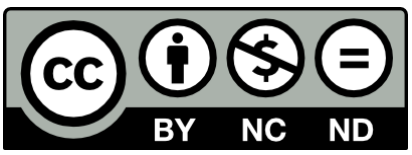
KŌRA

Jordi Navarro González

Ferran Tena Torres

CFGS Desenvolupament d'Aplicacions Web

2025/2026



Resumen proyecto

El proyecto se centra en crear una página web moderna para la venta online de ropa de vestir, ropa deportiva y accesorios. La página incluye diversas marcas de éxito y conocidas a nivel mundial, como Nike, Puma, Adidas, y Pull & Bear. También incluye secciones dedicadas a ropa deportiva, desde cualquier prenda para ir al gimnasio o correr, a camisetas de tus deportistas preferidos.

El objetivo principal ha sido desarrollar un sitio de venta online que ofrezca una experiencia de compra fácil, rápida y segura, al tiempo que agrega valor con funciones interactivas, un sistema de comentarios y atención al cliente personalizada. Para alcanzar nuestro objetivo, se ha diseñado y desarrollado una web con secciones dinámicas que exhiban los productos más vendidos y populares entre los usuarios, complementado con un sistema de reseñas y un programa de puntos que los usuarios puedan canjear por descuentos y productos, buscando asegurar la fidelidad de los clientes y diferenciarse de la competencia.

Con todo lo anterior se busca ofrecer a los usuarios una experiencia accesible y usable a la vez que confiable e interactiva, así aumentando la satisfacción del cliente.

Palabras clave

- FullStack
- Página Web
- Tienda de Ropa
- API
- Ropa Online
- Estilo y moda
- Base de datos
- Puntos canjeables

Abstract

Our goal is to develop a contemporary website that sells clothing, accessories, and sportswear online. Nike, Puma, Adidas and Pull & Bear will be among the successful and globally known brands that will be featured on the site. There will be sections devoted to sportswear, including clothes for gym or running, and t-shirts featuring your favorite athletes.

Our primary objective is to create an online sales site that provides a convenient, quick, and secure shopping experience, while also adding value through interactive features, a feedback system, and personalized customer service. To achieve our goal, we want to design and develop a website with dynamic sections that showcase the best-selling and popular products among users, complemented by a review system and a program of points that users can redeem for discounts and products, seeking to ensure customer loyalty and differentiate from the competition.

With all of the above, we aim to provide users with an accessible and usable experience that is both reliable and interactive, thus increasing customer satisfaction.

Keywords

- FullStack
- Website
- Clothing Store
- API
- Online Clothing
- Style and Fashion
- Database
- Redeemable Points

Índice

1. Introducción

Las páginas web se han vuelto parte del día a día. Antes, para buscar trabajo, consultar los resultados de fútbol o incluso organizar una reunión, había que recurrir a periódicos o desplazarse físicamente. Ahora todo está a unos escasos clics de distancia.

Actualmente, cualquier persona busca la máxima comodidad en su día a día y el máximo ahorro de tiempo posible; ahí es donde las tiendas web encuentran un porqué. Las páginas web permiten que cualquier persona pueda comprar desde la comodidad de su casa, o curiosear entre diversos productos sin moverse del sofá. Las páginas web te permiten hacer más cosas con menos esfuerzo.

En este caso, KŌRA es una tienda online de ropa para quienes quieren vestir bien sin la necesidad de desplazarse o hacer largas colas para comprar. KŌRA reúne en un mismo sitio las marcas más reconocidas, haciendo que puedas encontrar lo que buscas sin tener que pasar mucho tiempo mirando en diferentes tiendas, permitiendo al usuario encontrar más fácilmente ofertas y las prendas que busca, y permitiendo que puedas hacer conjuntos de forma más fácil al acceder a una gran cantidad de productos.

KŌRA nace con el propósito de combinar moda, comodidad y tecnología para ofrecer una experiencia de compra rápida y sencilla.

1.1 Contexto

Actualmente, el mercado de la ropa está muy segmentado. De forma general, las tiendas solo venden sus propios productos, lo que hace que la gente tenga que ir buscando de forma individual entre tienda y tienda, haciendo que el proceso sea lento y tedioso para aquellas personas que lo único que quieren es vestir bien, sin pasar horas y horas en tiendas buscando la prenda que mejor se adapte a él y a su estilo.

KŌRA busca hacer que esa experiencia ya no sea necesaria, unificando a las marcas de más exito en un mismo sitio, haciendo más fácil encontrar las prendas perfectas para tu outfit y ahorrando tiempo para el usuario. Actualmente no hay casi ninguna tienda que haga esto, así que se ha querido ofrecer al usuario la mejor experiencia a la hora de comprar ropa.

1.2 Justificación

En la actualidad, la oferta de ropa en el mercado es muy amplia; existen pocas plataformas que reúnan una gran variedad de marcas en un mismo sitio web, lo que limita las opciones de los consumidores que buscan una experiencia de compra diferente.

Por este motivo, KŌRA propone desarrollar una página web que ofrezca una alternativa accesible, fiable y cómoda para quienes deseen comprar ropa desde la comodidad de su sofá.

KŌRA tiene como objetivo proporcionar una plataforma intuitiva y atractiva donde los usuarios puedan descubrir, comparar y adquirir prendas de distintas marcas sin necesidad de recurrir a múltiples sitios web.

De esta forma, se busca ampliar la oferta de moda online, aportando una nueva opción a los consumidores y fomentando la competencia en el sector.

1.3 Objetivos

1.3.1 Objetivos generales:

1. Crear una página web innovadora de venta de ropa.
2. Fidelizar a los clientes.

1.3.2 Objetivos específicos:

- Creación de un catálogo online accesible y completo.
- Tablas interactivas con ranking de ventas.
- Sistemas de reseñas y comentarios
- Proceso de compra y login seguro.
- Comenzar y terminar el proceso de puntos canjeables.
- Página responsive.

1.4 Estrategia y planificación del proyecto

Hay dos estrategias principales que se podrían ejecutar para el desarrollo de esta web:

1.4.1 Desarrollar el producto utilizando otra plataforma

Al desarrollar el producto utilizando otra plataforma, el proceso pasa a centrarse en utilizar las herramientas o el sistema que se utilice para adaptar la base que se nos otorga al producto final que se ha querido.

A continuación, mostraré las ventajas y desventajas:

Ventajas:

- **Ahorro de tiempo:** Gran parte de las funcionalidades que tienes que implementar ya están integradas en la plataforma, haciendo el desarrollo mucho más rápido.

- **Facilidad de uso:** Las plataformas ofrecen herramientas intuitivas que no requieren de conocimiento previo.

- **Coste inicial reducido:** La mayoría de las plataformas ofrecen alojamiento de forma gratuita con la suscripción a sus servicios, aparte debido a su facilidad no es necesario invertir tanto en su desarrollo.

Desventajas:

- **Limitaciones en la personalización:** Aunque permite modificar el diseño a gusto del usuario, existen ciertas limitaciones a la hora de hacerlo si no se tienen conocimientos avanzados.

- **Dependencia de la plataforma:** Al utilizar una plataforma externa el proyecto queda ligado a esta, por lo que el usuario se tiene que adaptar a cualquier cambio que surja en la plataforma.

- **Rendimiento limitado:** En proyectos grandes estas plataformas suelen volverse lentas y no ser suficiente.

- **Menor diferenciación:** Al utilizar plantillas o estructuras comunes los sitios que comparten plataformas suelen parecerse en gran medida a otras páginas creadas con la misma herramienta.

1.4.2 Desarrollar el producto desde cero

Al desarrollar el producto desde cero el proceso se basa en crear la página web a partir del uso de frameworks y lenguajes de programación, esto hace que no tengas ninguna base con la que comenzar.

A continuación se mostraran las ventajas y desventajas:

Ventajas:

- **Personalización total:** Al hacer el código desde cero puedes crear la página con la estructura y funcionalidades que mejor se adapten al proyecto.

- **Independencia tecnológica:** Al no depender de plataformas externas el desarrollador y el cliente tienen todo el control.

- **Mejor rendimiento y optimización:** Al crear el código desde cero, este se puede optimizar para que este tenga un mejor rendimiento.

- **Escalabilidad:** Al hacer de cero el código es más fácil escalarlo en caso de que se tengan que añadir nuevas funcionalidades o que el número de usuarios en línea se incremente bruscamente.

Desventajas:

- **Mayor tiempo de desarrollo:** Crear una web desde cero requiere muchas más horas de trabajo y planificación que utilizar una plantilla o plataforma existente.

- **Coste inicial más alto:** Al ser más tiempo de desarrollo, se necesitan trabajadores cualificados y otros recursos (servidores, dominios, herramientas para el desarrollo, etc) el coste se eleva mucho.

1.4.3 Estrategia escogida y métodos de planificación

Para el proyecto se ha escogido la segunda estrategia de desarrollo, crear el código desde cero.

Crear el código desde cero permitirá al proyecto tener un mayor grado de personalización permitiendo que la página tenga identidad propia, además permitirá que en caso de ser necesario se pueda escalar la página para más usuarios de una forma más sencilla, y por último tener todo el control de la página, así permitiendo que el producto no se tenga que adaptar a nada externo.

1.5 Metodología de trabajo

Para el desarrollo del proyecto KŌRA , se ha optado por una metodología ágil, específicamente Scrum, debido a su mejor adaptación a la naturaleza colaborativa e iterativa de nuestro portal web. El objetivo primordial es mantener una planificación adaptable que permita responder a las modificaciones que puedan surgir durante el desarrollo, lo que a su vez mejora la calidad del producto final y minimiza los riesgos asociados.

A diferencia de las metodologías convencionales, como el modelo en cascada (Waterfall), donde es necesario completar cada fase antes de dar inicio a la siguiente, Scrum permite descomponer el proyecto en sprints breves, lo que posibilita la implementación y verificación continua de funcionalidades. Esta

organización favorece la identificación temprana de errores y mejora la comunicación entre los miembros del equipo.

