POWERFUEL

IES LUIS BRAILLE
2º DAW

Autores: Adrián García Torrente y Adrián Escribano Pérez

Índice

Índice de ilustraciones

RESUMEN

PowerFuel es un proyecto el cual su desarrollo está orientado para la venta de alimentos saludables para deportistas la cual consta de una aplicación para el Backend, está va a llevar a cabo toda la funcionalidad de gestionar toda la información sobre los productos y usuarios; una aplicación para el Frontend, está se va a encargar de mostrar la información y aportar funcionalidades a los usuarios, ya sea para la compra de productos como para la gestión de productos y usuarios; una aplicación externa para la gestión de compras, más a delante se desarrolla la razón por la que optamos por utilizar un servicio externo y porque esa en concreto; y por ultimo una aplicación de Base de Datos, está se va a encargar de almacenar casi todos tanto de los productos como de los usuarios.

Durante el desarrollo de este proyecto, hemos tenido que afrontar muchos contratiempos y sobre todo la gestión del tiempo para afrontar un proyecto de estas características. Durante el desarrollo de este conjunto de aplicaciones hemos tenido que afrontar el aprendizaje de muchas tecnologías para poder desempeñar correctamente las funcionalidades principales del proyecto, así como el uso de estas tecnologías y la anexión de estas juntas en el proyecto.

Las tecnologías más importantes aprendidas en este proyectos son, para el Frontend hemos usado el framework de Next en conjunto con React para el desarrollo de aplicación que se va a encargar de mostrar el contenido a los usuarios, para el Backend hemos utilizado el framework de ExpressJS con NodeJS el cual se encarga de gestionar las peticiones del servidor de Frontend, para el almacenamiento datos relevantes he imprescindibles para alguna de las funcionalidades hemos usado una base de datos basada en MySQL, para la gestión de las transacciones usamos la aplicación externa Stripe, para el diseño de la aplicación hemos utilizado TailwindCSS.

Este proyecto se pretende que tenga las siguientes funcionalidades principales de la aplicación, compra de productos por parte de usuarios estándar, administración de productos, registro de pedidos, registro de usuarios, registro de productos, administración de usuarios, gestión de roles de los usuarios, gestión de categorías de usuarios, adaptación de la página para cualquier dispositivo y gestión de pedidos por parte de los usuarios con roles específicos.

Gracias al desarrollo de este proyecto, hemos aprendido como es el desarrollo de una aplicación de esta magnitud, el desarrollo de funcionalidades para la aplicación, la gestión de permisos, nuevas tecnologías como NodeJS y React para el desarrollo de aplicaciones, el uso y gestión de librerías, gestión de repositorios git, despliegue y pruebas de la aplicación para eliminar posibles bugs y hacer que la experiencia del usuario sea la mejor, desarrollo en un entorno de equipo y así como el consenso de ideas y funcionalidades del equipo.

Este proyecto está almacenado en un repositorio público de Github para que cualquier persona pueda desarrollar en base a nuestra aplicación nuevas funcionalidades, así como poder usar este proyecto. Este repositorio se encuentra en PowerFuel.

ABSTRACT

PowerFuel is a project whose development is oriented to the sale of healthy food for athletes which consists of an application for the Backend, it will carry out all the functionality to manage all the information about the products and users; an application for the Frontend, it will be responsible for displaying the information and provide functionality to users, either for the purchase of products and for the management of products and users; an external application for the purchase management, the reason why we chose to use an external service and why that one in particular is developed further on; and finally a database application, which will be in charge of storing almost all the products as well as the users.

During the development of this project, we have had to face many setbacks and above all the time management to deal with a project of these characteristics. During the development of this set of applications we have had to face the learning of many technologies to be able to perform correctly the main functionalities of the project, as well as the use of these technologies and the annexation of these together in the project.

The most important technologies learned in this project are, for the Frontend we have used the Next framework in conjunction with React for the development of the application that will be responsible for displaying the content to users, for the Backend we have used the ExpressJS framework with NodeJS which is responsible for managing the requests from the Frontend server, for the storage of relevant data and essential for some of the features we have used a database based on MySQL, for the management of transactions we use the external application Stripe, for the design of the application we have used TailwindCS.

This project is intended to have the following main functionalities of the application, purchase of products by standard users, product management, order registration, user registration, product registration, user administration, user role management, user category management, page adaptation for any device and order management by users with specific roles.

Thanks to the development of this project, we have learned how is the development of an application of this magnitude, the development of functionalities for the application, permissions management, new technologies such as NodeJS and React for the

development of applications, the use and management of libraries, git repository management, deployment and testing of the application to eliminate possible bugs and make the user experience the best, development in a team environment and the consensus of ideas and functionalities of the team.

This project is stored in a public Github repository so that anyone can develop new functionalities based on our application, as well as be able to use this project. This repository is located in PowerFuel.

INTRODUCCIÓN

En un mundo cada vez más consciente de la importancia de la nutrición y el bienestar, el mercado de la alimentación y la suplementación alimentaria ha experimentado un crecimiento sin precedentes en estos últimos años.

Con la creciente demanda de productos que complementen y proporcionen la dieta alimentaria diaria, surge la necesidad de plataformas online que faciliten el acceso al usuario a una amplia gama de alimentos y suplementos nutricionales de calidad.

En este contexto, nace la idea del desarrollo de un proyecto que cree una plataforma de comercio electrónico dedicada exclusivamente en la venta de productos de suplementación.

El proyecto de PowerFuel nace como respuesta a la creciente demanda de una experiencia de compra conveniente y segura para aquellos que buscan mejorar su salud y rendimiento a través de la alimentación y la suplementación alimentaria para mantener una dieta equilibrada y saludables.

Somos conscientes de los desafíos que tenemos que afrontar y que nos van a acompañar durante el desarrollo de este proyecto, tales como el desarrollo de una aplicación adaptada a cualquier dispositivo, sea PC, Tablet o teléfono móvil, mantener la seguridad de los usuarios durante el uso de la aplicación y la confidencialidad de los datos de los usuarios. Sobre todo, desde PowerFuel, queremos garantizar que:

1. **Accesibilidad y Usabilidad**: La plataforma debe ser accesible y fácil de usar desde cualquier dispositivo. Esto implica un diseño responsivo que asegure una experiencia de usuario óptima tanto en ordenadores de escritorio como en dispositivos móviles.
2. **Seguridad y Confidencialidad**: La seguridad de los usuarios es una prioridad. Implementaremos medidas de seguridad robustas para proteger los datos personales y la información de pago de los usuarios. Esto incluye el uso de encriptación avanzada y protocolos de seguridad para garantizar que los datos se mantengan confidenciales y seguros en todo momento.

3. **Calidad y Variedad de Productos:** Nos comprometemos a ofrecer una amplia gama de productos de alta calidad. Esto incluye la selección de proveedores confiables y la garantía de que todos los productos cumplan con los estándares más estrictos de calidad y seguridad alimentaria.
4. **Experiencia de Compra Fluida**: Queremos que la experiencia de compra en PowerFuel sea lo más fluida y agradable posible. Esto incluye una navegación intuitiva, procesos de compra rápidos y sencillos.

PowerFuel no solo es una tienda en línea, sino una plataforma integral que busca mejorar la salud y el rendimiento de sus usuarios a través de la nutrición. Con una infraestructura tecnológica sólida y un enfoque centrado en el usuario, estamos preparados para enfrentar los desafíos y aprovechar las oportunidades que presenta este mercado en crecimiento. Nos comprometemos a ser un aliado confiable y valioso para todos aquellos que buscan alcanzar sus objetivos de bienestar y rendimiento deportivo.

OBJETIVOS

El proyecto de la aplicación PowerFuel, nos ponemos como objetivos principales el desarrollo y el lanzamiento de una plataforma de comercio electrónico que satisfaga las necesidades que demandan los usuarios y las expectativas de los consumidores interesados en una alimentación saludable.

Para poder proporcionar todo esto, implica la creación de una interfaz intuitiva y fácil de usar que este adaptada a los diferentes dispositivos del mercado, la selección cuidadosa de los productos y marcas que vamos a introducir en el proyecto, la implementación de medidas de seguridad para proteger la información de los usuarios y la prestación de servicios como la compra de productos y la gestión de la tienda electrónica.

A continuación, exponemos todos los objetivos previstos para el desarrollo de la aplicación, empezando desde los más básicos hasta los más complejos pasando por puntos como el aprendizaje de tecnologías, desarrollo de la base de datos, etc.

1. Aprendizaje de las tecnologías básicas del proyecto, como NodeJS, ExpressJS, React y Next.
2. Explorar las páginas de nuestros competidores para poder realizar una mejor interfaz cogiendo las mejores ideas y así poder desarrollar las nuestras propias.
3. Definir un proceso de registro e inicio de sesión de los usuarios para poder gestionarlos de una manera eficiente.
4. Definir un proceso de gestión de registro y asignación de roles de usuario para el uso de la aplicación y acceso a diferentes apartados de la página.
5. Definir un proceso de categorización de los productos para gestionarlos por la página.
6. Realizar un sistema que permita el registro de productos dentro de la web y un apartado de administración para estos.
7. Sistema de pago de la página lo más eficiente, intuitivo y adaptado para la gestión de las transacciones.
8. Permitir el guardado de imágenes de los productos.
9. Adaptar y asegurar un buen funcionamiento de la página en los diferentes dispositivos.
10. Permitir que los usuarios puedan escribir incidencias a través de un formulario.

