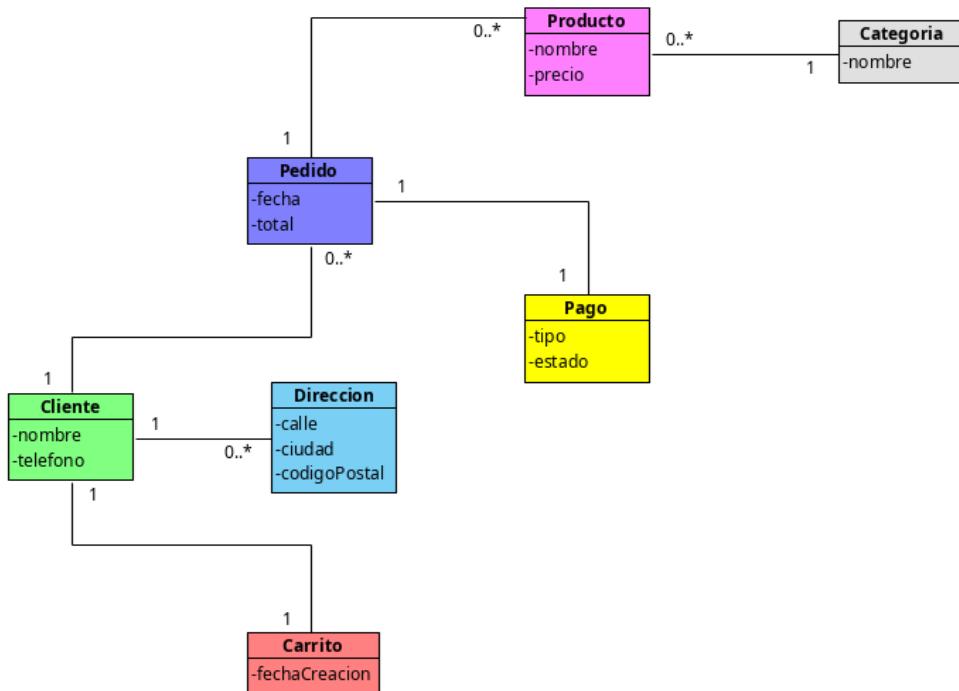


Figura 4.2: Diagrama de clases de análisis del sistema.

4.1.3. Diagrama de secuencia general del sistema (DSGS)

La Figura 4.3 muestra el **diagrama de secuencia general del sistema (DSGS)**, que representa de forma dinámica las interacciones entre los principales actores externos y la *WebApp*. En él se detalla el flujo completo de una compra, desde el inicio de sesión del cliente, la selección de productos y la realización del pedido, hasta la comunicación con la pasarela de pago *Stripe* y la notificación del resultado a la plataforma de automatización *n8n*. El diagrama también incluye un flujo alternativo en el que se muestra el manejo de un posible error de pago. Este modelo complementa al diagrama de casos de uso al ofrecer una visión temporal del comportamiento global del sistema.