Cada sprint incluirá las siguientes etapas:

- ➔ **Planificación**: definición de las tareas prioritarias y objetivos a alcanzar en cada iteración.
- ➔ **Desarrollo**: implementación del código y de las funcionalidades acordadas.
- ➔ **Revisión y pruebas**: comprobación del correcto funcionamiento de las nuevas partes añadidas y revisión conjunta.
- ➔ **Retrospectiva**: análisis del proceso para detectar puntos de mejora de cara al siguiente sprint.

Para el seguimiento y la organización del trabajo se ha utilizados las siguientes herramientas:

- **GitLab:** para el control de versiones y el trabajo colaborativo del código fuente. La gestión del flujo de trabajo se opta por Kanban.
- **Google Drive:** para compartir documentación y mantener un registro de todos los materiales del proyecto.
- **Figma:** para el diseño de la interfaz y la creación de prototipos visuales antes de la implementación.
- **Diagrama de Gantt (con GanttProject o herramienta similar):** para tener una visión general de la planificación temporal del proyecto y del progreso global.

Con esta metodología se espera conseguir una mejor comunicación, mayor eficiencia en el desarrollo y un producto final que responda mejor a las necesidades de los usuarios y a los objetivos iniciales del proyecto.

1.6 Estimación económica inicial

Se ha preparado un PRESUPUESTO inicial con las siguientes medidas razonables para que inicialmente se pueda empezar el proyecto. Podría haber desviaciones de costes en función de errores que surjan (alguna subida, bajada o ampliación).

Para concluir el presupuesto del proyecto se ha basado en las siguientes características:

- Tiempo estimado de los trabajadores
- Recursos de la empresa (luz, servidores)...
- Servicios de la empresa (Espacio del servidor, Servidores, mantenimiento)...
- Proceso de prueba de la web
- Análisis profundo de la aplicación web.

2. Descripción del proyecto

2.1 Análisis de requisitos

Saber los requisitos que el proyecto tendrá es una parte muy importante, así se puede saber que va a necesitar la web y que requisitos son indispensables y cuáles no.

Para terminar el proyecto como mínimo se debería de tener acabado las siguientes requisitos:

Gestión de productos:

La gestión de productos es la base del proyecto, este apartado permite agregar, editar y eliminar productos (nombre, descripción, precio, talla, color, categoría, stock, imágenes, etc)

Gestión de Usuarios:

Para la administración de usuarios, es necesario crear un Portal del usuario que ofrezca todas las opciones disponibles para personalizar sus preferencias. Además, las empresas deberían disponer de un área propia donde puedan gestionar su organización, así como controlar sus procesos de venta.

Tabla interactiva:

Tabla interactiva destinada a mostrar el Top 10 de marcas de ropa con mayores ventas, ofreciendo al usuario la posibilidad de aplicar distintos filtros para ajustar el ranking, visualizar datos actualizados y obtener una comparación clara del rendimiento de cada marca.

Sistema de pedidos y compras:

Toda la información que se encuentra en el sitio se actualizará de manera periódica o siempre que haya un cambio en cualquier parte de la página. Después de un tiempo predeterminado el sistema revisará cada actividad del día, anotando las modificaciones y gestionándolas de manera constante, para que los datos se adapten automáticamente. Esto asegurará que la plataforma

disponga siempre de información exacta, actual y en línea con las acciones realizadas en el sitio.

Valoración y comentarios:

El sistema para valoraciones y comentarios estará creado para ofrecer una experiencia actual, vibrante y completamente enfocada en la calidad. Los usuarios tendrán la oportunidad de expresar sus pensamientos de la forma más simple y cómoda posible, contribuyendo tanto con calificaciones como con comentarios completos que beneficien a la comunidad. Cada nueva revisión se incorporará a la plataforma de inmediato, actualizándose siempre que alguien añada, modifique o elimine un comentario. Además, el sistema hará un análisis continuo de todas las interacciones diarias para identificar tendencias, resaltar opiniones útiles y garantizar que el contenido que se muestra permanezca siempre actualizado, confiable y pertinente. Con este enfoque novedoso y creativo, el área de comentarios se transformará en una herramienta valiosa para los usuarios y en una fuente constante de mejora para la empresa.

2.1.1 Requisitos funcionales

Los requisitos funcionales son aquellos requisitos que hacen que la aplicación web sea usable y cumpla con su función, estos requisitos son básicos para saber la dirección que el proyecto debe tomar, para esto se ha dividido en grandes requisitos que dentro tienen diferentes apartados.

1. Catálogo de productos

Cualquier e-commerce tiene que tener un catálogo de productos funcional con una ejecución excelente para que al usuario le sea intuitivo, algunos de los aspectos más importantes són:

- Organización del catálogo por categorías
- Sistema de búsqueda de productos con filtro (por características)
- Página de producto con información detallada de este (precio,materiales, imágenes, etc)
- Actualización automática del stock de los productos en tiempo real

2. Gestión de usuarios

Para tener una aplicación web efectiva se necesita una buena gestión de usuarios para facilitar que los usuarios se creen una cuenta y la utilicen , algunos de los aspectos más importantes són:

- Registro vía correo electrónico, contraseña o redes sociales
- Sistema de recuperación de contraseña
- Edición de perfil, incluyendo sus datos personales
- Historial de pedidos personal de cada usuario

3. Gestión de la cesta y productos

La gestión de cesta es esencial para almacenar todos los productos que el usuario quiere comprar, dando a los usuarios una compra más cómoda. Algunos de los aspectos más importantes són:

- Cálculo del precio total de la cesta de forma automática
- Gestión del producto dentro de la cesta; añadir, eliminar, cambiar la cantidad
- Finalización de la compra y pasó a la facturación del pedido
- Guardado de la cesta para usuarios registrados

4. Facturación del pedido

La facturación del pedido permite que el pago se haga correctamente y que el pedido finalmente se haga efectivo, algunos de los aspectos más importantes són:

- Capacidad de hacer el pago por diferentes métodos (transferencia, PayPal o tarjeta bancaria)
- Verificación de las transacciones
- Factura/comprobante de compra que se envía al correo electrónico o se descarga formato PDF

5. Envío

El correcto funcionamiento del envio permite que el producto llegue al comprador, algunos de los aspectos más importantes són:

- Gestión de la dirección de envío
- Cálculo del tiempo de entrega estimado
- Tracking del pedido en tiempo real

6. Valoraciones y comentarios

El sistema de valoraciones y comentarios permite a los usuarios saber si un producto es bueno en base a la opinión de otros clientes, algunos de los aspectos más importantes són:

- Posibilidad de valorar un producto comprado con un sistema de puntuación (sobre cinco).
- Capacidad de dejar un comentario en los productos comprados

7. Promociones y descuentos

Es importante mantener a los usuarios informados de productos que les pueden ser interesantes y proveerlos de ofertas que les motiven a comprar, algunos de los aspectos más importantes són:

- Gestión de cupones y promociones
- Recomendaciones personalizadas dependiendo de la actividad del usuario
- Suscripción opcional a un newsletter

8. Panel de control

El panel de control permite al administrador tomar control y hacer cambios manuales si es necesario, algunos de los aspectos más importantes són:

- Gestión de usuarios, productos y pedidos
- Capacidad de consultar estadísticas y tráfico del sitio en tiempo real

- Gestión de categorías y marcas

2.1.2 Requisitos no funcionales

Los requisitos no funcionales son aquellos que no son obligatorios para la aplicación web pero si suben de manera notoria la calidad de esta, para esto se ha dividido en grandes requisitos que dentro tienen diferentes apartados.

1. Rendimiento

Un buen rendimiento de la página hace que el usuario tenga una buena experiencia de compra, algunos de los aspectos más importantes són:

- Tiempo de carga reducido en páginas principales
- Respuestas rápidas en las búsquedas, filtros y navegación
- Soporte de varios usuarios al mismo tiempo sin colapsar
- Optimizar imágenes y otros recursos multimedia

2. Usabilidad

La plataforma debe ser intuitiva, cómoda y fácil de utilizar para cualquier usuario, independientemente de su experiencia tecnológica. Algunos de los aspectos más importantes són:

- Interfaz clara, intuitiva y fácil de navegar
- Proceso de compra simple
- Coherencia de diseño entre páginas dentro de la web
- Facilidad para localizar productos de interés

3. Disponibilidad

La aplicación web tiene que ser accesible para los usuarios de forma continua para recibir visitas y compras en cualquier momento.

- Disponibilidad del servicio 24h
- Capacidad de seguir funcionando aún en periodos de alta actividad
- Sistemas para reaccionar ante posibles caídas

4. Escalabilidad

La aplicación web debe de tener la capacidad de crecer sin necesidad de refactorizar el código, permitiendo subir la capacidad de usuarios y añadir productos y funcionalidades. Algunos de los aspectos más importantes són:

- Capacidad de gestión en caso de crecimiento de usuarios y pedidos
- Posibilidad de añadir nuevo contenido sin afectar al funcionamiento de la aplicación web
- Arquitectura preparada para futuras ampliaciones

5. Seguridad y autenticación

La seguridad permite que no se pueda acceder a los datos y no se roben datos de los usuarios, algunos de los aspectos más importantes són:

- Protección de la información de los usuarios
- Encriptación de contraseñas
- Uso de conexiones seguras vía HTTPS

6. Mantenibilidad

La aplicación debe estar desarrollada de forma que sea sencillo corregirla, actualizarla y mejorarla con el tiempo. Algunos de los aspectos más importantes són:

- Código estructurado, organizado y documentado
- Facilidad a la hora de corregir errores o añadir nuevas funcionalidades
- Separación entre las distintas partes

7. Robustez

La aplicación web debe poderse abrir en diferentes dispositivos y navegadores y verse de forma correcta, algunos de los aspectos más importantes són:

- Compatibilidad con la mayoría de navegadores actuales como Chrome, Brave, Firefox, Safari, etc
- Adaptación del diseño dependiendo del dispositivo y tamaño de la pantalla
- Correcto funcionamiento en diferentes dispositivos, como ordenador, tablets o smartphones