ALCANCE DEL PROYECTO

I. REQUISITOS FUNCIONALES Y NO FUNCIONALES DEL PROYECTO

1. Requisitos Funcionales

1. Diseñar un formulario de registro de los usuarios.
2. Diseñar un sistema de autenticación de los usuarios a través de su correo electrónico y su contraseña.
3. Menú en el que se muestran las categorías principales para poder seleccionar de una forma rápida y sencilla por las diferentes categorías de los productos.
4. Buscador de productos para encontrar los productos de una manera rápida y sencilla.
5. Mostrar diferentes secciones en el Home para que el usuario pueda navegar por categorías aleatorias.
6. Panel de configuración del usuario.
7. Formulario de cambio de contraseña del usuario.
8. Panel de notificaciones donde podrá ver todas las notificaciones que iremos creando en las diferentes funciones que haga el usuario.
9. Carrito el cual podrá el usuario ir viendo que va añadiendo a la compra.
10. Panel de pedidos del usuario en el cual se podrá ver un historial de los pedidos con toda la información del pedido.
11. Panel de administración de la página en el que tendremos varios apartados y una página principal en la que podremos ver información de la página y del servidor.
12. Apartado de gestión de usuarios de la página donde se podrá añadir, ver y gestionar los usuarios y poder editar cualquier dato del usuario menos la contraseña, incluso activarlos o desactivarlos para permitir o no el acceso a la cuenta.
13. Apartado de gestión de productos de la página donde se podrá añadir, ver y gestionar los productos de la página.
14. Apartado de gestión de roles donde se podrá crear o eliminar los roles de los usuarios.
15. Apartado de gestión de categorías donde se podrá crear o eliminar las categorías de los productos.

16. Formulario para que los usuarios puedan mandar incidencias.
17. Posibilidad de cambiar los colores de la página con dos modos.

2. Requisitos no funcionales

1. Hay que asegurar que la página web sea compatible con una amplia gama de dispositivos (computadoras de escritorio, portátiles, móviles, tabletas) y navegadores webs populares (Chrome, Firefox, Safari, etc.). La interfaz debe ser amigable con el usuario.
2. Los usuarios se podrán registrar sin necesidad de que los administradores los estén registrando.
3. Los datos de información de usuario, productos, categorías, roles, etc.… serán almacenado en la base de datos.
4. El código estará comentado para su mantenimiento.
5. El código será lo más eficiente posible.
6. Optimizar el rendimiento del sitio web para que las páginas se carguen rápidamente, mejorando así la experiencia del usuario y el posicionamiento en buscadores.
7. Diseñar la infraestructura de la página web de manera que pueda manejar un aumento en el tráfico y el volumen de transacciones a medida que crece el negocio.
8. Garantizar que la página web sea accesible para personas con discapacidades, cumpliendo con las pautas de accesibilidad web establecidas por el consorcio W3C.
9. Asegurarse de que la página web cumpla con todas las leyes y regulaciones aplicables en materia de comercio electrónico, privacidad de datos, protección al consumidor, etc.

II. MOCKUP (PROTOTIPO)

1. Página de inicio

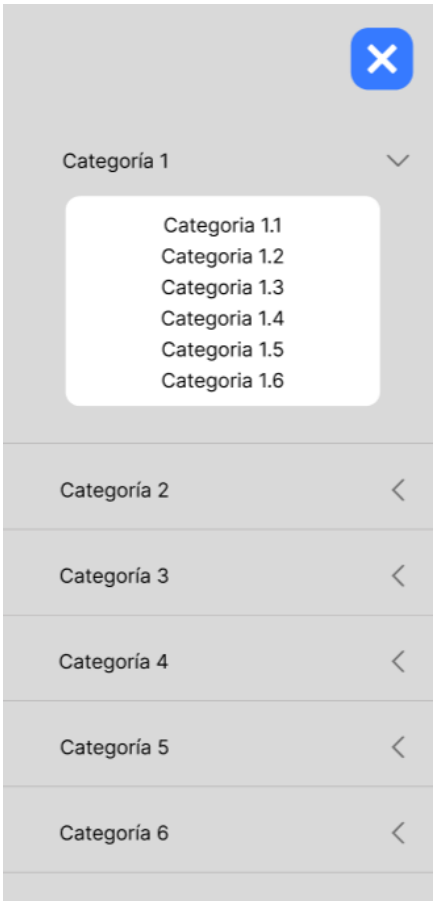

Ilustración 1 Menú de la izquierda de la página

Ilustración 2 Home de la pagina

Alumnos: Adrián García Torrente y Adrián Escribano Pérez
Curso: 2º DAW
ies.luisbraille.coslada@educa.madrid.org
www.iesluisbraille.es