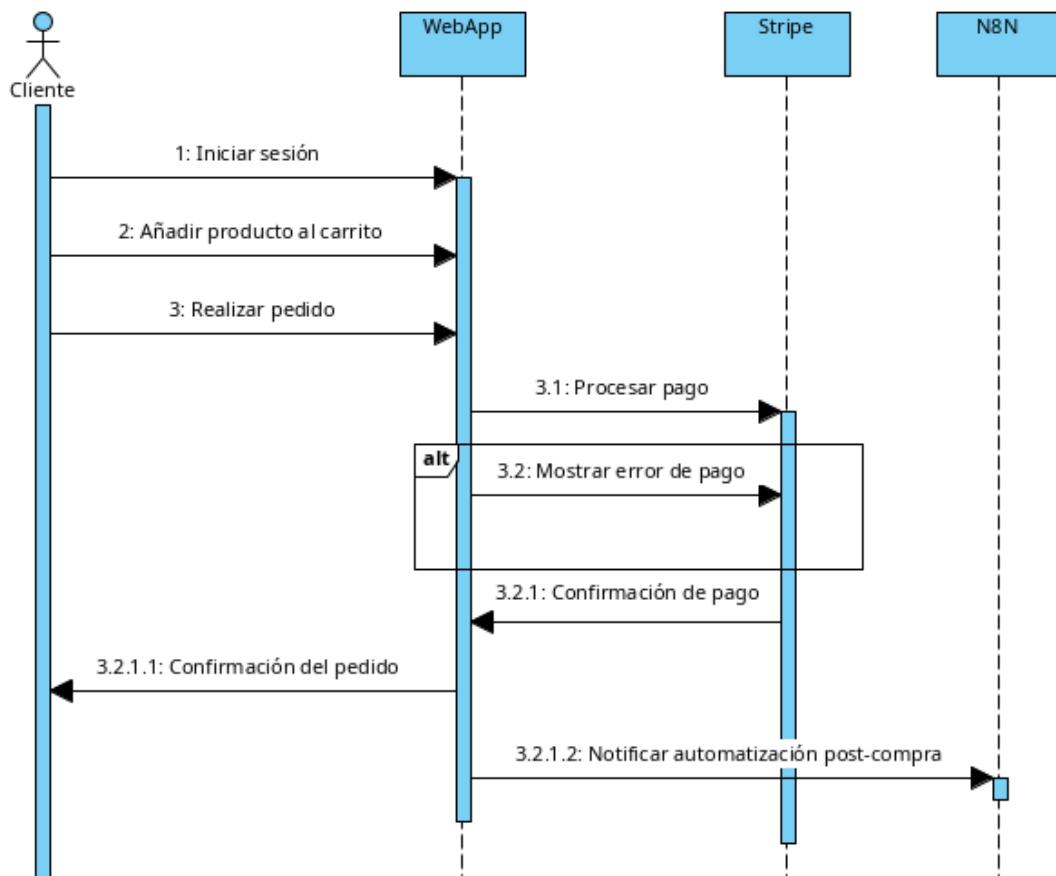


Figura 4.3: Diagrama de secuencia general del sistema (DSGS).

Capítulo 5

Diseño

En este capítulo se presenta el proceso de diseño del sistema desarrollado, que abarca tanto la definición de la arquitectura general como los distintos diagramas UML que describen la estructura y el comportamiento de la aplicación. Finalmente, se incluye el diseño de la base de datos, que servirá como soporte a la implementación posterior.

5.1. Arquitectura general del sistema

El sistema se basa en una arquitectura de tres capas que facilita la separación de responsabilidades y la escalabilidad del proyecto:

- **Frontend:** desarrollado con un framework moderno (Angular o React), encargado de la interacción con el usuario y la comunicación con el backend mediante peticiones HTTP.
- **Backend:** implementado con *Spring Boot*, que gestiona la lógica de negocio, las operaciones CRUD y la conexión con la base de datos.
- **Base de datos:** gestionada con el motor *MySQL*, donde se almacenan las entidades principales del sistema.

Además, el sistema integra un agente automatizador *n8n* encargado de tareas como el envío de notificaciones post-compra, la generación automática de facturas o la detección de carritos abandonados. Esta arquitectura modular permite una comunicación fluida entre componentes y facilita la futura ampliación del sistema.

5.2. Diagramas UML

Los diagramas UML se han elaborado con la herramienta *Visual Paradigm* para representar de forma visual los componentes y sus interacciones. Se incluyen diagramas de casos de uso, clases y secuencia.

5.2.1. Diagrama de clases

En la Figura 5.1 se muestra el diagrama de clases que define la estructura interna del sistema, las entidades principales y sus relaciones. Entre las clases más destacadas se encuentran:

- **Usuario:** almacena la información del cliente o administrador.
 - **Producto:** contiene los datos de los artículos disponibles.
 - **Carrito y CarritoItems:** gestionan los productos añadidos por cada usuario.
 - **Pedido, Pago y Factura:** representan el flujo de compra y facturación.