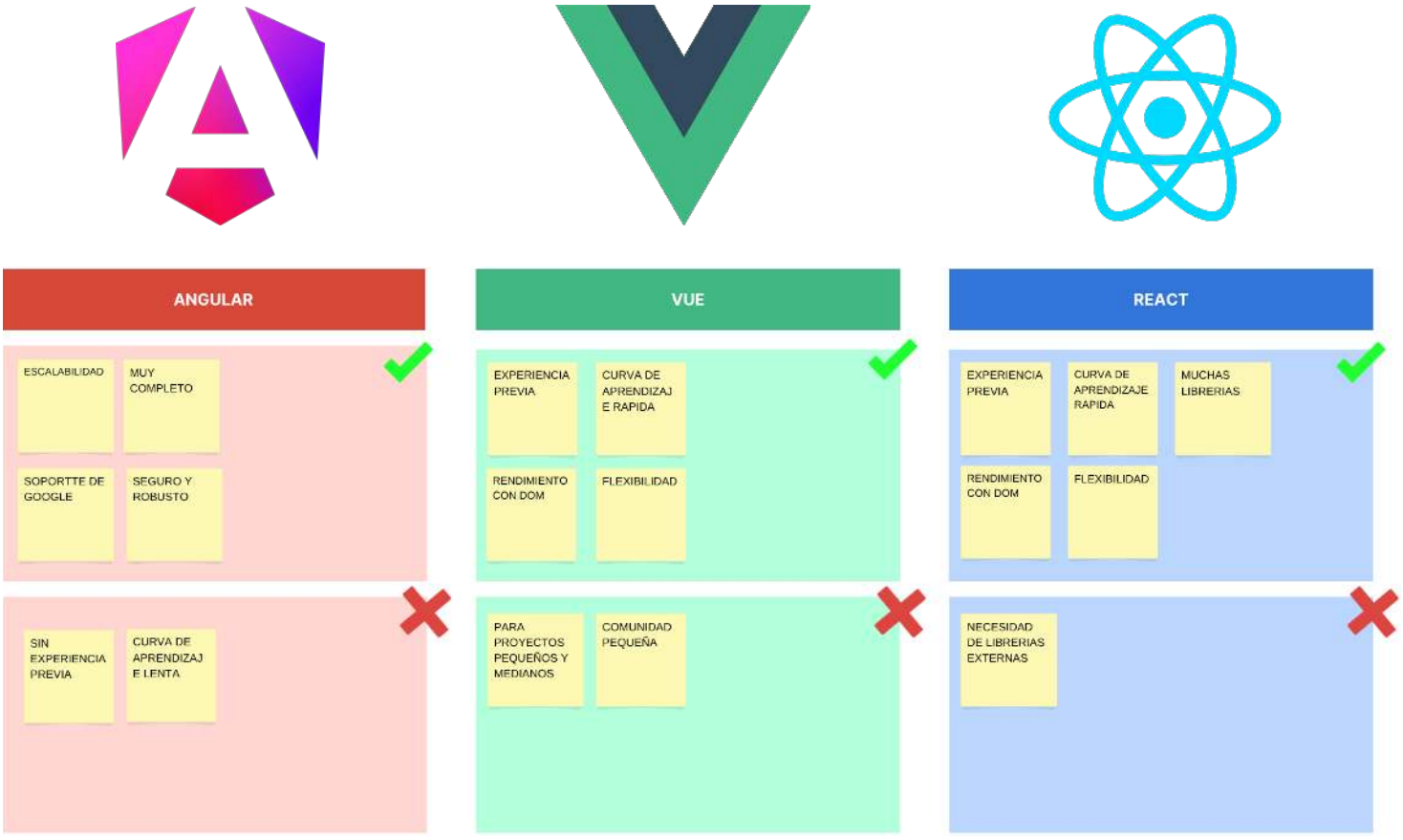
2.2 Tecnologías

2.2.1 Comparativa entre tecnología

Para comparar entre tecnologías se ha utilizado la lista de pros y contras como el método para valorar qué tecnología utilizar para cada tarea.

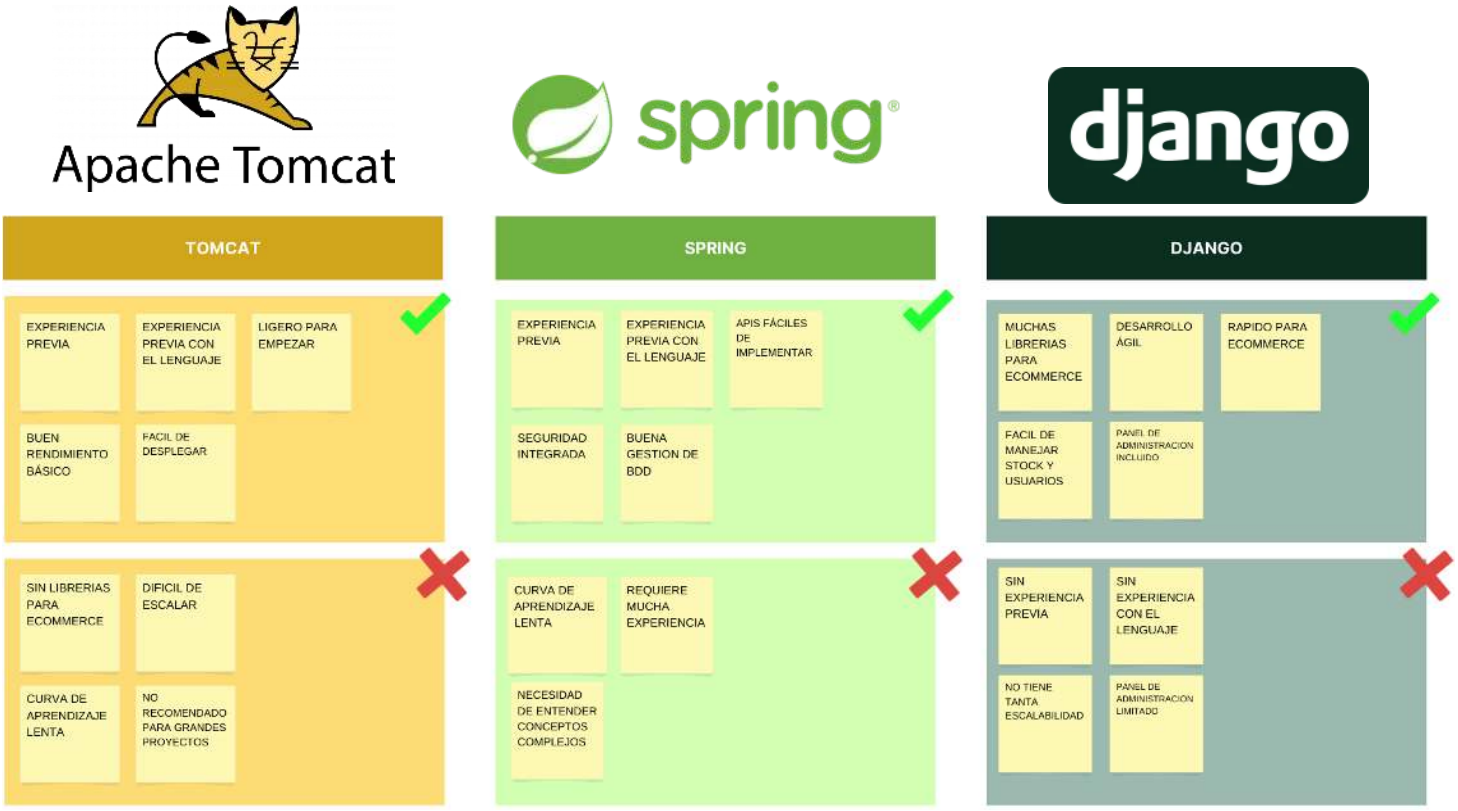
FRONTEND:

En el frontend se utiliza HTML, CSS y JS como lenguajes básicos. Pero a la hora de escoger un framework se ha valorado entre 3, REACT, ANGULAR y VUE.



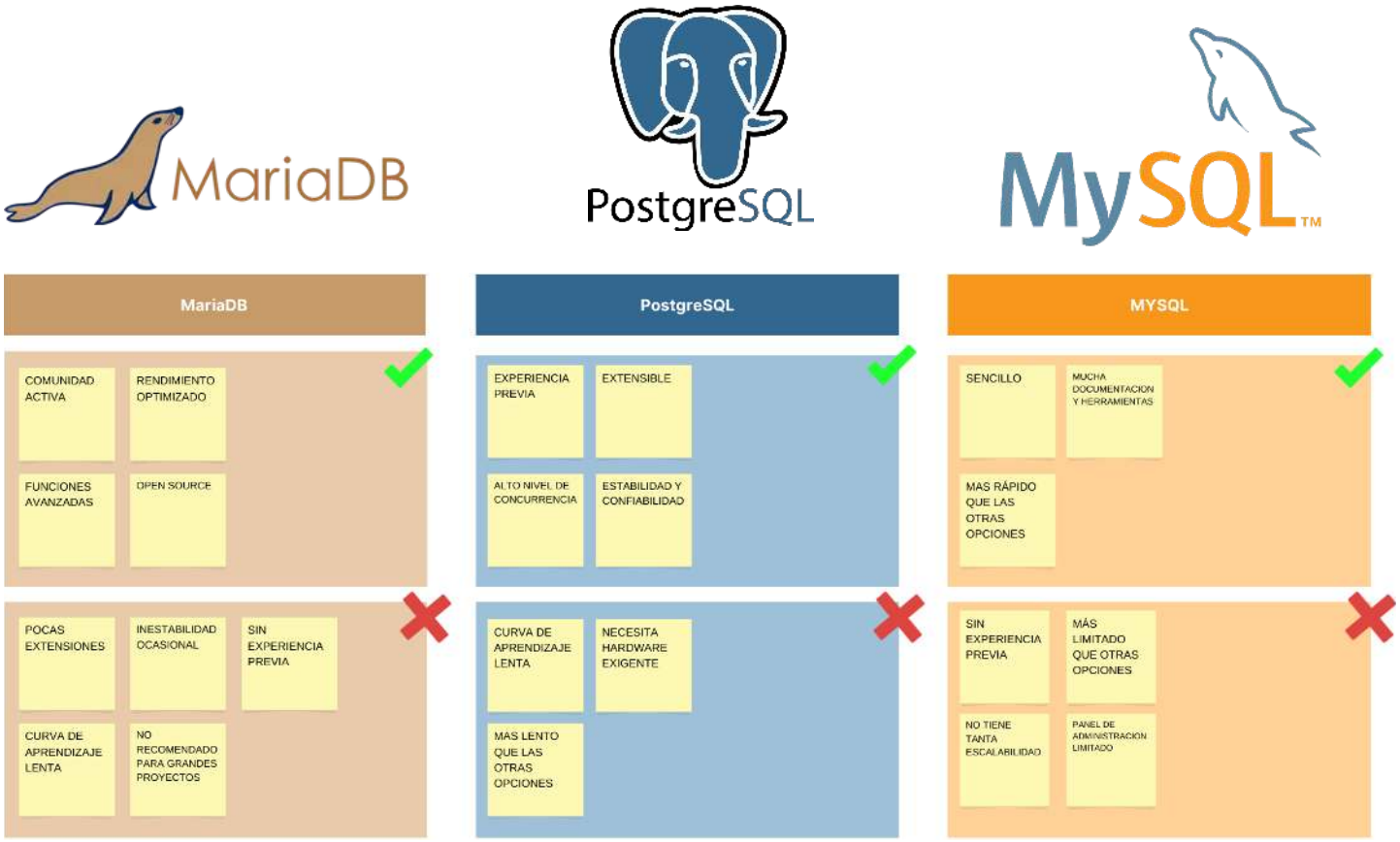
BACKEND:

Para el backend no hay unos lenguajes básicos, aunque se decidirá por utilizar JAVA o Python descartando otros lenguajes como JavaScript. Las tres opciones a valorar.



BASE DE DATOS:

Se ha pensado en diferentes sistemas de gestión de bases de datos, todos basados en SQL.



2.2.2 Tecnologías escogidas

Tras una larga valoración se ha decidido las tecnologías para el proyecto, en la valoración ha sido determinante la experiencia que se tienen con esas tecnologías y que se adapten a las necesidades del proyecto.

FRONTEND:

En el frontend se utilizará HTML, CSS y JavaScript, que son los lenguajes base que se utilizan. El framework/librería escogido ha sido React.

React es la librería más utilizada para el Frontend, ha sido escogido por su versatilidad y por que hay una mínima experiencia previa trabajando con este.

Con REACT se utilizara Next.js, esto nos ayudará a hacer la página más rápida y que tenga mejor SEO, posteriormente se explicará más sobre ello.

Se ha utilizado Tailwind CSS para facilitar el apartado del diseño.

BACKEND:

Para el apartado de Backend el lenguaje escogido ha sido Java, dado que es con el que hay una experiencia previa mayor. El framework que se utilizara es Spring .

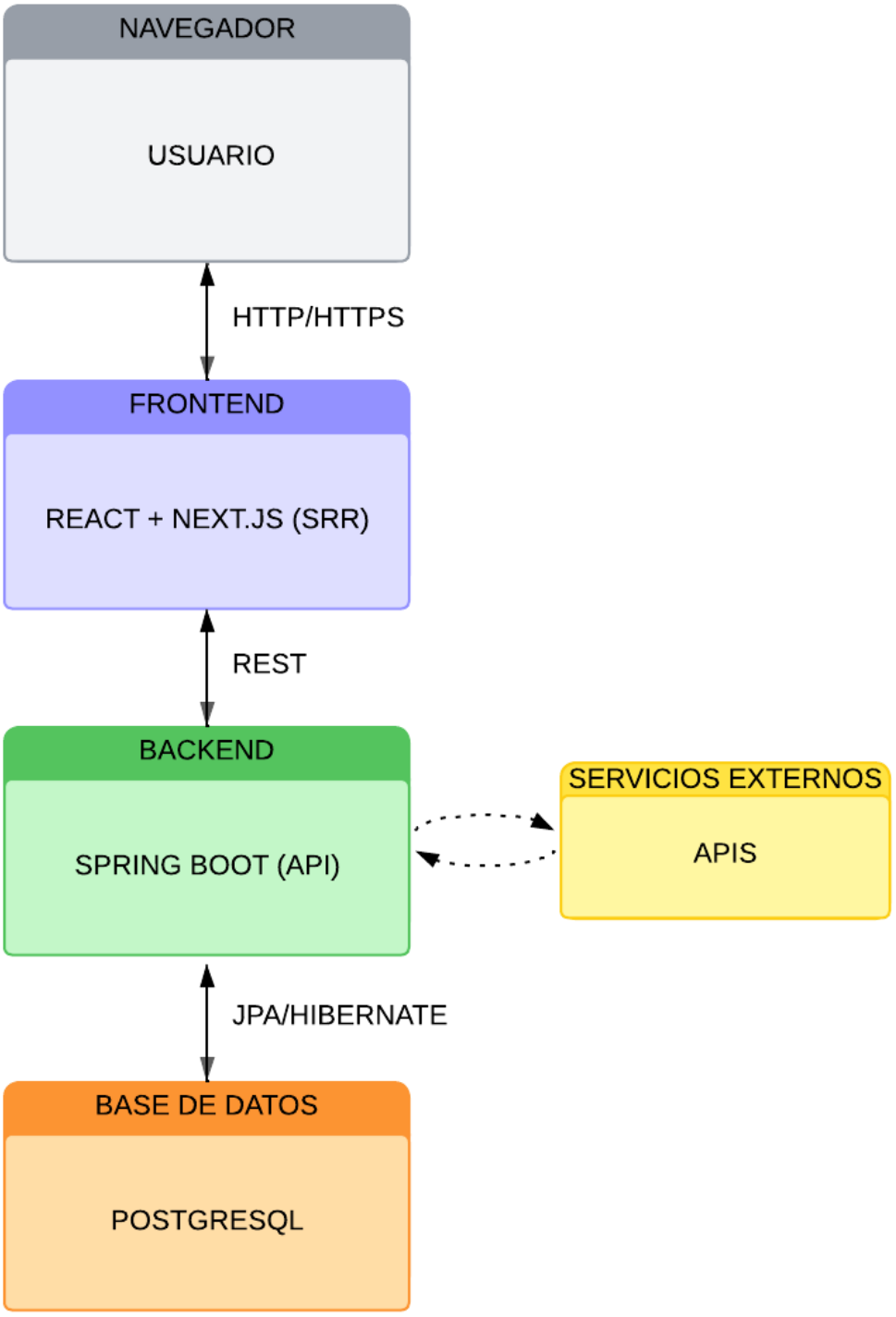
Spring es un framework que simplifica la creación de aplicaciones empresariales, facilitando el desarrollo. También ha influido el actual aprendizaje de este framework en la materia de FullStack.

BASE DE DATOS:

Para la base de datos PostgreSQL ha sido el escogido, ya que la sintaxis entre las diferentes base de datos era muy similar lo que ha hecho un punto de inflexión ha sido la experiencia previa que hay con PostgreSQL.

2.3 Estructura del proyecto

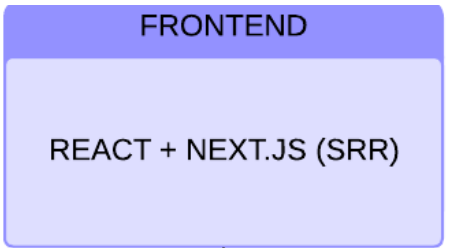
El proyecto sigue la arquitectura cliente servidor, a continuación hay un diagrama de bloques que muestra la estructura del proyecto de forma sencilla:



2.4 Estructura del proyecto

A continuación se explica la estructura base del proyecto y la importancia de cada parte de este.

2.4.1 Frontend



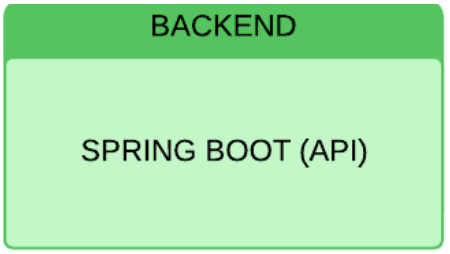
La parte del cliente se ha desarrollado utilizando React junto con Next.js, lo que permite el uso del SSR (Server-Side Rendering).

El SSR hace que el servidor renderice inicialmente las páginas, lo que mejora el posicionamiento SEO y reduce el tiempo de carga de la aplicación.

El frontend es el encargado de mostrar la interfaz para que el usuario pueda interactuar con la tienda y comunicarse con el backend para realizar lo que el usuario pide.

2.4.2 Backend

El backend se ha desarrollado utilizando Spring y funciona como una API REST.

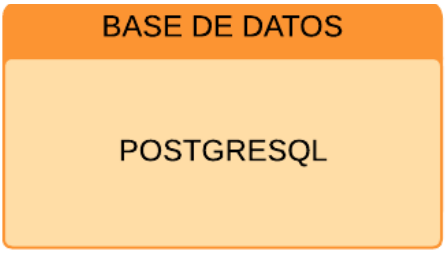


Una API REST permite que el frontend solicite información al backend y reciba respuestas estructuradas, utilizando los métodos HTTP estándar.

El backend es el encargado de gestionar toda la lógica de negocio del sistema, procesa toda la información que se recibe desde el frontend y gestiona toda la información de la base de datos.

2.4.3 Base de datos

Para la base de datos PostgreSQL ha sido el escogido, ya que la sintaxis entre las diferentes base de datos era muy similar lo que ha hecho un punto de inflexión ha sido la experiencia previa que hay con PostgreSQL.
Backend



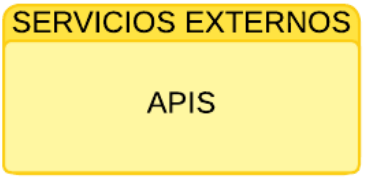
Para la base de datos se ha escogido PostgreSQL, una base de datos relacional.
Esta se comunicará de forma exclusiva con el backend y este gestionará que tiene acceso a qué información.