2. Barra de navegación

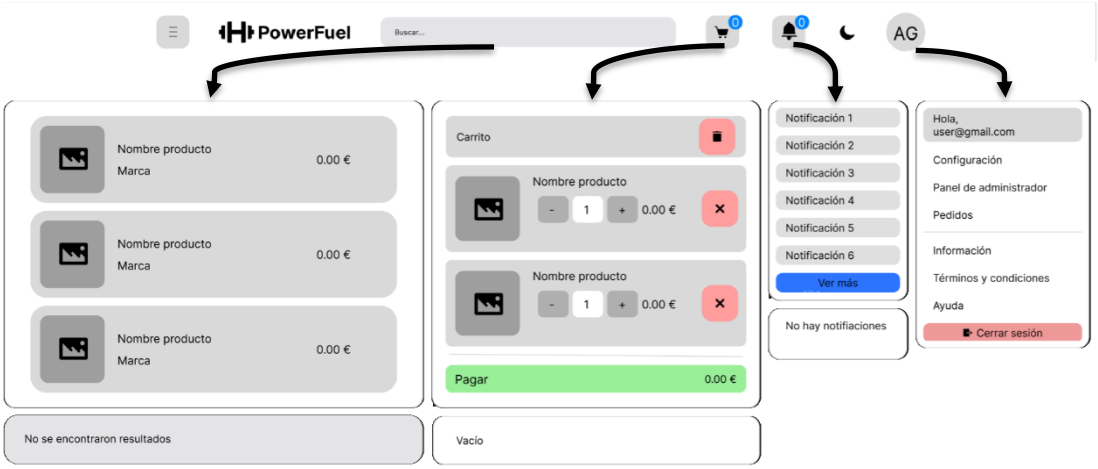

Ilustración 3 Barra de navegación

3. Menú de autenticación

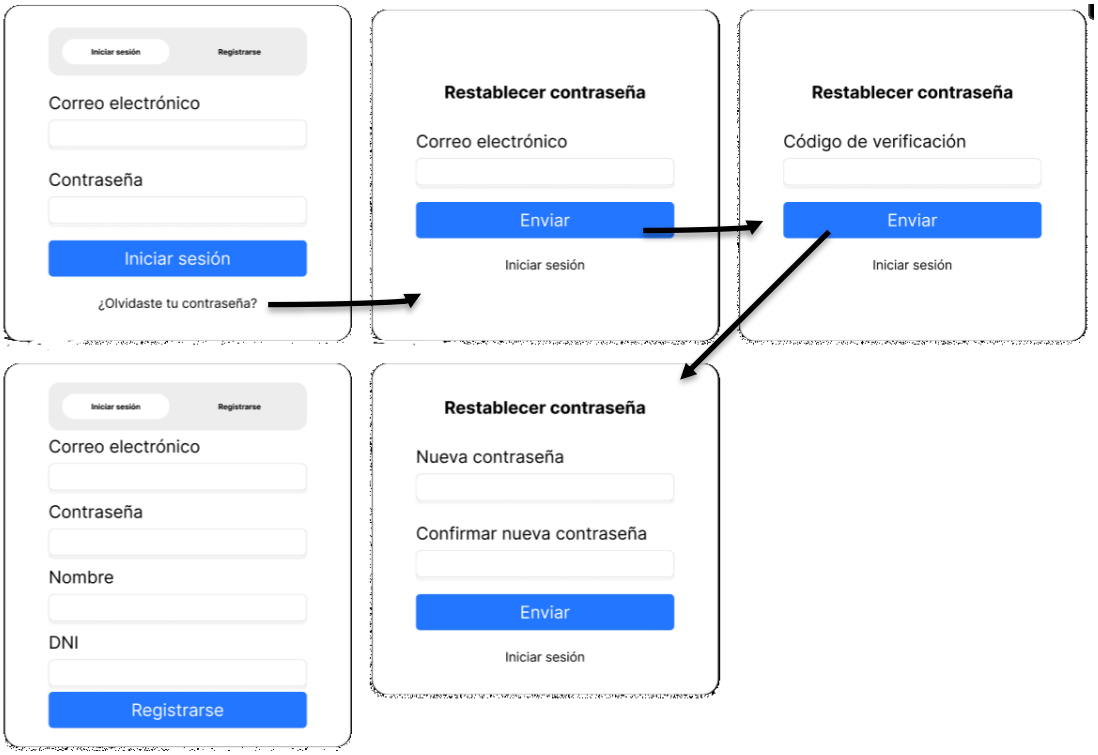

Ilustración 4 Menú de autenticación

4. Panel de usuario

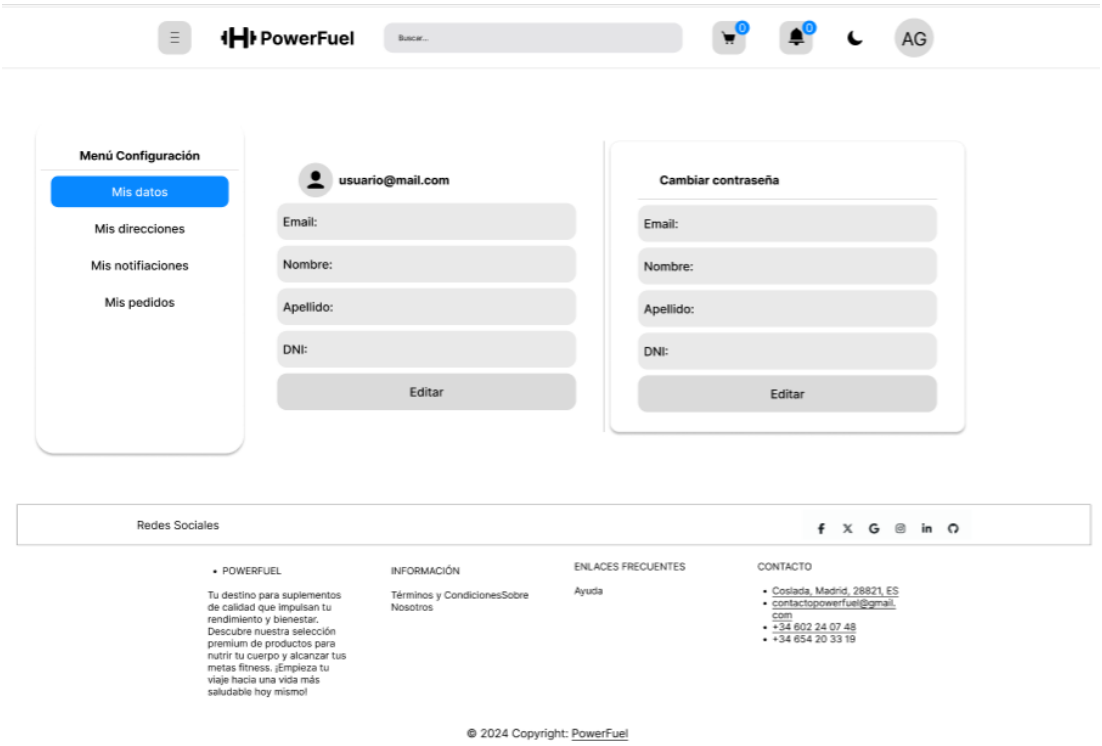

Ilustración 5 Menú de datos de usuarios

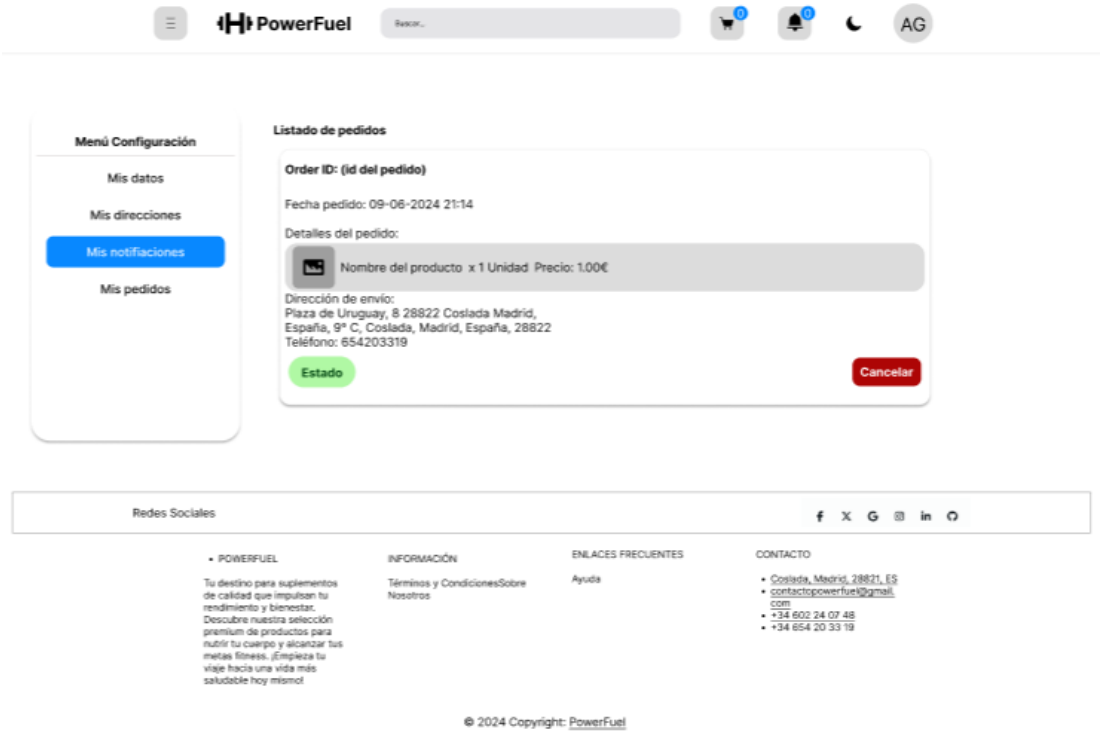

Ilustración 6 Menú de pedidos

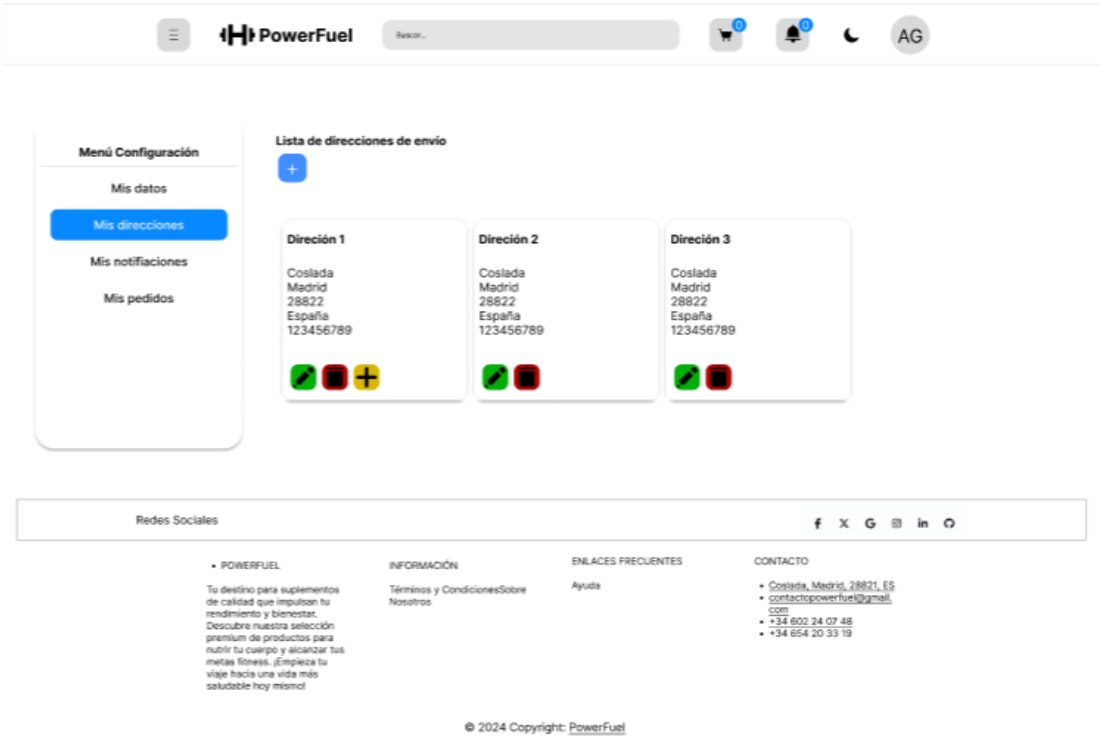

Ilustración 7 Menú de direcciones de envió del usuario

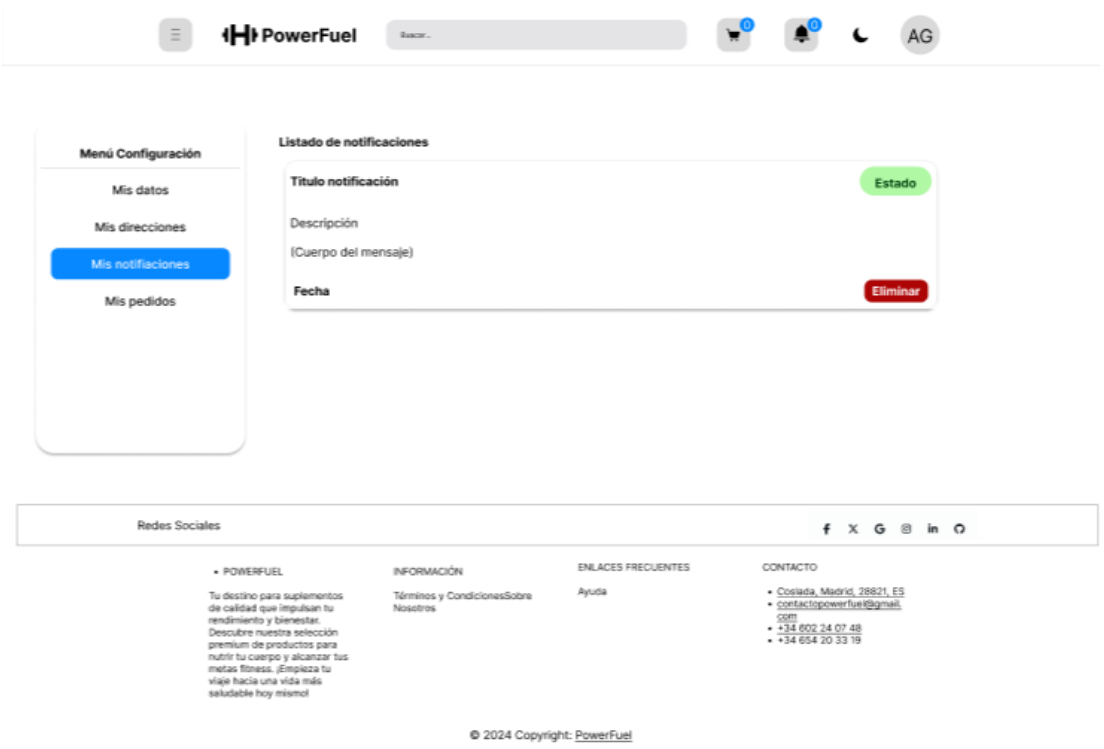

Ilustración 8 Menú de notificaciones del usuario

5. Panel de administración

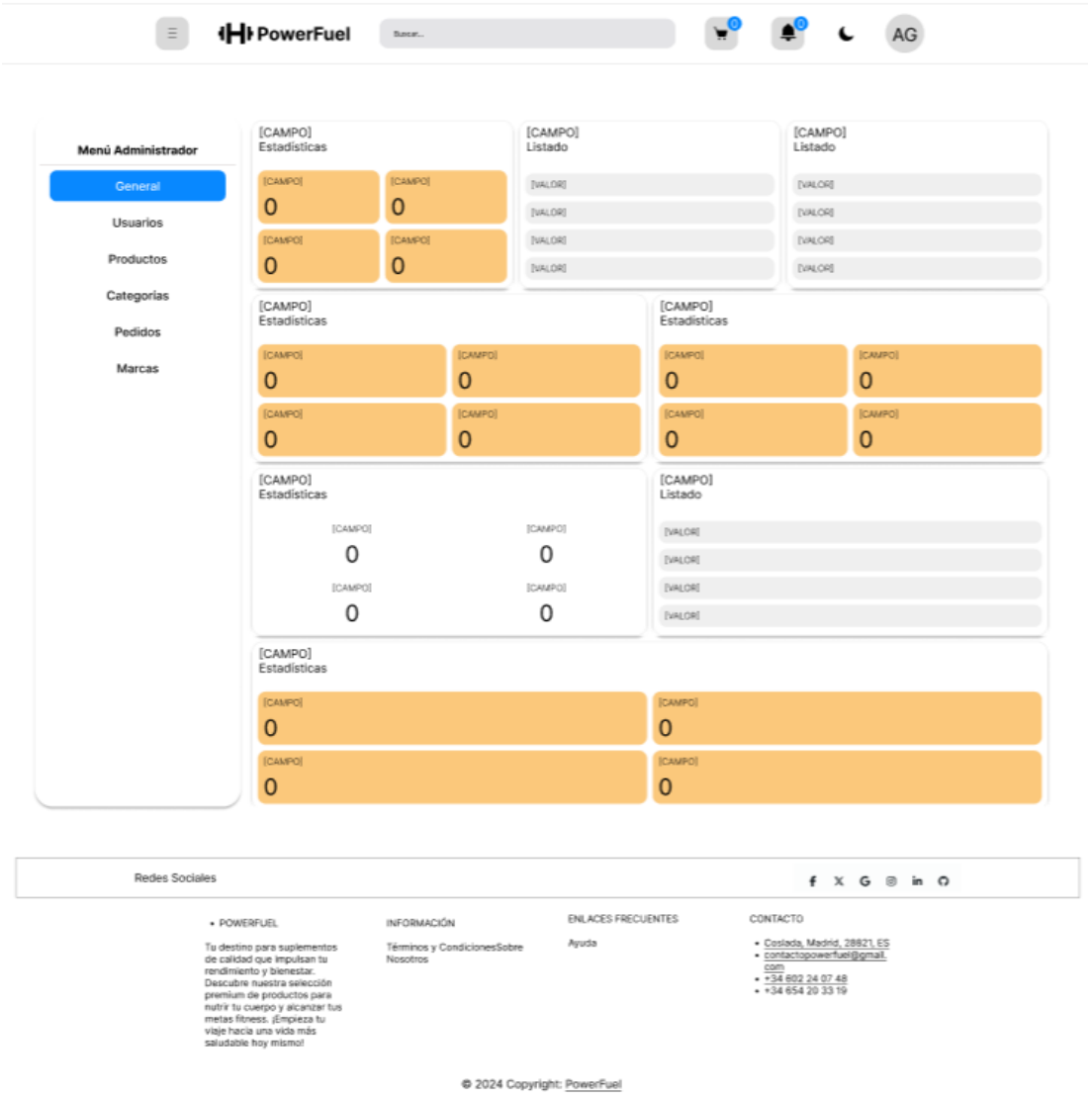

Ilustración 9 Menú de administración de la página

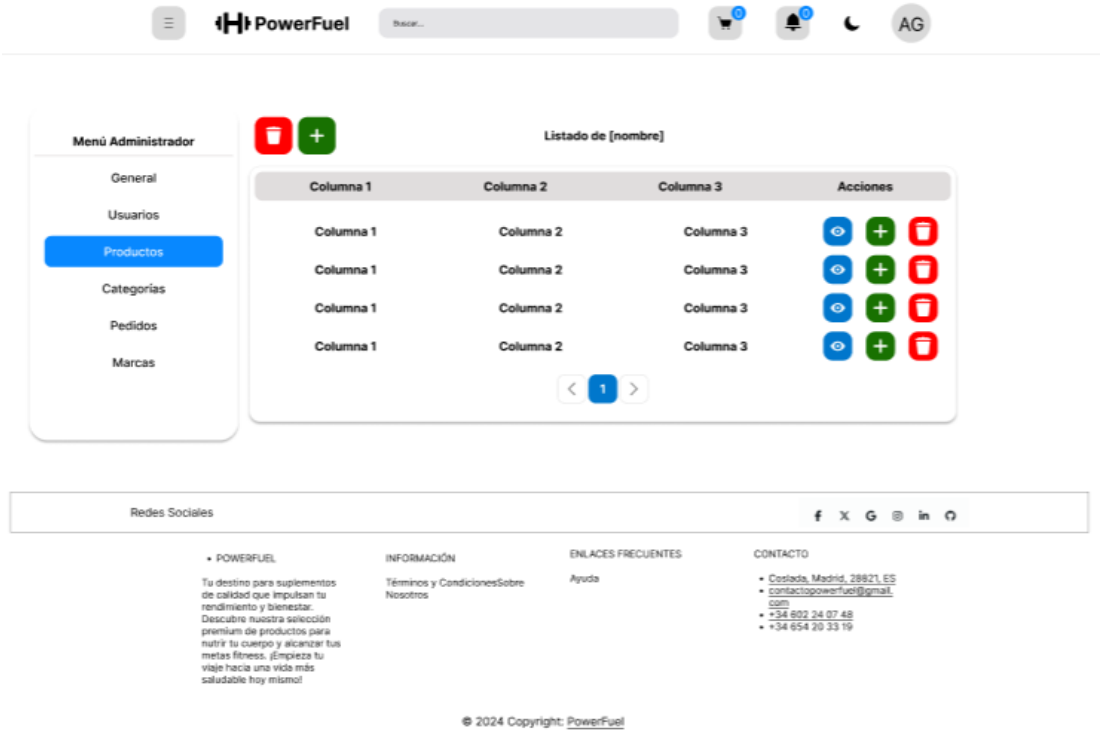

Ilustración 10 Menús de administración de la página

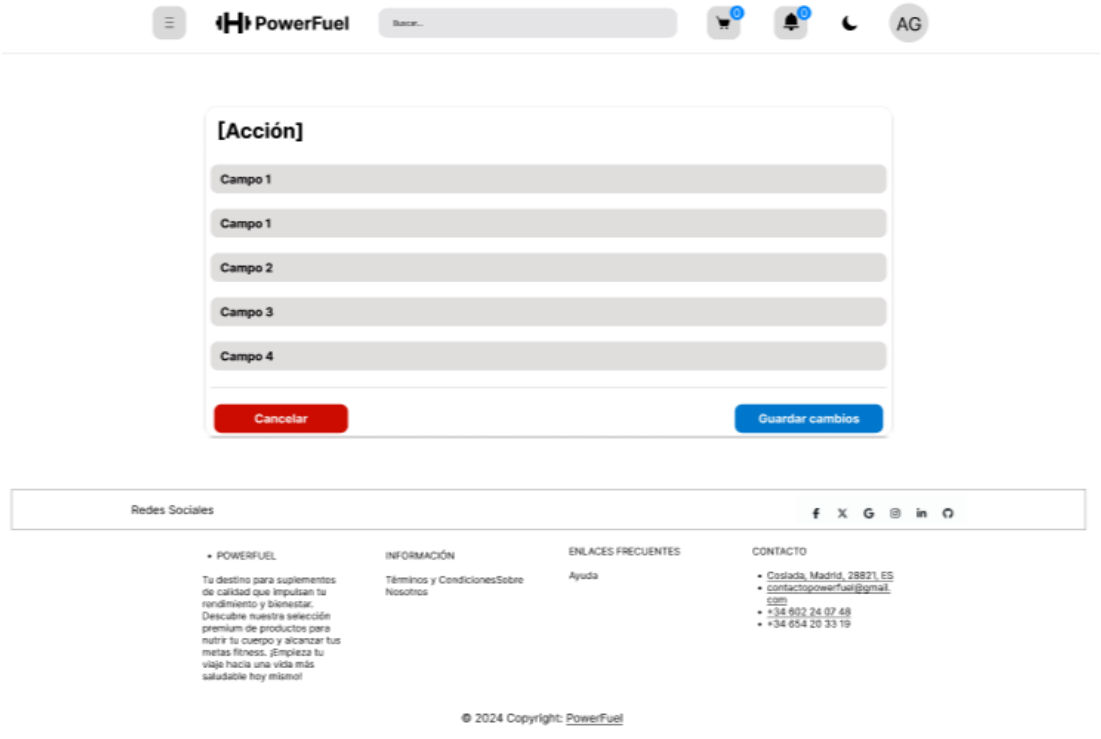

Ilustración 11 Menús de acciones de administración

6. Página de producto

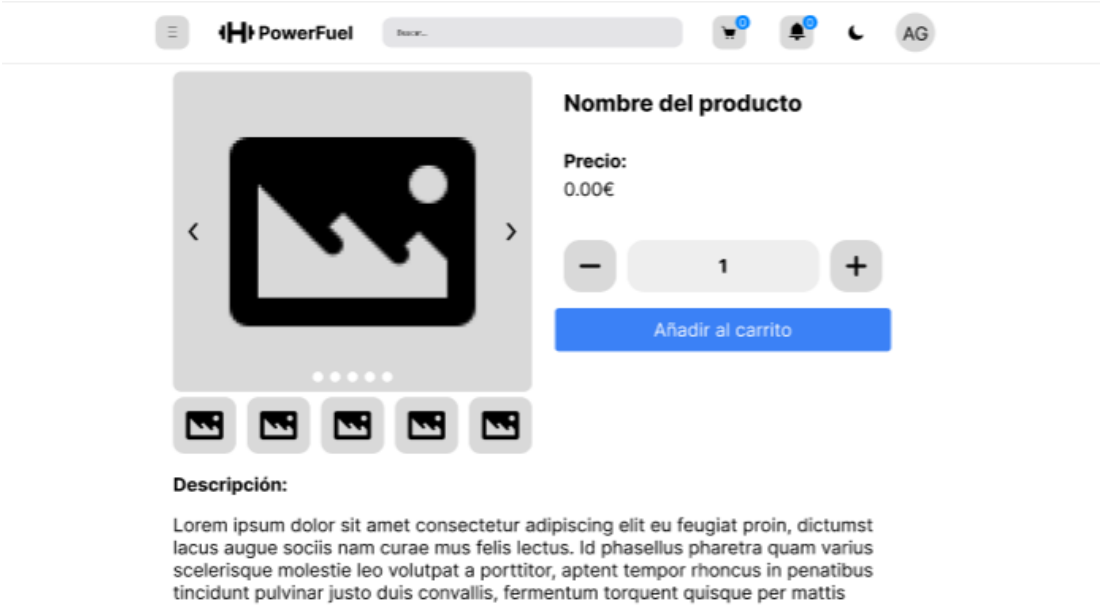

Ilustración 12 Página del producto

7. Página de categorías y resultado de búsqueda

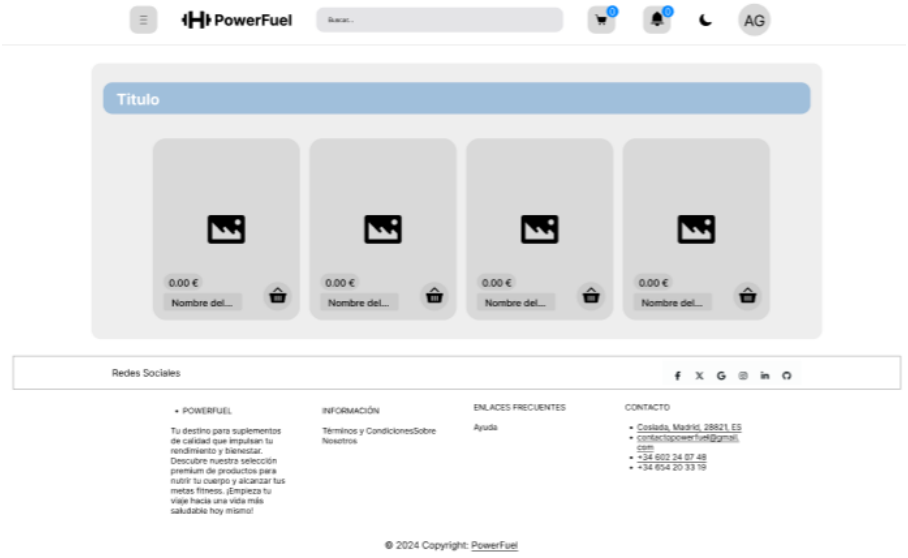

Ilustración 13 Página de categorias y resultados de busqueda

III. TECNOLOGIAS USADAS

En este aparatado de la documentación de este proyecto, vamos a exponer la gran mayoría de las tecnologías que utilizamos en el proyecto para poder realizar todas las funcionalidades que tiene el proyecto. También vamos a exponer los motivos por los cuales hemos decidido utilizar cada una de las tecnologías empleadas.

1. Visual Studio Code

Visual Studio Code es un editor de código fuente gratuito y de código abierto desarrollado por Microsoft, popular por su rendimiento y versatilidad. Funciona en Windows, macOS y Linux, ofreciendo una experiencia uniforme en todas las plataformas. VS Code es conocido por ser ligero y rápido. Además, incluye herramientas integradas como depuración, control de versiones con Git y un terminal integrado, lo que permite realizar múltiples tareas sin necesidad de cambiar de aplicación. Su extensibilidad con un Marketplace lleno de extensiones que añaden funcionalidades adicionales y soporte para más lenguajes y herramientas de desarrollo.

Ilustración 14 Logo de Visual Studio Code.

Ilustración 15 Logo de LiveShare