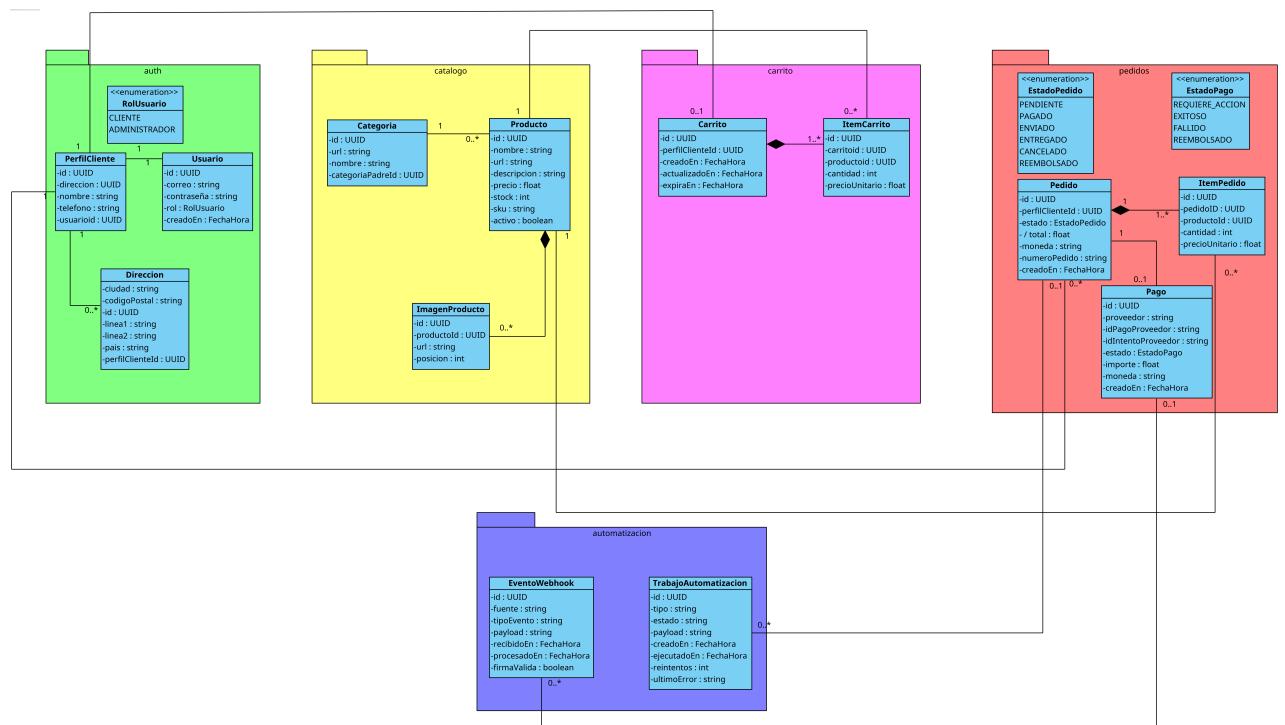


Figura 5.1: Diagrama de clases del sistema

5.2.2. Diagramas de secuencia

Los diagramas de secuencia reflejan los flujos de interacción entre los distintos componentes del sistema en diferentes escenarios funcionales. Estos esquemas permiten entender cómo se comunican los distintos servicios y agentes en cada proceso clave del sistema.

Gestión de carrito En este diagrama se representa el flujo de interacción entre el cliente, la aplicación web y los servicios del catálogo y del carrito. El usuario puede añadir, modificar o eliminar productos del carrito, y el sistema actualiza los datos en tiempo real mostrando la información correspondiente a cada producto.

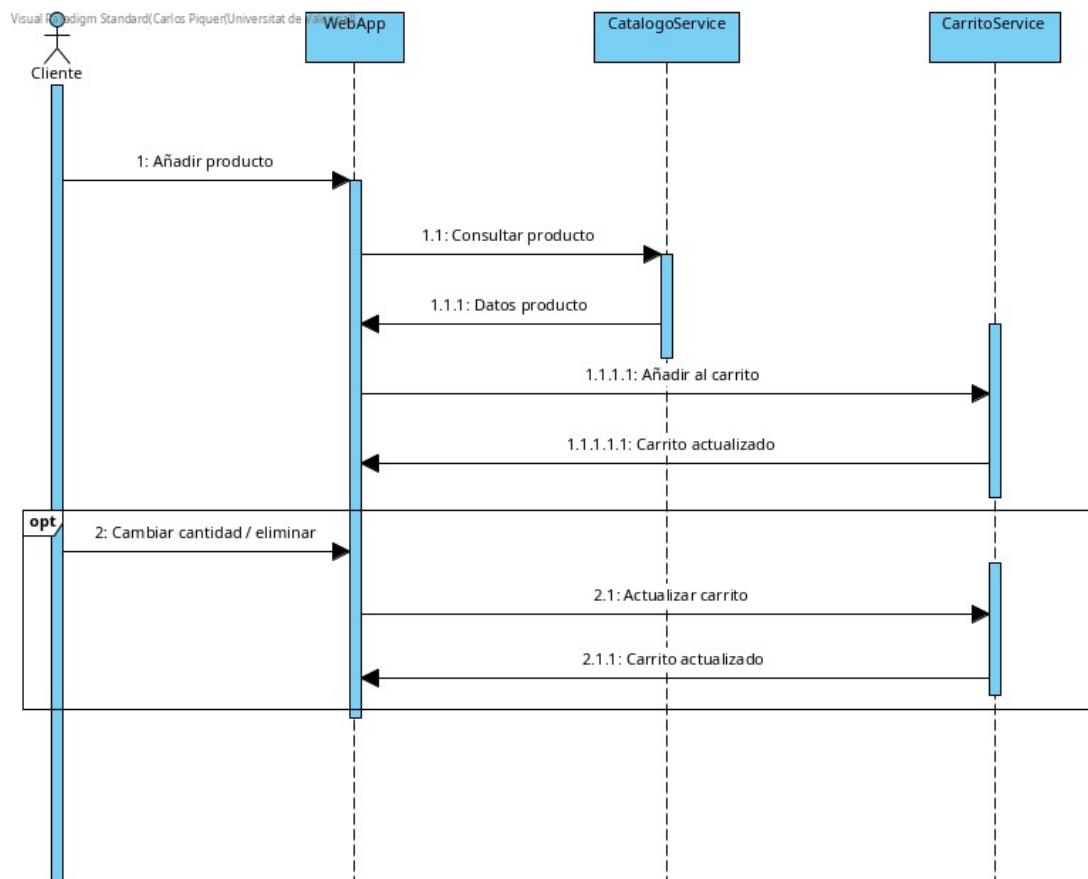


Figura 5.2: Diagrama de secuencia — gestión del carrito

Compra estándar A continuación, se muestra el flujo completo del proceso de compra. Incluye la autenticación del usuario, la creación del checkout, la comunicación con el servicio de pago *Stripe* y la confirmación del pedido. También se representa la respuesta automatizada del agente *n8n* para confirmar la compra, descontar el stock y generar la factura.