La base de datos contiene toda la información de los usuarios y de los productos, algo totalmente necesario para el proyecto.

2.4.3 APIs

El proyecto integra diferentes API que integran diferentes funcionalidades.

La comunicación con las API se realiza exclusivamente desde el backend, es este quien da información de la base de datos a las API, esto para tener más seguridad.



3. Desarrollo del proyecto

En un inicio la estructura principal se hizo en MP0615 (Diseño de interfaces web), en esta asignatura se debía crear una página web y se empezó a encarar la misma pensando en el proyecto. La página no hacía falta que tuviera lógica, solo simular el flujo de la aplicación y cierta cantidad de funcionalidades, todo centrado en el diseño visual.

Una vez acabada se busca feedback de compañeros para que comentaran cómo veían la web, si les parecía intuitiva, bonita, práctica y amigable. El feedback recibido no fue el esperado, ya que a varios compañeros les parecía que la web no era intuitiva y con colores más cercanos a otros sectores que a un e-commerce de ropa. Tras esto se decidió hacer un refactor del código para darle una nueva identidad y orientar la aplicación de forma totalmente diferente.

Esta decisión marcó un punto de inflexión muy importante, ya que obligó a repensar casi todos los aspectos de la aplicación, desde cosas simples como un cambio en los colores o los iconos a cambios muy significativos como cambiar por completo pantallas ya establecidas o añadir funcionalidades que no estaban pensadas en un punto inicial, todo con el objetivo de hacer la experiencia del usuario mucho más clara y agradable.

A partir de ese punto se replanteó el proyecto entero en base a las funcionalidades, cada una con un claro objetivo que lograr y con la idea de conseguir que KŌRA no fuera únicamente una página visualmente atractiva, sino un e-commerce de ropa completo, funcional e intuitivo.

En este apartado se explicará, en primer lugar la estructura utilizada tanto en el frontend como en el backend, para después dar paso a explicar cada una de las funcionalidades incorporadas a la aplicación web, donde se especificara el proceso de análisis, investigación y toma de decisiones que se ha llevado a cabo hasta llegar a la solución aplicada.

3.1 Arquitectura del proyecto

La arquitectura del proyecto ha sido un punto clave para saber cómo organizar el proyecto y así diferenciar fácilmente dependiendo de donde se ubica el contenido del archivo.

3.1.1 Backend

Para el backend se ha utilizado una arquitectura por capas basada en el patrón MVC, orientada al funcionamiento de una API REST. Esta estructura permite la separación de las responsabilidades y la organización del código de forma clara.

```
src/
└── main/
    └── java/
        └── com.kora.backend/
            ├── config/
            ├── controller/
            ├── dto/
            ├── mapper/
            ├── model/
            ├── repository/
            ├── service/
            └── BackendApplication
```

- **config:** Esta carpeta contiene las clases encargadas de la configuración general del proyecto. En el caso de KŌRA se utiliza para gestionar la configuración del CORS, la autenticación y la codificación de las contraseñas.

- **controller:** Esta carpeta contiene, como su nombre indica, todos los controllers REST de la aplicación, que reciben las peticiones HTTP para posteriormente procesarlas y así llamar al servicio indicado y devolver la respuesta adecuada al cliente.

- **dto:** Esta carpeta contiene, como su nombre indica, los DTO. Estas clases se utilizan para controlar y limitar la información que se envía o se recibe del frontend, así evitando exponer datos sensibles o innecesarios.

- **mapper:** Esta carpeta contiene las clases encargadas de transformar las entidades en DTO y viceversa. Posteriormente se implementan en los servicios para que se pueda hacer uso de ellas.

- **model:** Esta carpeta contiene las entidades, aquellas clases que contienen los atributos y relaciones que se ven plasmadas en la base de datos, a partir de estas clases se crearán DTOs para limitar los datos que se envían al frontend.

- **repository:** Esta carpeta contiene las interfaces que se utilizaran en cada servicio, estas funciones permiten realizar operaciones CRUD sobre la base de datos.

- **service**: Esta carpeta contiene la lógica de negocio. Esta es la capa donde se realizan las operaciones principales del backend.

- **BackendApplication:** Esta es la clase principal desde la cual se inicia el backend.

3.1.2 Frontend

Para el frontend se ha utilizado una arquitectura que se adaptase al funcionamiento de App Router de Next.js. Esta misma organiza la aplicación por rutas, facilitando el mantenimiento del código, la separación de responsabilidades y la localización de las secciones dentro del proyecto.

La aplicación se organiza dentro de la carpeta src, donde se ha separado el contenido de forma principal en cuatro grandes bloques:

```
src/
├── api/
├── app/
├── components/
└── styles/
```

- **api**: Esta carpeta contiene las funciones encargadas de comunicarse con el backend, separadas por funcionalidad. Esto permite centralizar todo el código que se ocupa de realizar las operaciones CRUD, evitando que este se repita dentro de diferentes componentes o páginas de la aplicación.

- **app**: Esta carpeta contiene las páginas de la aplicación, organizadas por secciones. Al utilizar App Router, las rutas se generan a partir de la estructura de carpetas del proyecto, por lo que cada carpeta dentro de *app* es una ruta en la aplicación web.

- **components:** Esta carpeta contiene los componentes que se utilizan en el proyecto. Estos componentes son partes del código o de la interfaz que se utilizan en diferentes secciones en toda la aplicación. El uso de los componentes nos permite aplicar la abstracción y así evitar repetir la misma estructura en diferentes partes de la aplicación.

- **styles:** Esta carpeta contiene archivos CSS utilizados en el proyecto. Aunque se utiliza Tailwind CSS para la mayoría del estilado, estos archivos permiten definir estilos reutilizables a través de la aplicación, volviendo a aplicar la abstracción para no repetir el mismo código en diferentes partes.

3.2 Gestión de usuarios

El objetivo de esta funcionalidad es la de crear eficientemente un sistema que pueda crear usuarios a través de una experiencia convencional de registro. Permitir que puedan entrar a través de un login, que guarde los datos del cliente y los pedidos que ha hecho.

Desde el primer momento el registro de usuarios funcionó correctamente , los datos se guardaron en la base de datos sin problemas. Una vez comprobado su funcionamiento, se implementó el encriptado de la contraseña al guardarla en la base de datos, algo básico para mantener la seguridad, para lo que se utilizó BCrypt, que permitió gestionar la codificación de contraseñas fácilmente. Para la autenticación se decidió utilizar JWT, ya que permite generar un token cuando el usuario inicia sesión correctamente y utilizarlo después para identificarlo en las peticiones protegidas.

El flujo implementado comienza cuando el usuario introduce su correo electrónico. El backend comprueba el correo, si existe se redirige a login para pedirte la contraseña, en caso de lo contrario se redirige al formulario para registrarse. En caso del login se verifica que la contraseña introducida sea la correcta comparándola con la contraseña codificada en la base de datos. Si los datos son correctos, el backend genera un token JWT y lo devuelve al frontend para mantener la sesión del usuario.

En una primera versión, el token recibido tras el inicio de sesión se almacenaba en el navegador mediante localStorage. Sin embargo, posteriormente se decidió cambiar este sistema y utilizar cookies, ya que se consideró una opción más adecuada para gestionar la sesión dentro de la aplicación.

3.2.1 Backend

El backend de la gestión de usuarios se construyó sobre Spring Boot con una arquitectura en capas compuesta por el controlador `AuthController`, el servicio `AuthService`, la entidad `User` y el repositorio `UserRepository`. La entidad `User` almacena los campos esenciales: nombre real (para envíos), nombre de usuario público único, email único, contraseña codificada, rol y dirección.

El flujo de autenticación comienza en el endpoint `POST /auth/check-email`, que comprueba si el email introducido ya existe en la base de datos y devuelve `LOGIN` o `REGISTER` según el resultado, permitiendo así redirigir al usuario a la pantalla correcta. El registro se realiza a través de `POST /auth/register`, que valida que el email y el nombre de usuario no estén ya en uso antes de crear el nuevo usuario; la contraseña se codifica en el momento del guardado mediante BCrypt, gestionado por el `PasswordConfig`. El inicio de sesión se gestiona en `POST /auth/login`, donde el servicio recupera al usuario por email, compara la contraseña introducida con el hash almacenado usando `passwordEncoder.matches`, y si las credenciales son correctas genera un token JWT mediante el `JwtService`.

El `JwtService` construye el token incluyendo en los claims el `id`, el `email`, el `name`, el `username` y el `role` del usuario, con una caducidad de 24 horas. Para cada petición autenticada, el `JwtFilter` intercepta la cabecera `Authorization`, extrae y valida el token, y carga los detalles del usuario en el `SecurityContextHolder` para que Spring Security pueda aplicar las reglas de acceso definidas. El sistema también contempla la autenticación mediante OAuth (Google y Facebook) a través de `POST /auth/oauth-login`, que crea el usuario si no existe asignándole un nombre de usuario derivado de su nombre, y en caso de existir actualiza sus datos de perfil. Adicionalmente se implementó la recuperación de contraseña mediante token de un solo uso con caducidad de 30 minutos, enviado por email a través del `EmailService`.

3.2.2 Frontend

En el frontend, el flujo de autenticación se diseñó en tres páginas diferenciadas bajo `/auth`. La primera (`/auth/page.tsx`) presenta un único campo de email con el botón de continuar; al enviarlo, llama a `checkEmail` y redirige al usuario a `/auth/login` si el email ya está registrado, o a `/auth/register` si es nuevo. El email introducido se guarda tanto en el `AuthContext` como en el `sessionStorage` para preservarlo en caso de recarga de página.

La página de registro (`/auth/register`) presenta los campos de nombre de usuario, contraseña, nombre y apellidos. El nombre de usuario se valida en tiempo real con un debounce de 500 ms que llama a `checkUsername` para comprobar su disponibilidad, mostrando un icono de carga, confirmación o error según el resultado. La contraseña se valida localmente contra una expresión regular que exige mayúscula, minúscula, número y símbolo especial, e impide enviar el formulario si no se cumplen los requisitos. Tras el registro exitoso, se realiza automáticamente el login para obtener el token JWT y redirigir al perfil.