Hemos decidido utilizar VS Code para poder utilizar una extensión que permite el trabajo colaborativo, esta extensión se llama Live Share y permite la colaboración desde un entorno de diseño a la vez.

2. Figma

Figma es una aplicación de diseño de interfaces de usuario web que ofrece herramientas intuitivas y una colaboración en tiempo real, lo que la convierte en una opción popular para equipos de diseño de todo el mundo.

Ilustración 16 Logo de Figma.

Figma ofrece un conjunto completo de herramientas de diseño de interfaces de usuario, que incluyen herramientas de diseño vectorial, herramientas de prototipado interactivo y herramientas de colaboración en tiempo real. Usuarios pueden crear diseños de alta calidad, prototipos interactivos y documentación de diseño, todo dentro de la misma plataforma.

Varios usuarios pueden editar un diseño simultáneamente, lo que permite a los equipos trabajar juntos de manera eficiente y ver los cambios en tiempo real. Esto elimina la necesidad de enviar archivos de diseño por correo electrónico o sincronizar archivos manualmente, lo que ahorra tiempo y reduce la posibilidad de errores.

Hemos decidido trabajar con esta herramienta ya que nos ha permitido crear Mokups y plantillas de como tendría que ser la interfaz gráfica de la aplicación.

3. Ubuntu

Ubuntu es un sistema operativo basado en Linux que ofrece una experiencia de usuario intuitiva y una amplia gama de características y funcionalidades. Ofrece una interfaz de usuario intuitiva y amigable que permite a los usuarios acceder y utilizar el sistema operativo de manera eficiente.

Ilustración 17 Logo de Ubuntu.

Ubuntu es altamente personalizable y adaptable a una amplia variedad de casos de uso. Ofrece soporte para una amplia gama de hardware gracias a su eficiencia con el uso del Hardware. Ubuntu es altamente compatible con una amplia gama de software y tecnologías, lo que lo convierte en una opción versátil para una variedad de aplicaciones y proyectos.

Ubuntu ofrece un conjunto de herramientas y características de seguridad integradas para proteger los datos y la privacidad de los usuarios.

Hemos decidido utilizar este Sistema Operativo ya que es muy eficiente en el uso del Hardware y sobre todo por su amplia compatibilidad en cuanto al uso de Software.

4. Apache Server