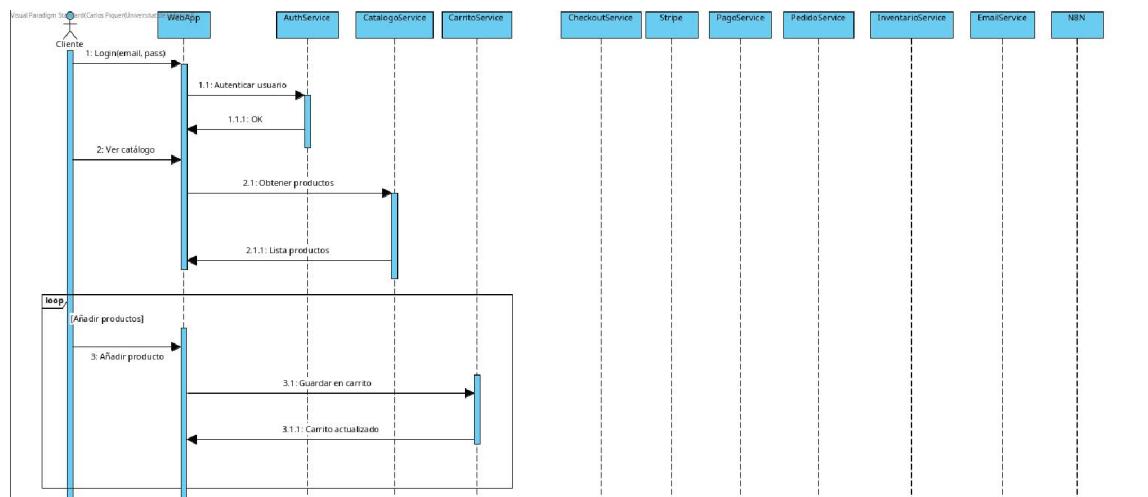


Figura 5.3: Diagrama de secuencia — parte 1: autenticación y carrito

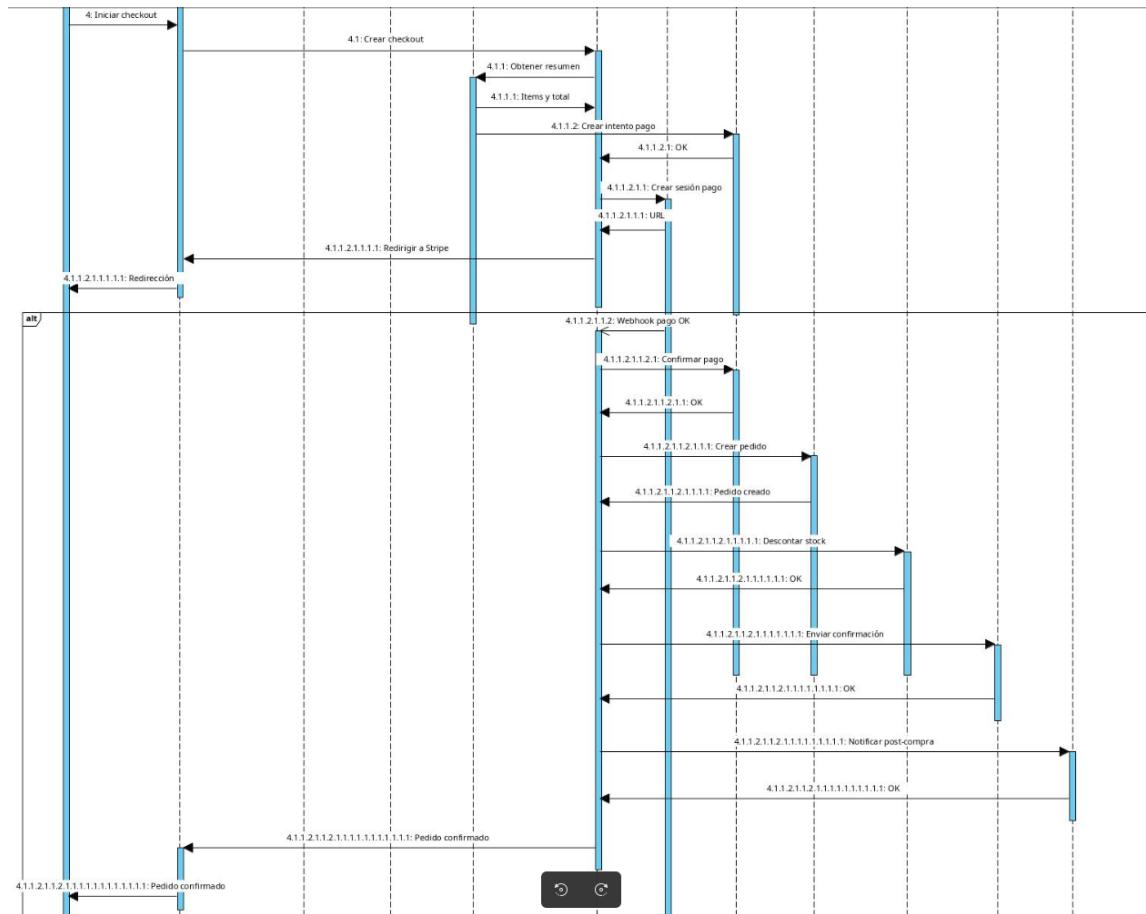


Figura 5.4: Diagrama de secuencia — parte 2: proceso de checkout y confirmación

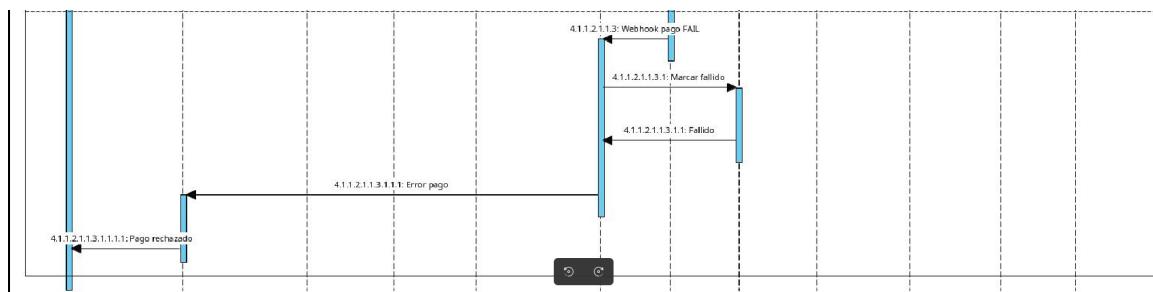


Figura 5.5: Diagrama de secuencia — parte 3: flujo alternativo de pago fallido

Generación de factura Este diagrama muestra el proceso automatizado de creación de facturas tras la confirmación de un pedido. Una vez validado el pago, el sistema