La página de login (`/auth/login`) recibe la contraseña y, antes de completar el inicio de sesión, comprueba si el dispositivo está marcado como de confianza para ese usuario mediante una cookie `2fa_trusted` vinculada al email. Si el dispositivo no está en la lista de confianza, se envía un código de verificación en dos pasos al email del usuario y se muestra el modal `TwoFactorModal` para introducirlo. Una vez superada la verificación, el token JWT obtenido del backend se guarda en una cookie `httpOnly` mediante una llamada al API Route de Next.js `/api/auth/set-token`, que establece la cookie con los flags `httpOnly`, `secure` y `sameSite: strict` para protegerla de accesos desde JavaScript. La duración de la cookie es de 1 día por defecto o 30 días si el usuario marcó la opción "Recuérdame". El `AuthContext` gestiona el estado global de sesión consultando al arranque el endpoint `/api/auth/me`, que decodifica el token de la cookie en el servidor y devuelve los datos del usuario autenticado sin que el token salga nunca al cliente.

3.3 Gestión de productos

El objetivo de esta funcionalidad es desarrollar un sistema de gestión de productos que permita almacenar, organizar y mostrar las prendas disponibles en la tienda.

Para la gestión de productos fue necesario hacer una buena base, aunque un primer momento se guardó el stock de forma general y no por cada talla , se cambió desde una parte temprana del proyecto porque no era la forma correcta de enfocarlo, adaptando el stock del producto dependiendo de cada talla.

En el frontend, el objetivo principal era mostrar los productos de una forma visual, clara y minimalista. Al principio los productos se mostraban directamente haciendo un map en cada página, pero al ver que esta estructura se repetía en varias partes de la aplicación, se decidió crear un componente reutilizable para la tarjeta del producto. De esta forma, se unifica el diseño y se ahorra escribir el mismo código en muchos sitios. Si clicas en este salia la pagina de detalle del producto donde se podia añadir al carrito.

Después el producto también se gestiona en la cesta, donde se tiene que cambiar la cantidad si el usuario lo necesitaba y se tiene que obtener algunos datos para mostrarlos, y posteriormente que la prenda se comprase.

3.3.1 Backend

Para realizar la gestión de productos se han creado tres entidades y un Enum, estas son:

Product: Cuenta con una id (Long) generada automáticamente, junto con la información principal de cada producto, como el nombre, descripción, precio, categoría e imágenes. Esta entidad representa la información general de cada prenda dentro de la tienda.

Size: Representa las diferentes tallas disponibles de los productos, permitiendo reutilizar las mismas tallas entre distintas prendas.

ProductSize: Es la entidad encargada de relacionar un producto con una talla concreta. Además de guardar la relación entre ambas entidades, también

almacena el stock disponible de cada talla, permitiendo controlar el inventario de forma individual.

Gender: Es el enum que establece los géneros a los que está dirigido la ropa, con cuatro estados, MEN , WOMEN, KIDS y UNISEX.

Se hicieron los mappers, DTOs y un único service, controller y repository donde estarían los métodos que se ejecutan para crear,modificar, obtener o eliminar la cesta y el controlador para dirigir las peticiones.

3.3.2 Frontend

Para el frontend solo se hicieron diferentes componentes y páginas para mostrar los productos y para facilitar lo anterior, aparte del endpoint, donde se crearon las funciones necesarias para conectar el frontend con el backend.

Como componente principal se hizo el ProductCard, contiene como el nombre indica una tarjeta preestablecida para en vez de en donde se tenga que mostrar tener que volver a hacer todo el código solo tener que mostrar un map de la tarjeta.

Otro componente es el ProductSlider que es simplemente un componente para mostrar productos con la forma de un carrusel de todas las tiendas de ropa.

El último componente es RecomendedProducts que se ubica solo dentro de la página de detalles de cada producto y simplemente muestra un carrusel con productos de la misma categoría.

En cuanto a las páginas, se creó una página de búsqueda donde se muestran todos los productos disponibles y se permite filtrarlos según los filtros básicos de una tienda de ropa. También se creó una página de detalle para cada producto, donde el usuario puede ver la información completa de la prenda, sus imágenes, el precio, las tallas disponibles y la opción de añadirlo a la cesta.

También se crearon páginas específicas para secciones como hombre, mujer, niños y sneakers, pero de estas se especifica más adelante en la sección Ampliación.

3.4 Gestión de la cesta

El objetivo de esta funcionalidad es permitir que los usuarios puedan añadir productos a una cesta antes de realizar la compra, gestionando correctamente el stock, las tallas y la persistencia del carrito. También se buscó que tanto los usuarios invitados como los usuarios registrados pudieran utilizar esta funcionalidad.

Para la gestión de la cesta fue necesario plantear cómo almacenar correctamente los productos que el usuario añadía antes de realizar el pago. Desde el principio se vio que no era suficiente con guardar únicamente una lista de productos, ya que también era necesario controlar la cantidad, la talla seleccionada y el stock disponible de cada uno. Además, se añadió un sistema de estados mediante un Enum para gestionar si la cesta estaba activa, expirada o pagada, permitiendo controlar su ciclo de vida y evitar reutilizar cestas antiguas.

Posteriormente, se implementó una diferenciación entre usuarios invitados y usuarios registrados. Los usuarios invitados utilizan una cookie para recuperar su cesta, mientras que los usuarios con sesión iniciada pueden cargar directamente la cesta asociada a su cuenta desde la base de datos, permitiendo mantenerla incluso desde diferentes dispositivos a la vez. Una vez completado el pago, se muestra una página de confirmación y se elimina automáticamente la cookie de la cesta para evitar mantener un carrito ya finalizado.

En general, la funcionalidad no presentó grandes problemas técnicos, aunque sí fue importante organizar correctamente toda la lógica relacionada con la persistencia de la cesta, el control del stock y la separación entre usuarios invitados y registrados, además de dejar preparada la estructura para integrarla posteriormente con la pasarela de pago.

3.4.1 Backend

Para hacer el la gestión de cesta se han creado dos entidades y un Enum, estas son:

Bag: Cuenta con una id (String) generada automáticamente, un atributo opcional para el userId y la fecha de creación, actualización y expiración de la cesta.

También dispone de un enum para saber en qué momento está la cesta y dependiendo de este tratarla de una forma u otra y una lista con todos los productos añadidos dentro de esta (la entidad BagItem).

BagItem: Cuenta con una id (Long) generada automáticamente, una relación con la cesta a la que pertenece, una relación con el producto que representa, la talla escogida y la cantidad de este producto que se quiere comprar.

BagStatus: Es el enum que establece el estado de la cesta, con tres estados, ACTIVE, EXPIRED y PAID.

Se hicieron los mappers, DTOs y un único service, controller y repository donde estarían los métodos que se ejecutan para crear,modificar, obtener o eliminar la cesta y el controlador para dirigir las peticiones.

3.4.2 Frontend

Para el frontend solo se tuvo que hacer la página de la cesta, donde si hay cesta se muestran los productos y si no la hay se muestra otra página indicando e invitando a buscar productos para comprar y los endpoints, donde se crearon las funciones necesarias para conectar el frontend con el backend. Desde esta parte se gestionan las peticiones para obtener la cesta, añadir productos, eliminarlos y modificar cantidades, separando también el comportamiento de las cookies entre usuario invitado y usuario registrado.

3.5 Pasarela de pago

El objetivo de esta funcionalidad es permitir que los usuarios puedan completar la compra de forma segura y cómoda. Para esta parte se utilizó Stripe (en modo dev porque no se puede utilizar en modo producción, debido a que no se manejara dinero real), ya que nos parecía la mejor implementación para hacer esta funcionalidad de la forma más segura posible.

Al principio se usó la plantilla base de Stripe, pero después se quiso utilizar nuestra propia plantilla para realizar el pago, cosa que hizo que se cambiase la gran mayoría de lógica de negocio de esta parte y se crease mucho código en el frontend.

Las tarjetas de pago al estar en modo de prueba solo se permite utilizar números específicos proporcionados por Stripe. En el anexo se ha dejado el enlace al recurso oficial donde hay un listado de tarjetas que se pueden utilizar para realizar los pago dependiendo de la marca que se quiera utilizar, también hay pruebas en caso de se quiera probar una tarjeta rechazada o un intento de fraude: https://docs.stripe.com/testing?testing-method=card-numbers#cards

3.5.1 Backend

Para la parte del backend se crearon principalmente dos controladores. El primero fue PaymentIntentController, encargado de generar el pago mediante Stripe a partir de la cesta del usuario. Desde este controlador se calcula el importe total de la compra y se crea el PaymentIntent, que posteriormente utiliza el frontend para mostrar el formulario de pago.

El segundo fue StripeWebhookController, encargado de recibir la respuesta de Stripe cuando el pago cambia de estado. Gracias a este controlador, el backend puede comprobar si el pago se ha completado correctamente y, en ese caso, marcar la cesta como pagada. Esto permite evitar cerrar una compra antes de tiempo o si la tarjeta falla.

3.5.2 Frontend

Para el frontend se creó la página de checkout, donde el usuario puede introducir los datos de envío y realizar el pago de la compra, la página de success, que cuando se realizaba la compra elimina la cookie de la bolsa del navegador en caso de ser invitado, también se creó los endpoints, donde se crearon las funciones necesarias para conectar el frontend con el backend. Desde esta parte se gestionan las peticiones para obtener la cesta, añadir productos, eliminarlos y modificar cantidades, separando también el comportamiento de las cookies entre usuario invitado y usuario registrado.

Para el formulario de pago se hizo el componente PaymentForm, para este se utilizó PaymentElement de Stripe, que integraba Stripe en tu propio formulario.

3.6 Gestión de la lista de deseados

El objetivo de esta funcionalidad es que los usuarios tengan un espacio donde guardar prendas que les gusten para verlas más adelante sin miedo a que se borren.