Apache Server es un servidor web y multiplataforma desarrollado por la Apache Software. Apache es uno de los servidores web más utilizados en el mundo debido a su fiabilidad, flexibilidad y rendimiento.

Ilustración 18 Logo de Apache.

Apache es altamente configurable y extensible, lo que permite a los administradores de sistemas personalizar su funcionamiento mediante la adición de módulos que

proporcionan funcionalidades adicionales como la autenticación, la reescritura de URL y el soporte para múltiples lenguajes de programación. Estos módulos pueden ser habilitados o deshabilitados según las necesidades específicas del servidor.

Hemos decidido utilizar este software para ver la diferencia con la tecnología vista en clase y así poder explorar cuales son las funcionalidades que nos ofrece.

5. Git

Git es un sistema de control de versiones distribuido de código abierto ampliamente utilizado en el desarrollo de software. Git ofrece un método eficiente y confiable para administrar el historial de cambios en el código fuente de un proyecto.

Ilustración 19 Logo de git.

Una de las principales características de Git es que permite realizar un seguimiento preciso de los cambios en el código fuente a lo largo del tiempo. Esto se logra mediante la creación de instantáneas (commits) que registran los cambios realizados en los archivos del proyecto. Estas instantáneas se almacenan de manera incremental, lo que permite a los desarrolladores retroceder en el tiempo y recuperar versiones anteriores del código en caso de ser necesario.

Git es una herramienta útil para el trabajo colaborativo en proyectos de software. Permite a varios colaboradores trabajar en el mismo proyecto de manera simultánea y fusionar sus cambios de manera segura y eficiente.

Lo hemos decido utilizar para tener un historial de desarrollo sobre el proyecto y también para poder desarrollarlo desde cualquier lado sin necesidad de tener que almacenarlo en algún dispositivo móvil.

6. GitHub

GitHub es una plataforma de alojamiento de repositorios de código fuente basada en la web, que utiliza el sistema de control de versiones Git. GitHub es un centro fundamental para el desarrollo de software colaborativo, proporcionando herramientas y servicios para que los desarrolladores trabajen juntos de manera eficiente.

Ilustración 20 Logo de GitHub.