genera el documento asociado y lo envía al cliente mediante el agente *n8n*, garantizando así un flujo de trabajo totalmente automatizado.

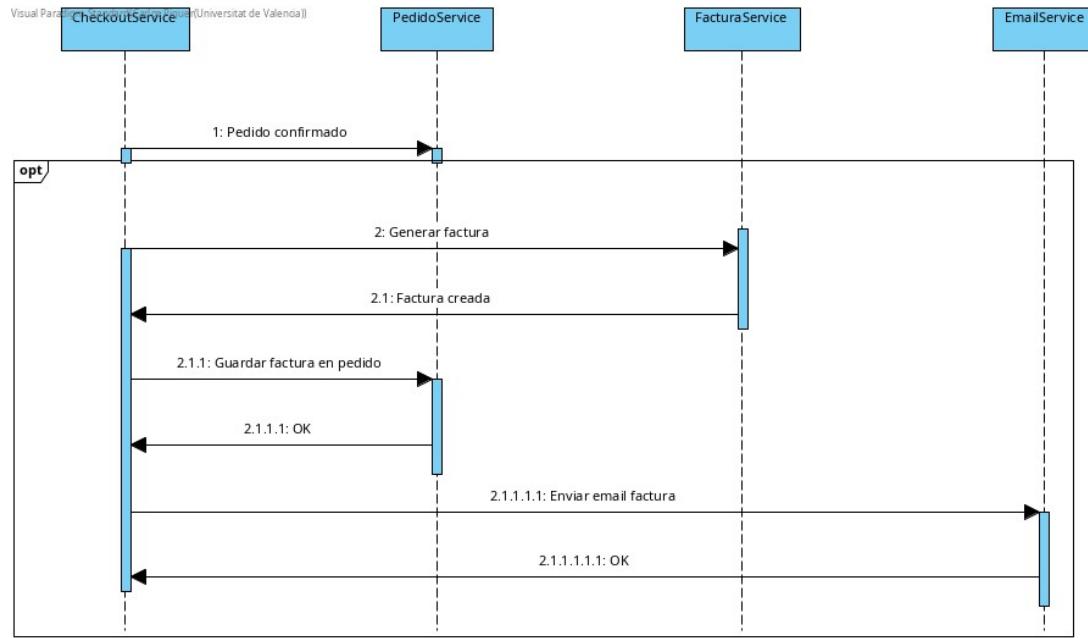


Figura 5.6: Diagrama de secuencia — generación de factura

Abandono de carrito Este diagrama refleja el funcionamiento del proceso de detección y eliminación de carritos inactivos. Si un usuario añade productos pero no completa la compra en un tiempo determinado, el sistema marca el carrito como abandonado y libera los productos reservados.