Para la lista de deseados primero se hizo indiferentemente de si el usuario era invitado o no, pero después se pensó en hacer que esta funcionalidad solo estuviese disponible para usuarios registrados, y en caso que alguien quisiera añadir algo a la lista le saldría un modal para iniciar sesión

3.6.1 Backend

Para la lista de deseados se creó sólo una entidad:

Wishlist: Cuenta con una id (Long) generada automáticamente, con la id de la cesta a la que pertenece, una relación con el usuario que ha añadido el producto, y otra relación con el propio producto y la fecha en que se ha añadido.

También se hicieron los mappers, DTOs y un único service, controller y repository donde estarían los métodos que se ejecutan para crear,modificar, obtener o eliminar la cesta y el controlador para dirigir las peticiones.

3.6.2 Frontend

Para el frontend solo se tuvo que hacer la página de la lista de deseados, donde sí hay productos estos se muestran utilizando la ProductCard en formato tarjetas y si no la hay se muestra otra página indicando e invitando a buscar productos para añadir y/o comprar. También se hicieron los endpoints, donde se crearon las funciones necesarias para conectar el frontend con el backend. Desde esta parte se gestionan las peticiones para obtener la lista de deseados, añadir productos y eliminarlos.

3.7 Valoraciones y comentarios

El objetivo de esta funcionalidad es que los usuarios tengan la capacidad de valorar y comentar productos para ayudar a otros usuarios a escoger.

3.7.1 Backend

Para las reseñas se creó sólo una entidad:

Review: Cuenta con una id (Long) generada automáticamente, una relación con el usuario que realiza la valoración y una relación con el producto valorado. También almacena la puntuación mediante rating, el comentario escrito por el usuario y la fecha de creación de cuándo se ha hecho el comentario.

También se hicieron los mappers, DTOs y un único service, controller y repository donde estarían los métodos que se ejecutan para crear,modificar, obtener o eliminar la cesta y el controlador para dirigir las peticiones.

3.7.2 Frontend

Para la parte de frontend se crearon dos componentes principales. El primero es ReviewForm, que contiene el formulario para que el usuario pueda realizar la valoración del producto.

El segundo componente es ReviewSection, el encargado de mostrar las valoraciones del producto y también de mostrar el formulario de valoración (el ReviewForm). Este componente es el que se utiliza dentro de la página de cada producto.

3.8 Panel de control

El objetivo era incluir un panel de control para poder hacer las configuraciones de manera clara y rápida para poder obtener mejor rendimiento para la configuración de la página.

3.8.1 Backend

El panel de control requería en el backend un conjunto de endpoints exclusivos para administradores, protegidos a nivel de seguridad para que ningún usuario sin el rol adecuado pudiera acceder a ellos. Para ello se creó el `AdminController`, mapeado bajo `/api/admin`, que concentra todas las operaciones de administración: consulta de todos los usuarios registrados, consulta y actualización de pedidos, y una verificación extra de identidad antes de acceder al panel.

Spring Security se configuró en `SecurityConfig` para que todas las rutas bajo `/api/admin` (excepto `/api/admin/promote`, que es la ruta inicial para asignar el primer administrador mediante un secreto de entorno) exigieran el rol `ADMIN`. El sistema de roles se define como un enum con dos valores: `USER` y `ADMIN`, y se almacena directamente en la entidad `User` de la base de datos.

Para la verificación adicional de identidad dentro del panel, se implementó el endpoint `POST /api/admin/verify`, que recibe la contraseña del administrador autenticado y la comprueba contra el hash almacenado en base de datos usando BCrypt, añadiendo una capa de seguridad extra antes de conceder acceso a las secciones sensibles. La gestión de pedidos se realiza a través de `GET /api/admin/orders`, que devuelve todos los pedidos ordenados por fecha descendente, y `PATCH /api/admin/orders/{id}/status`, que permite actualizar el estado de un pedido siguiendo el flujo `PENDING → SHIPPED → IN_TRANSIT → DELIVERED`. El listado de usuarios se sirve mediante `GET /api/admin/users`, devolviendo la información de cada usuario mapeada a un `UserResponse` que excluye datos sensibles como la contraseña.

3.8.2 Frontend

En el frontend, el panel de administración se construyó como una sección separada bajo la ruta `/admin`, con un layout propio (`AdminLayout`) que actúa como barrera de acceso antes de renderizar cualquier contenido del panel. Este layout verifica en primer lugar que el usuario esté autenticado y que su rol sea `ADMIN` consultando el `AuthContext`; si no se cumplen estas condiciones, redirige automáticamente a la página de inicio o al login según corresponda.

Una vez confirmado el rol administrador, el layout muestra una pantalla intermedia de verificación de contraseña antes de dar acceso al panel. Esta pantalla envía la contraseña introducida al API Route de Next.js `/api/admin/verify`, que lee el token JWT de la cookie `httpOnly` del servidor sin exponerlo al cliente, y lo reenvía junto con la contraseña al backend de Spring Boot para su validación. Para reforzar la seguridad frente a ataques de fuerza bruta, se implementó un sistema de bloqueo progresivo: tras tres intentos fallidos el formulario se bloquea 15 segundos, y tras cinco intentos el bloqueo se extiende a 60 segundos.

La página principal del panel (`/admin/page.tsx`) muestra un menú de tarjetas que enlazan a cada sección disponible: productos, usuarios, pedidos, promociones, reportes y galería de fotos. La sección de usuarios (`/admin/usuarios`) obtiene el listado completo a través del API Route `/api/admin/usuarios` y lo presenta en una tabla con buscador en tiempo real, mostrando estadísticas de totales, número de administradores y usuarios con dirección guardada. La sección de pedidos (`/admin/pedidos`) muestra todos los pedidos con filtros por estado y buscador por nombre, email o ID, y permite avanzar el estado de cada pedido con un clic. Cada fila es expandible para mostrar una línea de tiempo visual del progreso del pedido, los datos del cliente y la dirección de envío. Todos los API Routes del panel leen el token de la cookie `httpOnly` del servidor y lo adjuntan como cabecera `Authorization` en las peticiones al backend, de modo que el token JWT nunca queda expuesto en el lado del cliente.

3.9 Ampliación

Este apartado está dedicado a las páginas y componentes realizados íntegramente con inteligencia artificial. Debido a la falta de tiempo para desarrollar todas las partes visuales manualmente, se decidió utilizar la IA como apoyo para acelerar el desarrollo y mejorar el aspecto general de la aplicación.

Esta sección no incluye funcionalidades complejas ni partes importantes de la lógica del proyecto, sino elementos principalmente visuales que ayudan a que la web se vea más completa y se acerque más a la apariencia de una tienda de ropa real, dándole un añadido real al proyecto.

3.9.1 Páginas del footer

Esta sección incluye todos los enlaces que hay en el footer, para hacer un footer más realista se añadieron un número de enlaces con la información habitual que hay en las tiendas online para el apartado legal, para ayudar al usuario y para dar a conocer tu marca.

Estas páginas son principalmente informativas y son generalmente texto, por lo que se utilizó la inteligencia artificial para que esos enlaces no quedasen vacíos, dándoles una función real y así dando una mejor experiencia al usuario. Gracias a la IA, cada página cuenta con un texto personalizado.

3.9.2 CategoryShowCase

CategoryShowCase es un componente utilizado para destacar diferentes categorías o secciones de productos, lo hace de forma visual por unas tarjetas que son principalmente una imagen, dándole a la web un toque más moderno y visual.

Este componente se implementa en el Home (el home no está hecho con IA), y en todas las secciones que aparecen en la barra de navegación.

3.9.3 Páginas de la barra de navegación

Esta sección incluye las diferentes páginas que se pueden acceder desde la barra de navegación, hombre, mujer, niños y SNKRS. La estructura de estas páginas es casi idéntica y hacen uso tanto de CategoryShowCase como de ProductSlider. Estas páginas son sólo para mostrar productos de forma visual y más atractiva para el usuario.

3.10 Despliegue

Para el despliegue de la aplicación se utilizaron diferentes plataformas dependiendo de cada parte del proyecto. El frontend desarrollado con Next.js se desplegó en Vercel, mientras que el backend desarrollado con Spring Boot y la base de datos PostgreSQL se desplegaron en Railway.

Durante el proceso fue necesario configurar variables de entorno y permitir la comunicación entre frontend y backend mediante CORS.

4. Conclusiones

En esta sección se hará una reflexión en retrospectiva para ver el resultado del desarrollo del proyecto, analizando como se ha llevado a cabo el proyecto, tanto con los aspectos positivos como aquellos a mejorar.

4.1 Conclusiones generales

El desarrollo de KŌRA ha sido un desarrollo que ha permitido crear un e-commerce de ropa funcional, mezclando frontend y backend para crear esta aplicación web. A lo largo del desarrollo se han ido implementando funcionalidades que tienen las tiendas online reales, todo buscando ofrecer una experiencia intuitiva, simple y moderna.

Durante el proyecto se han trabajado muchos conocimientos aprendidos a lo largo del curso y se ha aprendido otros para poder completarlo. También se ha trabajado otras aptitudes no relacionadas con la programación, como la organización, la comunicación y la toma de decisiones.

A nivel general se ha creado una aplicación web funcional, por lo que se está contento, ya que se ha dado el 100% en el desarrollo del proyecto para que saliese adelante y se ha trabajado al máximo para conseguir el mejor resultado posible, por lo que se piensa que el desarrollo de la aplicación ha sido un éxito.

4.2 Consecución de objetivos

A continuación se revisará si se han conseguido los objetivos establecidos a principio de desarrollo del proyecto:

Catálogo de productos: Se ha creado correctamente un catálogo de productos, con categorías, un filtro funcional, una página de producto con su información detallada y actualización automática del stock en tiempo real. Así que este objetivo está 100% logrado

Gestión de usuarios: Se ha logrado una gestión de usuarios funcional, con registro funcional, sistema de recuperación de contraseña, edición de perfil y cada usuario con un historial de pedidos, incluso se ha logrado hacer más de lo

que se buscaba , como la verificación 2FA. Por lo que este objetivo se ha cumplido correctamente.