La principal función de GitHub es alojar repositorios de código fuente de manera pública o privada. Esto permite compartir su trabajo con otros de manera abierta y colaborativa, o mantenerlo privado para uso interno o proyectos sensibles.

Otra de las principales funciones de GitHub es su comunidad activa. Los desarrolladores pueden descubrir proyectos interesantes, contribuir a proyectos de código abierto, y conectarse con otros desarrolladores de todo el mundo.

Hemos decidido utilizar esta herramienta para almacenar nuestro repositorio para poder descubrir y explotar todas sus funcionalidades. Aparte queríamos aprender como seria el desarrollo colaborativo de un proyecto.

7. MySQL

MySQL es un sistema de gestión de bases de datos relacional ampliamente utilizado en el desarrollo de aplicaciones web y empresariales. MySQL en una opción popular entre los desarrolladores debido a su fiabilidad, escalabilidad y rendimiento.

Ilustración 21 Logo de MySQL.

MySQL tiene la capacidad para gestionar grandes volúmenes de datos de manera eficiente y escalable. MySQL utiliza un modelo de almacenamiento basado en tablas que permite a los desarrolladores organizar y acceder a los datos de manera estructurada. MySQL también ofrece una amplia gama de características avanzadas para optimizar el rendimiento de la base de datos.

Hemos decidido utilizar esta tecnología de almacenamiento porque era una de las más utilizadas del mercado y queríamos coger bastante experiencia de esta tecnología.

8. HeidiSQL

HeidiSQL es una herramienta de administración de bases de datos para MySQL, MariaDB, Microsoft SQL Server y PostgreSQL. Permite gestionar y manipular bases de datos de manera eficiente a través de una interfaz gráfica intuitiva.

Ilustración 22 Logo de HeidiSQL.

HeidiSQL ofrece características como la ejecución de consultas SQL, la administración de usuarios y permisos, la visualización de esquemas de bases de datos, la exportación e importación de datos, entre otras.
Su interfaz fácil de usar y su amplia compatibilidad con diferentes sistemas de gestión de bases de datos lo convierten en una herramienta popular para trabajar con bases de datos en entornos de desarrollo y producción.

Hemos decidido utilizar esta herramienta de gráfica para la gestión de la base de datos por su facilidad para ver, editar y mostrar las bases de datos y así poder cambiar o eliminar de una forma más rápida he intuitiva las bases de datos.

9. NodeJS

NodeJS es un entorno de tiempo de ejecución de JavaScript de código abierto y multiplataforma que permite ejecutar código JavaScript fuera de un navegador web. Esta característica lo hace especialmente útil para el desarrollo de aplicaciones del lado del servidor y aplicaciones de red. NodeJS ha ganado una gran popularidad debido a su eficiencia y escalabilidad.

Ilustración 23 Logo de NodeJS.

Node.js cuenta con un amplio ecosistema de paquetes npm (Node Package Manager). NPM es uno de los repositorios de software más grandes del mundo, lo que permite a los desarrolladores acceder a una amplia gama de módulos y bibliotecas de código abierto que pueden integrarse fácilmente en sus proyectos.

Node.js fomenta un enfoque modular en el desarrollo de aplicaciones, lo que significa que se pueden crear aplicaciones mediante la composición de módulos reutilizables. Esto promueve la organización del código y facilita el mantenimiento a medida que la aplicación crece en complejidad.

Hemos decidido utilizar esta tecnología porque es una de las más utilizadas para el entorno de desarrollo de Backend y así poder aprender y ver las diferencias con otros entornos de desarrollo ya vistos.

10. ExpressJS

ExpressJS es un framework de NodeJS para la creación de aplicaciones web flexibles y minimalistas que simplifica el desarrollo de aplicaciones web y APIs. ExpressJS se ha convertido en una opción popular entre los desarrolladores de Node.js debido a su flexibilidad, simplicidad y robustez.

Ilustración 24 Logo de ExpressJS.

Express.js es extensible y permite a los desarrolladores agregar funcionalidades adicionales a través de middleware. Los middlewares son funciones que tienen acceso tanto a la solicitud como a la respuesta en el ciclo de vida de una solicitud HTTP. Esto permite agregar funcionalidades como autenticación, compresión, registro de solicitudes y respuestas, y manejo de errores de manera modular y reutilizable.

Otra característica de ExpressJS es su amplia comunidad y ecosistema de paquetes. Express.js es compatible con npm, lo que permite acceder a una amplia gama de módulos y bibliotecas de código abierto que pueden integrarse fácilmente en sus proyectos.

Hemos decidido utilizar ExpressJS por su amplia compatibilidad con diversas tecnologías y su rendimiento a para responder a solicitudes HTTP.

11. Bcrypt

Bcrypt es una herramienta de cifrado de contraseñas diseñada para proteger las contraseñas almacenadas en aplicaciones y sistemas. Utiliza el algoritmo para realizar un hash criptográfico de las contraseñas, añadiendo una única criptografía a cada contraseña antes de cifrarla. Esta criptografía asegura que incluso si dos usuarios tienen la misma contraseña, sus hashes serán diferentes.

Bcrypt también permite ajustar el costo de procesamiento, lo que significa que se puede aumentar la complejidad del hash con el tiempo para contrarrestar el aumento en el poder de cómputo de los atacantes. Esto lo hace útil para proteger las contraseñas contra ataques de fuerza bruta, ofreciendo una capa adicional de seguridad en la gestión de contraseñas.

Hemos decidido utilizar esta tecnología para el encriptado de contraseñas debido a su alta escalabilidad en el tiempo y así poder mantener la seguridad de cada usuario.

12. Cluster

Cluster es un módulo que permite crear procesos que comparten el mismo puerto del servidor, lo que mejora el rendimiento y la escalabilidad de las aplicaciones al aprovechar los múltiples núcleos de CPU disponibles en el servidor.

Esto ayuda a manejar más conexiones concurrentes y a distribuir la carga de trabajo de manera más eficiente. Facilita la creación y gestión de estos procesos, permitiendo que el proceso principal se encargue de la distribución de las solicitudes entrantes a los diferentes procesos. En caso de que uno de los procesos falle, el módulo puede generar un nuevo proceso automáticamente, garantizando así la estabilidad y disponibilidad de la aplicación.

Lo hemos decidido utilizar para que el servidor pueda utilizar todo el hardware a su disposición y si una de las peticiones conlleva mas tiempo o se ha quedado parada, se pueda utilizar otras partes del hardware para poder mantener el servicio.

13. Cors

CORS es una política de seguridad web que permite a los servidores indicar qué orígenes pueden acceder a sus recursos. ORS es esencial para controlar las solicitudes HTTP realizadas desde un dominio distinto al del servidor. Esto es crucial para la seguridad, ya que previene el acceso no autorizado a recursos sensibles y protege contra ataques.

Este módulo se utiliza para habilitar y configurar CORS en aplicaciones web. Este módulo permite definir reglas específicas sobre qué dominios pueden realizar solicitudes, qué métodos HTTP están permitidos y qué encabezados pueden ser utilizados.

Lo hemos decidido utilizar para poder mandar por las cabeceras de las peticiones HTTP información encriptada sobre los usuarios.

14. Dotenv

Dotenv es una herramienta para cargar variables de entorno desde un archivo ".env" en aplicaciones Node.js. Este archivo almacena configuraciones sensibles como claves

de API, credenciales de bases de datos y otros parámetros de configuración en un formato de clave-valor.

Utilizando dotenv, estas variables se cargan automáticamente en process.env al iniciar la aplicación, lo que facilita la gestión de configuraciones y mejora la seguridad al mantener la información sensible fuera del código fuente.

Hemos decidido utilizar este módulo para almacenar credenciales importantes para el uso de algunas de las tecnologías ya que, al ser archivos ocultos, son más difíciles de ser interceptadas por atacantes y así poder mantener la seguridad de la aplicación.

15. Express-Fileupload

Express-Fileupload es un middleware para aplicaciones ExpressJS que facilita la carga de archivos desde el lado del cliente al servidor. Este middleware simplifica el proceso de gestionar archivos subidos a través de formularios HTML, proporcionando una interfaz fácil de usar para acceder y manipular los archivos en el servidor.

Este módulo permite recibir archivos cargados y almacenarlos en el servidor, leer sus contenidos, y realizar diversas operaciones como mover, renombrar o eliminar archivos. También soporta la configuración de límites de tamaño de archivo y la gestión de múltiples archivos simultáneamente.

Hemos decidido utilizar este módulo para recoger y almacenar las imágenes de los usuarios y de los productos en el servidor de Backend.

16. Express-session

Express-Session es un middleware para aplicaciones ExpressJS que gestiona sesiones de usuario en el servidor. Este middleware permite almacenar datos de sesión en el lado del servidor, asignando a cada usuario una sesión única mediante una cookie de sesión. Las sesiones son útiles para mantener el estado entre diferentes solicitudes HTTP.

Utilizamos este módulo para la gestión de las cabeceras y Cors y así poder gestionar los tokens de los usuarios que se mandan para poder identificar de quien es cada una de las peticiones que se realizan al Backend.

17. FS

Este modulo proporciona una API para interactuar con el sistema de archivos. Este módulo permite realizar operaciones de lectura, escritura, apertura, cierre y eliminación de archivos y directorios de manera sincrónica o asincrónica.

Es fundamental para cualquier aplicación Node.js que necesite manipular el sistema de archivos. Puede usarse para leer archivos de configuración, registrar actividades en archivos de log, almacenar y recuperar datos de archivos.

Hemos decidido utilizar este modulo para poder recoger información de alguno de los componentes del hardware y así podérselos mandar al servidor de frontend para que los administradores puedan ver el uso de alguno de los componentes del hardware.