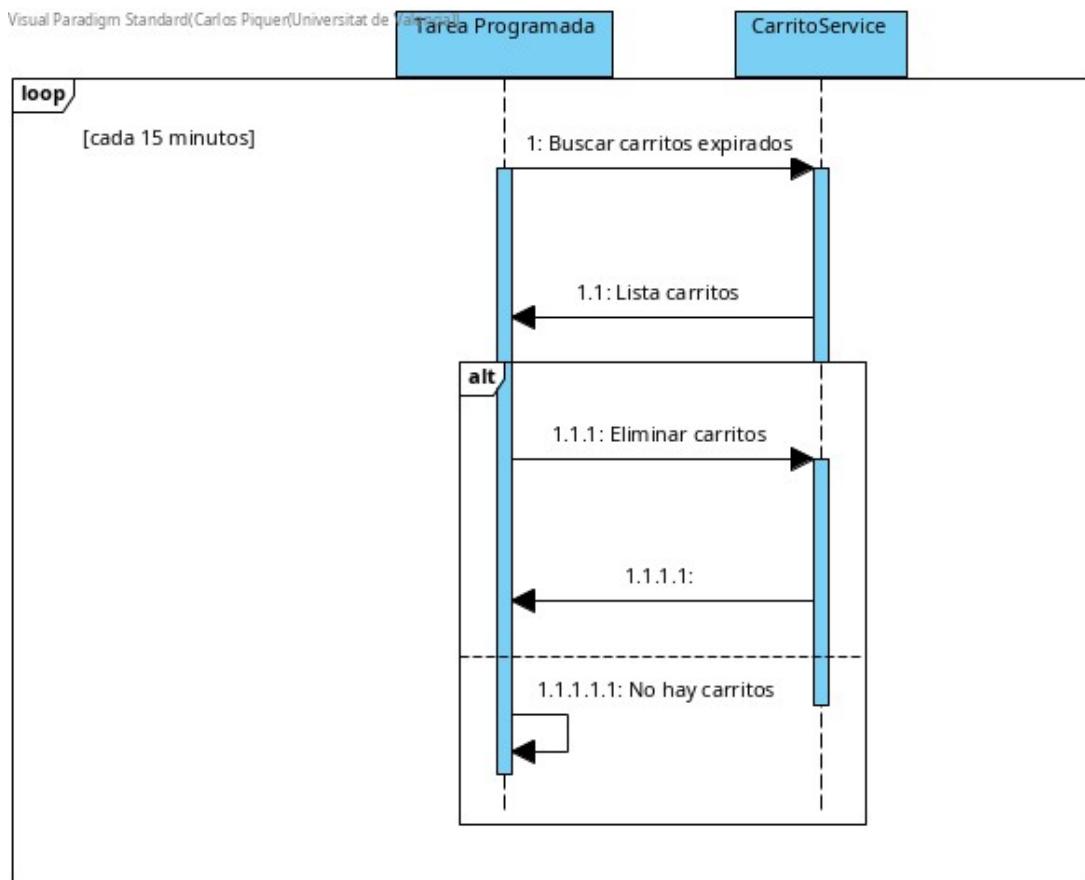


Figura 5.7: Diagrama de secuencia — abandono de carrito

Bajo stock Por último, se representa el proceso de notificación de stock bajo. Cuando el inventario de un producto alcanza un umbral mínimo, el sistema ejecuta una tarea automatizada que envía un aviso al administrador mediante el agente *n8n*, permitiendo reponer el producto a tiempo.