Gestión de cesta y productos: Se ha logrado una gestión de usuarios funcional, con cálculo del precio automáticamente, se puede gestionar la cesta correctamente y da paso a la facturación del pedido correctamente, por lo que este objetivo ha sido logrado.

Facturación del pedido: Se ha logrado integrar una pasarela de pago funcional mediante Stripe, permitiendo realizar pagos con tarjeta bancaria y verificar correctamente las transacciones antes de confirmar el pedido. También se muestra una página de confirmación tras completar la compra. Sin embargo, no se han implementado métodos adicionales como PayPal, sino Amazon Payy se ha logrado enviar de forma automática de facturas en PDF o al mail, por lo que este objetivo se ha cumplido totalmente.

Envío: Se ha implementado un formulario de dirección de envío funcional dentro del checkout, permitiendo guardar los datos necesarios para realizar el pedido. También se muestra una estimación visual de entrega, que obviamente no funciona con un envío real, pero esta tiene diferentes estados que se van actualizando automáticamente al pasar el tiempo o al manualmente el administrador cambiarlos, por lo que este objetivo se puede dar por cumplido totalmente.

Valoraciones y comentarios: Se ha conseguido implementar un sistema funcional de valoraciones y comentarios, permitiendo que los usuarios puedan puntuar productos y dejar opiniones visibles para otros usuarios. Además, se evita que un mismo usuario pueda valorar varias veces el mismo producto, por lo que este objetivo se ha cumplido correctamente.

Promociones y descuentos: Actualmente no se ha implementado un sistema de promociones, cupones o recomendaciones personalizadas. Tampoco existe una suscripción a la newsletter. Aunque son funcionalidades planteadas para futuras versiones, este objetivo no se ha llegado a desarrollar.

Panel de control: No se ha desarrollado un panel de administración completo. La idea inicial era implementarlo utilizando métricas cosa que al final no se ha implementado, por lo que este objetivo se ha cumplido parcialmente

Tabla interactiva: La tabla interactiva no se ha realizado por falta de tiempo, así que este objetivo no se ha cumplido.

Rendimiento: La aplicación ofrece tiempos de carga rápidos y una navegación fluida gracias al uso de Next.js y a la optimización de componentes e imágenes. Además, la separación entre frontend y backend ayuda a mantener un funcionamiento estable, por lo que este objetivo se ha cumplido correctamente.

Usabilidad: Se ha buscado mantener una interfaz limpia, intuitiva y sencilla de utilizar, intentando que el proceso de compra sea cómodo y fácil para cualquier usuario. También se ha mantenido una coherencia visual entre todas las páginas de la aplicación, por lo que este objetivo se ha cumplido correctamente.

Disponibilidad: La aplicación está preparada para funcionar de forma continua al tratarse de una aplicación web desplegable online. Sin embargo, no se han implementado sistemas avanzados de monitorización o recuperación ante caídas, por lo que este objetivo se ha cumplido parcialmente.

Escalabilidad: La estructura del proyecto se ha desarrollado pensando en futuras ampliaciones, utilizando separación entre frontend y backend, componentes reutilizables y una arquitectura organizada
, por lo que este objetivo se ha cumplido correctamente.

Seguridad y autenticación: Se ha implementado autenticación mediante JWT, encriptación de contraseñas con BCrypt y conexiones seguras mediante HTTPS en producción. También se añadió verificación 2FA y autenticación mediante Google, por lo que este objetivo se ha cumplido correctamente.

Mantenibilidad: El proyecto se ha desarrollado manteniendo una estructura organizada y separada por responsabilidades, utilizando componentes reutilizables, DTOs, mappers y separación entre frontend y backend. Esto facilita corregir errores y añadir nuevas funcionalidades, por lo que este objetivo se ha cumplido correctamente.

Robustez: La aplicación se adapta correctamente a diferentes tamaños de pantalla y dispositivos gracias al diseño responsive implementado, por lo que este objetivo se ha cumplido correctamente.

4.3 Valoración de la metodología y planificación

En un principio se comenzó con una buena planificación y buena metodología de trabajo, se trabajaba en base a features y se utilizaba un tablero Kanban en el propio repositorio de GitLab que permite ordenarse en base a diferentes estados de las tarjetas.

Pero en cuanto menos tiempo quedaba más se comenzó a descuidar este aspecto y más se comenzó a ordenarse en base a mensajes directos y notas físicas. Este cambio fue creciendo, todo apremiado por la sensación de falta de tiempo y por esa misma sensación de tener que hacer más en menos tiempo.

Hubiese sido conveniente mantener el uso de Kanban durante todo el desarrollo, aunque fuese simplificando un poco su uso.

4.4 Visión de futuro

El proyecto tiene una clara visión de futuro con una gran cantidad de implementaciones y mejoras para ampliar el alcance de la web.

Entre las mejoras se podrían incluir un sistema de recomendaciones personalizadas, en base a las búsquedas y las compras de cada usuario. También sería interesante mejores métricas, guardando información de cada producto como las veces agregadas al carrito, las compras de ese producto en concreto, en cuantas listas de deseados ha estado/está, permitiendo un mejor seguimiento de lo que funciona y de lo que no funciona.

Otra gran implementación sería traducir la página para diferentes idiomas mediante i18n. Además se podría añadir un cuestionario para enviar un mensaje en caso de necesitar ayuda, pedir una devolución o otro motivo para su revisión posterior.

Este tipo de proyecto siempre tiene capacidad de mejora y evolución, pudiendo adaptarse a las necesidades del mercado y a las demandas de los usuarios en cada momento.

5. GLOSARIO

API REST: Arquitectura que permite la comunicación entre frontend y backend mediante peticiones HTTP.

Backend: Parte de la aplicación encargada de la lógica de negocio, la seguridad y la comunicación con la base de datos. Procesa las peticiones del frontend y devuelve la información necesaria.

BCrypt: Sistema utilizado para encriptar contraseñas de forma segura antes de guardarlas en la base de datos. Permite que las contraseñas no se almacenen en texto plano.

Controller: Capa del backend encargada de recibir las peticiones HTTP y dirigirlas al servicio correspondiente. También devuelve la respuesta al frontend.

CRUD: Siglas de Create, Read, Update y Delete. Hace referencia a las operaciones básicas sobre datos: crear, leer, actualizar y eliminar.

DTO: Siglas de Data Transfer Object. Es un objeto utilizado para transferir datos entre backend y frontend, evitando exponer información sensible o innecesaria.

Endpoint: Ruta de la API que permite realizar una acción concreta, como obtener productos, iniciar sesión, añadir un producto a la cesta o realizar un pago.

Entidad: Clase que representa una tabla de la base de datos. Sus atributos suelen corresponderse con las columnas de dicha tabla.

Frontend: Parte visual de la aplicación con la que interactúa el usuario. Incluye las páginas, botones, formularios, menús y todos los elementos que permiten navegar y utilizar la web.

Fullstack: Tipo de desarrollo que incluye tanto frontend como backend.

Hibernate: Herramienta utilizada para mapear entidades Java con tablas de la base de datos. Facilita trabajar con bases de datos relacionales desde código Java.

HTTP: Siglas de HyperText Transfer Protocol. Es el protocolo utilizado para la comunicación entre cliente y servidor mediante peticiones y respuestas.

JWT: Siglas de JSON Web Token. Es un token utilizado para mantener la autenticación del usuario en la aplicación y permitir el acceso a rutas protegidas.

Kanban: Método de organización basado en tarjetas y estados de trabajo. Permite visualizar tareas pendientes, en proceso y completadas.

Mapper: Clase encargada de transformar entidades en DTOs y viceversa. Ayuda a separar los datos internos de la aplicación de los datos que se envían al frontend.

MVC: Siglas de Model, View, Controller. Es un patrón de arquitectura que separa la aplicación en modelo, vista y controlador para organizar mejor el código.

Repository: Capa encargada de comunicarse con la base de datos. Permite realizar consultas y operaciones sobre las entidades del proyecto.

Responsive: Diseño adaptable a diferentes tamaños de pantalla y dispositivos. Permite que la web se vea correctamente en ordenador, tablet o móvil.

Service: Capa donde se encuentra la lógica principal de negocio. Se encarga de procesar los datos antes de devolverlos al controlador o guardarlos en la base de datos.

Stripe: Plataforma utilizada para gestionar pagos online. En el proyecto se utiliza para simular pagos mediante tarjeta en modo de prueba.

Token: Cadena de texto utilizada para identificar o validar una sesión.

6. Bibliografía

React Documentation. Disponible en: https://react.dev/
Next.js Documentation. Disponible en: https://nextjs.org/docs
Spring Boot Documentation. Disponible en: https://spring.io/projects/spring-boot
Spring Security Documentation. Disponible en: https://spring.io/projects/spring-security
PostgreSQL Documentation. Disponible en: https://www.postgresql.org/docs/
Stripe Documentation. Disponible en: https://docs.stripe.com/
Tailwind CSS Documentation. Disponible en: https://tailwindcss.com/docs
JWT Introduction. Disponible en: https://jwt.io/introduction
Hibernate ORM Documentation. Disponible en: https://hibernate.org/orm/documentation/
MDN Web Docs. Disponible en: https://developer.mozilla.org/
TypeScript Documentation. Disponible en: https://www.typescriptlang.org/docs/
Git Documentation. Disponible en: https://git-scm.com/doc
Vercel Documentation. Disponible en: https://vercel.com/docs
Railway Documentation. Disponible en: https://docs.railway.com/
Stack Overflow. Disponible en: https://stackoverflow.com/
W3Schools. Disponible en: https://www.w3schools.com/
Google Fonts. Disponible en: https://fonts.google.com/
Swiper Documentation. Disponible en: https://swiperjs.com/
js-cookie Documentation. Disponible en: https://github.com/js-cookie/js-cookie

7. ANEXOS

Stripe Testing Cards. Disponible en:
https://docs.stripe.com/testing?testing-method=card-numbers#cards