18. Jsonwebtoken

Jsonwebtoken es un módulo que se utiliza para crear y verificar JSON Web Tokens (JWT). JWT es un estándar abierto para transmitir información de manera segura entre un cliente y un servidor como un objeto JSON.

Jsonwebtoken es ampliamente utilizado para la autenticación y autorización en aplicaciones web. Los tokens generados contienen información codificada y firmada digitalmente, lo que garantiza que los datos no hayan sido alterados. Esto permite a los servidores autenticar a los usuarios sin necesidad de almacenar información de sesión en el servidor, ya que toda la información relevante está contenida en el token.

Lo hemos decidido utilizar para crear tokens a los usuarios que inician sesión en la aplicación y así mandarles un token identificativo encriptado para poder garantizar la identidad del usuario en las peticiones entre el Frontend y el Backend.

19. Nodemailer

Nodemailer es un módulo que nos permite enviar correos electrónicos desde una aplicación de una manera sencilla y eficiente. Esta biblioteca es muy útil para integrar funcionalidades de correo electrónico, como enviar confirmaciones de registro, restablecer contraseñas, notificaciones y más, en aplicaciones web.

Este módulo es altamente configurable y permite enviar correos electrónicos de manera programática, especificando detalles como el remitente, el destinatario, el

asunto, el contenido del mensaje y cualquier adjunto. Además, nodemailer es compatible con varios servicios de correo electrónico y protocolos de transporte, lo que permite enviar correos electrónicos a través de SMTP, API de servicios de correo electrónico, o incluso servicios de terceros como Gmail, Outlook y otros.

Hemos decido utilizar esta tecnología ya que es muy fácil de configurar y muy versátil para mandar correos electrónicos.

20. Nodemon

Nodemon es una herramienta de desarrollo para Node.js que supervisa los archivos de un proyecto y automáticamente reinicia el servidor cuando detecta cambios en el código. Esto evita tener que reiniciar manualmente el servidor cada vez que se realiza una modificación en el código, lo que agiliza el proceso de desarrollo y mejora la productividad del desarrollador.

Nodemon monitorea los archivos del proyecto en busca de cambios y, cuando se detecta un cambio, detiene el servidor, reinicia automáticamente y vuelve a cargar los archivos actualizados. Esto permite ver los cambios en tiempo real mientras se desarrolla la aplicación, sin interrupciones en el flujo de trabajo.

Hemos decidido utilizar este modulo para agilizar y simplificar el reinicio del servidor al agregar o modificar cambios en las diferentes funcionalidades del servidor.

21. Os

Os proporciona funcionalidades para interactuar con el sistema operativo en el que se ejecuta la aplicación. Este módulo permite acceder a información sobre el sistema operativo, como el nombre del sistema operativo, la arquitectura del procesador, la versión del kernel, la información de red y mucho más.

Hemos decidido utilizar este modulo para monitorizar partes del hardware del equipo y así poder mostrar a los administradores de una manera gráfica y sencilla del uso del hardware del equipo.

22. Path

Path es un módulo que proporciona utilidades para trabajar con rutas de archivos y directorios de manera consistente entre diferentes sistemas operativos. Este módulo facilita la manipulación y construcción de rutas de archivos de forma segura y eficiente.

Path puede realizar operaciones como unir rutas, resolver rutas relativas y absolutas, obtener la extensión de un archivo, y extraer el nombre del archivo o directorio de una ruta dada.

Lo hemos decidido utilizar para controlar las rutas de acceso del Backend a las diferentes funcionalidades ya que es un módulo muy fácil de utilizar y que funciona muy bien con las peticiones HTTP.

23. Sequelize

Sequelize facilita la interacción con bases de datos relacionales utilizando JavaScript. Este módulo permite trabajar con bases de datos SQL de una manera más orientada a objetos, eliminando la necesidad de escribir consultas SQL manualmente y facilitando la gestión de datos en aplicaciones.

Sequelize permite definir modelos de datos que representan tablas en la base de datos, así como establecer relaciones entre estos modelos. Sequelize se encarga de traducir las operaciones de consulta y manipulación de datos realizadas en JavaScript a consultas SQL correspondientes, lo que simplifica el proceso de acceso y manipulación de datos en la base de datos.

Hemos decidido utilizar esta tecnología ya que simplifica el acceso a los datos de la base de datos y también previene automáticamente de la inyección de SQL por parte de los usuarios.

24. MySQL2

MySQL2 es un controlador MySQL que permite a las aplicaciones conectarse y comunicarse con bases de datos MySQL de manera eficiente. Este controlador está diseñado para ser más rápido y escalable.

Este módulo permite de una manera fácil y simplificada de la conexión entre el servidor de Backend y la base de datos permitiendo así hacer consultas de una manera fácil y simplificada.

Este modulo lo hemos decidido utilizar debido a que Sequelize necesita este modulo y es uno de los mejores para la conexión de la base de datos.

25. React

React es una biblioteca de JavaScript desarrollada para construir interfaces de usuario interactivas y reactivas. Utiliza un enfoque basado en componentes, lo que significa que las interfaces de usuario se dividen en pequeños elementos reutilizables llamados componentes. Estos componentes encapsulan el estado y el comportamiento de la interfaz de usuario, lo que facilita la creación y mantenimiento de aplicaciones complejas.

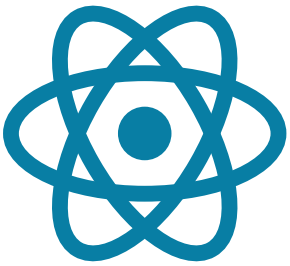

Ilustración 25 Logo de React.

React también promueve la unidireccionalidad del flujo de datos, lo que significa que los datos fluyen en una dirección, desde los componentes principales hacia los componentes secundarios. Esto facilita el mantenimiento del estado de la aplicación y reduce la posibilidad de errores debido a cambios inesperados en los datos.

Hemos decidido utilizar esta tecnología para explorar nuevas formas de crear servidores de Frontend utilizando componentes para poder reutilizar en diferentes partes el código.

26. Next

Next es un framework de React que se utiliza para construir aplicaciones web de una sola página y aplicaciones web estáticas. Ofrece una forma eficiente y productiva de crear aplicaciones web modernas con React, proporcionando funcionalidades adicionales que van más allá de lo que React ofrece por sí solo.

Ilustración 26 Logo de Next.

Next ofrece una capacidad para renderizar tanto en el servidor como en el cliente. Esto permite que las páginas pueden pre-renderizarse en el servidor y luego enviarse al cliente, lo que mejora el rendimiento y la velocidad de carga inicial. Además, Next permite la generación de páginas estáticas en tiempo de compilación, lo que resulta en un mejor SEO y tiempos de carga más rápidos.

Next también incluye características como enrutamiento basado en archivos, pre-procesamiento de CSS, optimización de imágenes, soporte para TypeScript y mucho más, lo que facilita el desarrollo de aplicaciones web complejas y de alto rendimiento. Además, cuenta con una amplia gama de complementos y bibliotecas que amplían su funcionalidad.

Hemos decidido utilizar esta tecnología para ver la diferencia que ofrecía sobre Angular y así poder ver nuevos frameworks de creación de interfaces basado en componentes.

27. NextUI

NextUI es una biblioteca de componentes de interfaz de usuario para React que facilita la creación de aplicaciones web con un diseño moderno y atractivo. Desarrollada para integrarse perfectamente con Next y otras aplicaciones React, NextUI proporciona una colección de componentes reutilizables y estilizados que permiten a los desarrolladores construir interfaces de usuario de alta calidad de manera rápida y eficiente.

Ilustración 27 Logo de NextUI.

Los componentes de NextUI están optimizados para ser ligeros y rápidos, lo que ayuda a mantener un rendimiento óptimo en las aplicaciones, lo que es especialmente útil para aplicaciones construidas con Next.js, mejorando la velocidad de carga y el SEO.

Lo hemos decidió utilizar ya que aporta componentes muy eficientes de una manera fácil y sencilla. Estos componentes son fáciles de modificar y personalizar para la temática de la página.

28. Axios

Axios es una biblioteca de JavaScript para realizar solicitudes HTTP desde el navegador o desde Node.js. Es ampliamente utilizada debido a su facilidad de uso y su amplia compatibilidad con diferentes entornos de ejecución.

Axios puede realizar solicitudes HTTP de forma sencilla y eficiente, incluyendo solicitudes GET, POST, PUT, DELETE y otras. Axios proporciona una API fácil de usar que permite configurar y personalizar solicitudes con facilidad, incluyendo opciones como encabezados personalizados, parámetros de consulta, y datos de solicitud.

Una de las características más útiles de Axios es su manejo de respuestas y errores. Permite manejar respuestas en diferentes formatos como JSON y maneja automáticamente los errores HTTP, lo que simplifica la gestión de errores en las solicitudes HTTP.

Hemos decidido utilizar esta herramienta para la conexión entre el servidor de Frontend y el Backend ya que aparte permitir respuestas entre ellos, no permite utilizar las cabeceras de estar peticiones para mandar información importante del usuario.

29. Stripe

Stripe es una plataforma de pagos en línea que proporciona herramientas y APIs para encargarse de pagos en línea y gestionar transacciones financieras de manera segura y eficiente.

Ilustración 28 Logo de Stripe.

Stripe permite implementar fácilmente funcionalidades de pago en línea, como procesamiento de tarjetas de crédito y débito, pagos por suscripción, pagos móviles, y mucho más. Stripe ofrece una API intuitiva y fácil de usar que permite realizar diversas operaciones, como crear cargos, gestionar clientes y suscripciones, generar informes y mucho más.