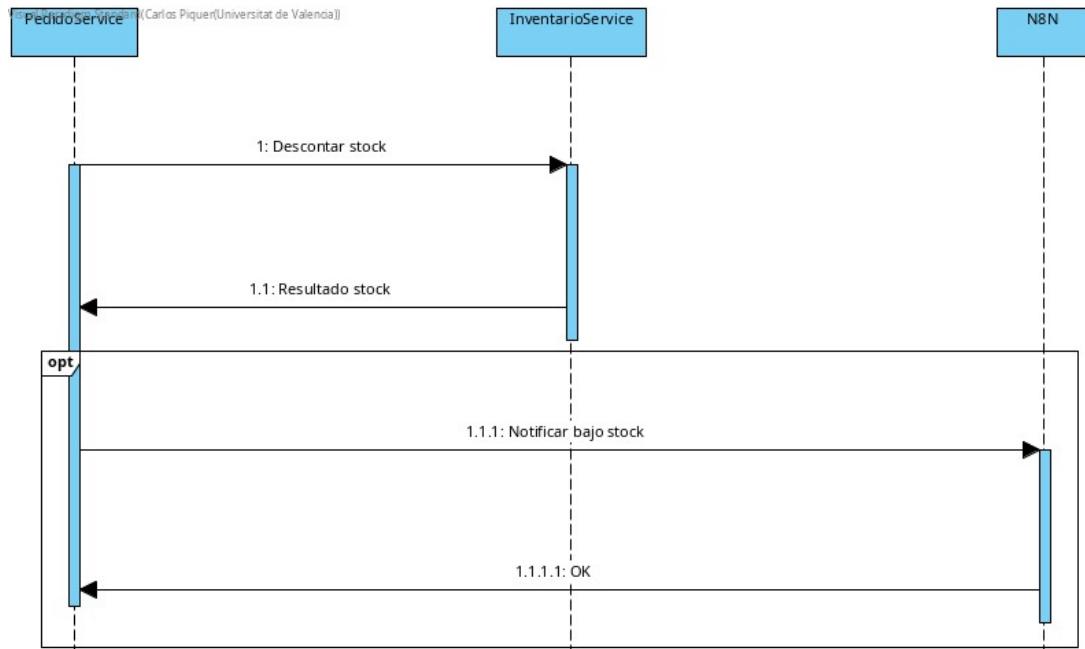


Figura 5.8: Diagrama de secuencia — notificación de bajo stock

5.3. Diseño de la base de datos

El diseño de la base de datos se llevó a cabo inicialmente mediante *dbdiagram.io*, para definir de manera visual las entidades, sus atributos y relaciones. Posteriormente, el modelo se validó e implementó en *MySQL Workbench*, comprobando su coherencia con el backend en *Spring Boot*.

5.3.1. Modelo entidad–relación

El modelo E/R del sistema representa las entidades principales (*usuarios*, *productos*, *carritos*, *pedidos*, *pagos*, *facturas*) y las relaciones entre ellas. Estas relaciones garantizan la integridad referencial mediante el uso de claves primarias, foráneas y restricciones de unicidad.

5.3.2. Modelo relacional y prototipo inicial

El modelo relacional resultante se compone de tablas como *usuarios*, *productos*, *carritos*, *carrito_items*, *pedidos*, *pagos* y *facturas*. La Figura 5.9 muestra el esquema final diseñado.

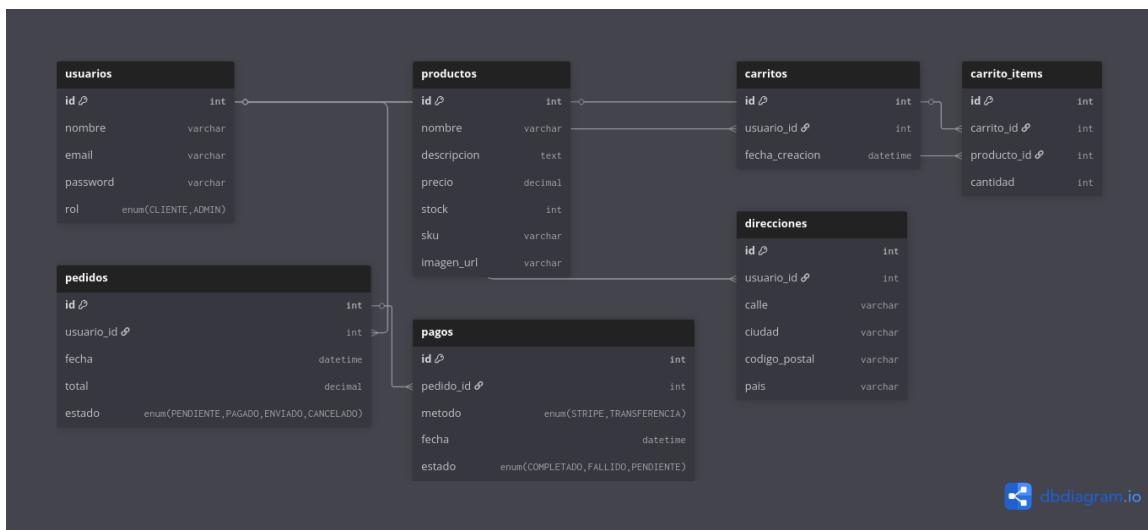


Figura 5.9: Prototipo de la base de datos relacional

Capítulo 6

Implementación y pruebas

6.1. Implementación

6.2. Pruebas funcionales

6.3. Pruebas de rendimiento

6.4. Pruebas de usabilidad

Capítulo 7

Conclusiones

7.1. Revisión de costes

7.2. Conclusiones

7.3. Trabajo futuro

Apéndice A

Apéndice

A.1. Ejemplos del lenguaje de marcado Latex

This document is an example of BibTeX using in bibliography management. Three items are cited: *The L^AT_EX Companion* book [?], the Einstein journal paper [?], and the Donald Knuth's website [?]. The L^AT_EX related items are [?, ?]¹.

Texto en el párrafo 1.

Texto en el párrafo 2.

Texto en el párrafo 3.

- Consideración 1
- Consideración 2

1. Punto 1

2. Punto 2

A continuación se muestra una ecuación:

$$\int_0^1 \frac{1}{x^2 + 1} dx$$

Podemos incluir imágenes en formato: png, pdf o jpg.

En la figura A.1 se muestra un diagrama realizado con <https://www.yworks.com/products/yed>:

Imagen 1

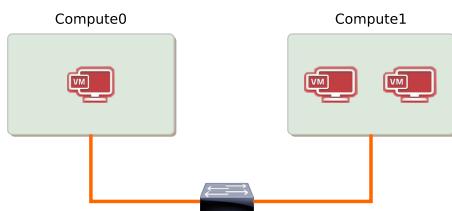
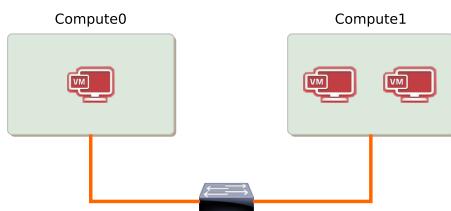


Imagen 2



¹Esto está tomado de https://www.overleaf.com/learn/latex/Bibliography_management_with_bibtex

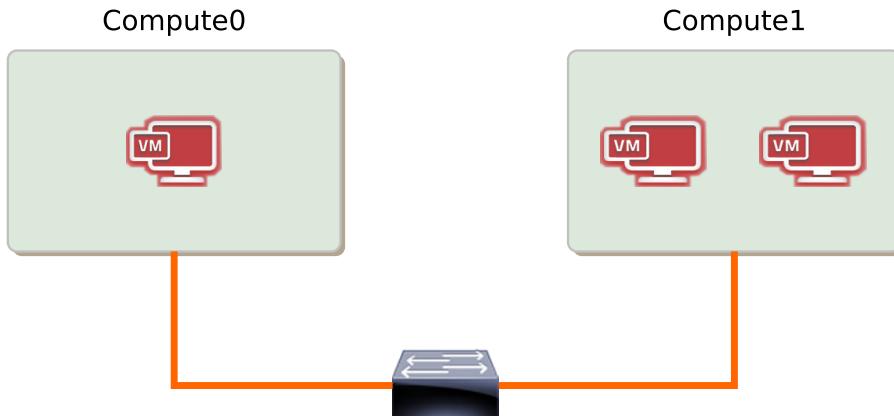


Figura A.1: Esta es una figura que latex decide donde colocar (floating) en el documento.

Este es un ejemplo de una tabla:

Columna 1	Columna 2
1	2

O la misma tabla centrada:

Columna 1	Columna 2
1	2

Para generar el fichero PDF:

```
pdflatex ejemplo-memoria.tex  
bibtex ejemplo-memoria  
pdflatex ejemplo-memoria.tex
```

También se puede usar `latexmk` que automáticamente regenera la bibliografía.

```
latexmk -pdf ejemplo-memoria.tex
```

Bibliografía

- [1] Wix.com Ltd. Wix Help Center. <https://support.wix.com/>, 2025. Consultado en octubre de 2025.
- [2] Squarespace Inc. Squarespace Help Center. <https://support.squarespace.com/>, 2025. Consultado en octubre de 2025.
- [3] PrestaShop S.A. Documentación Oficial de PrestaShop. <https://devdocs.prestashop-project.org/>, 2025. Consultado en octubre de 2025.
- [4] Google Developers. Angular Documentation. <https://angular.io/docs>, 2025. Consultado en octubre de 2025.
- [5] Meta Platforms, Inc. React Official Documentation. <https://react.dev/>, 2025. Consultado en octubre de 2025.
- [6] Evan You. Vue.js Official Guide. <https://vuejs.org/guide/introduction.html>, 2025. Consultado en octubre de 2025.
- [7] VMware, Inc. Spring Boot Documentation. <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>, 2025. Consultado en octubre de 2025.
- [8] OpenJS Foundation. Node.js Documentation. <https://nodejs.org/en/docs/>, 2025. Consultado en octubre de 2025.
- [9] Django Software Foundation. Django Official Documentation. <https://docs.djangoproject.com/en/5.0/>, 2025. Consultado en octubre de 2025.
- [10] Zapier Inc. Zapier Help Center. <https://help.zapier.com/>, 2025. Consultado en octubre de 2025.
- [11] Celonis SE. Make (formerly Integromat) Documentation. <https://www.make.com/en/help>, 2025. Consultado en octubre de 2025.
- [12] n8n.io. n8n Documentation. <https://docs.n8n.io/>, 2025. Consultado en octubre de 2025.
- [13] Oracle Corporation. MySQL Documentation. <https://dev.mysql.com/doc/>, 2025. Consultado en octubre de 2025.
- [14] MongoDB Inc. MongoDB Manual. <https://www.mongodb.com/docs/>, 2025. Consultado en octubre de 2025.
- [15] PayPal Holdings Inc. PayPal Developer Documentation. <https://developer.paypal.com/docs/>, 2025. Consultado en octubre de 2025.

- [16] Redsys Servicios de Procesamiento S.L. Manual de Integración de Pasarela de Pagos. <https://pagosonline.redsys.es>, 2025. Consultado en octubre de 2025.
- [17] Stripe Inc. Stripe API Documentation. <https://stripe.com/docs/api>, 2025. Consultado en octubre de 2025.
- [18] Project Management Institute. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)*. PMI, Newtown Square, Pennsylvania, 2021.