Stripe se ha focalizado en la seguridad. Utiliza prácticas de seguridad, así como la tokenización de tarjetas de crédito, para proteger la información confidencial del cliente durante las transacciones. Además, Stripe es compatible con el cumplimiento

de estándares de seguridad, lo que garantiza que las transacciones sean seguras y cumplan con los requisitos legales y de cumplimiento.

Hemos decidido utilizar esta aplicación para procesar los pagos debido a la legislación con el almacenamiento y procesamiento de los datos bancarios ya que para poder procesarlos necesitas unos estándares de seguridad muy superiores y con unas certificaciones muy difíciles de conseguir.

30. TailwindCSS

Tailwind CSS es un framework de diseño para construir interfaces de usuario modernas y receptivas. A diferencia de otros frameworks CSS como Bootstrap, Tailwind CSS se centra en la creación de componentes de interfaz de usuario mediante la aplicación directa de clases predefinidas en lugar de estilos CSS personalizados.

Ilustración 29 Logo de TailwindCSS.

Una de las ventajas de Tailwind CSS es su enfoque de bajo nivel que permite una mayor flexibilidad y personalización. Los desarrolladores pueden combinar clases de utilidad para crear diseños complejos y adaptar el diseño a las necesidades específicas del proyecto. Además, Tailwind CSS es altamente extensible y permite la creación de clases personalizadas y temas para adaptarse al estilo y la estética del proyecto.

Hemos decidido utilizar este framework para ver las diferencias entre TailwindCSS y Bootstrap y así aprender otro framework de desarrollo. Aparte de esta decisión, también hemos utilizado este framework ya que carga las clases necesarias para renderizar los componentes que se están renderizando y así mejorar la eficiencia y el SEO de la página.

31. ChatGPT

ChatGPT es un modelo de lenguaje desarrollado por OpenAI que utiliza inteligencia artificial para generar texto basado en las entradas proporcionadas por los usuarios. ChatGPT es capaz de comprender y generar texto de manera coherente y contextual, lo que lo hace útil para una amplia variedad de aplicaciones, como chatbots, asistentes virtuales, generación de contenido, y más.

Ilustración 30 Logo de ChatGPT.

Hemos decidido utilizar ChatGPT como una herramienta de comprobación y explicación de errores. También lo hemos utilizado para coger ideas de como generar funcionalidades para la aplicación.

32. GitHub Copilot

GitHub Copilot es una herramienta de inteligencia artificial desarrollada por GitHub en colaboración con OpenAI que asiste a los desarrolladores en la escritura de código. Utiliza el modelo de lenguaje GPT-3 para proporcionar sugerencias y completar automáticamente el código mientras los desarrolladores escriben.

Ilustración 31 Logo de GitHub Copilot.

Lo hemos utilizado como asistente a preguntas que nos han ido surgiendo en el desarrollo del proyecto ya que nos daba sugerencias de posibles soluciones a las preguntas que nos iban surgiendo.

IV. DIAGRAMAS DE ENTIDAD-RELACION

1. DIAGRAMA DE LA BASE DE DATOS

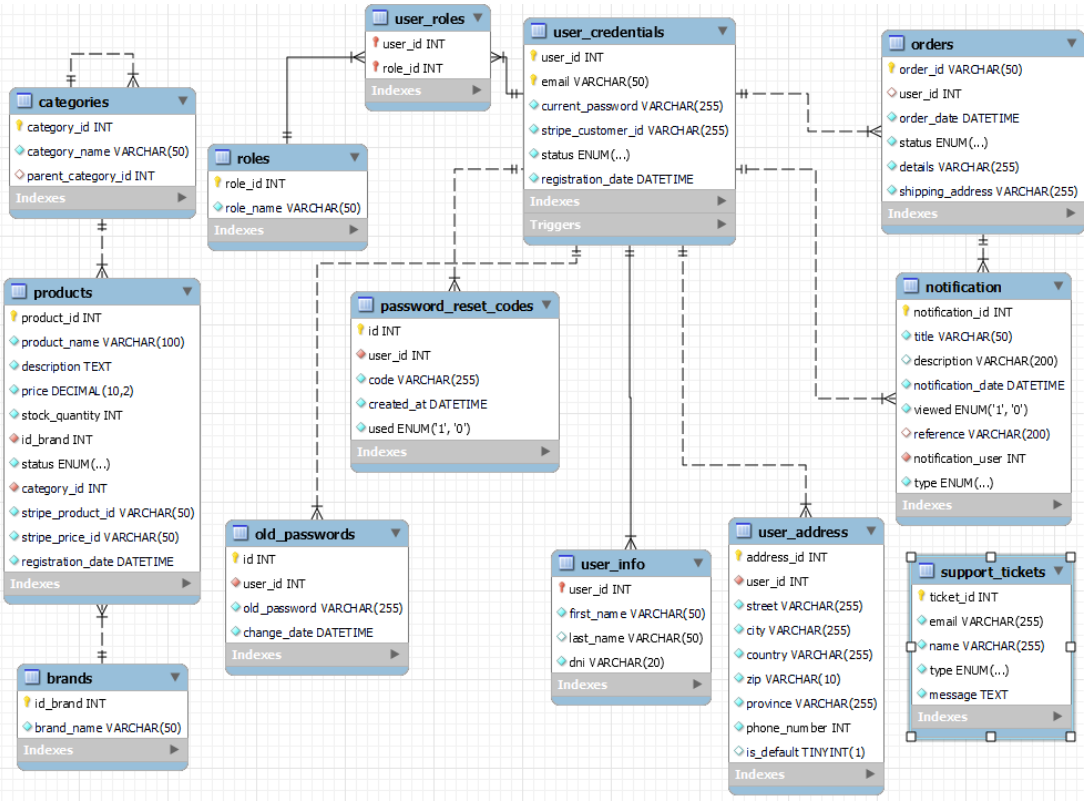

Ilustración 32 Diagrama de entidad-relación.

V. DIAGRAMA DE CASOS DE USO

Para ver el diagrama de casos de uso esta en PDF anexo a la entrega.

MARCO PRÁCTICO

En este apartado del trabajo vamos a describir y analizar los aspectos prácticos relacionados con la estructuración del proyecto y como se utilizan alguno de los apartados de la página web en cuestión.

I. ESTRUCTURA DEL PROYECTO

1. Estructuración del Backend

La carpeta del servidor de Backend, contiene las carpetas de database, public y src; aparte de estas capetas, contiene los archivos de .env, .gitignore, packge-lock.json y package.json.

- **.env**
  Este archivo contiene directivas de seguridad secretas sobre la aplicación y los diferentes servicios que se utilizan en ella.

- **.gitignore**
  En el caso de este proyecto, estamos utilizando la tecnología de GitHub. En este archivo almacenamos los archivos que no queremos que se suban a GitHub.

- **Package-lock.json**
  Es esencial para gestionar de manera confiable y segura las dependencias en proyectos Node.js.

- **Package.json**
  Configura y describe tu proyecto Node.js, facilitando la gestión de dependencias, automatización de tareas y asegurando la compatibilidad del entorno.

- **Database**
  Esta carpeta contiene el diagrama de la base de datos, el SQL para generar la base de datos entera y algunos datos para la base de datos como ejemplos.

- **Public**
  Contiene imágenes que se usan desde el frontend como por ejemplo las imágenes de perfil de usuario, las imágenes de los productos añadidos, etc.

- **Src**
  Contiene el archivo que ejecuta el servidor de Backend, el enrutador de las rutas del servidor de Backend, el modelo de la base de datos, así como cada una de las entidades de la base de datos con sus respectivos controller, router y modelos.

- **Index.js**
  El fichero index.js es el archivo principal del Backend, el cual sirve como el punto de entrada principal de la aplicación, donde se configura y se levanta el servidor Express, se definen las rutas de la API, se establece la conexión con la base de datos y se configuran los middlewares necesarios para el funcionamiento adecuado de la aplicación.

- **Routes.js**
  El fichero routes.js contiene las routes de los diferentes endpoints done se podrán hacer peticiones desde el frontend para obtener datos o realizar alguan acción en la base de datos.

- **API**
  Contiene los ficheros para cada endpoint de la API, tanto el controlador, el fichero de rutas y el modelo que contiene las diferentes acciones CRUD (Create, Read, Update, Delete), están organizadas con el respectivo nombre del enpoint.

- **Model**
  Contiene los diferentes modelos de base de datos creados con el módulo de NodeJS "Sequelize", esperados cada uno por cada diferente modelo de la base de datos, y unidos por un fichero index.js que contiene las relaciones entre estos.

- **Middelware**
  Contiene las funciones que se tiene que hacer entre las peticiones HTTP y las respuestas del servidor.

2. Estructura del Frontend

La carpeta del servidor de Frontend, contiene las carpetas de .next, .vscode, components, config, context, hooks, icons, layouts, pages, public, services, styles y types; aparte, también contiene los ficheros de .env, .eslintrc.json, .gitignore, LICENSE, next.config.js, packge-look.json, package.json, postcss.config.js, README.md, tailwind.config.js y tsconfig.json.

- **.env**
  Este archivo contiene directivas de seguridad secretas sobre la aplicación y los diferentes servicios que se utilizan en ella.

- **.eslintrc.json**
  Es un archivo de configuración clave para ESLint que define las reglas y entornos para el análisis estático del código JavaScript, mejorando la calidad y consistencia del mismo.

- **.gitignore**
  En el caso de este proyecto, estamos utilizando la tecnología de GitHub. En este archivo almacenamos los archivos que no queremos que se suban a GitHub.

- **LICENSE**
  Especifica los términos de uso, distribución, y modificación del software, proporcionando un marco legal que protege tanto a los creadores como a los usuarios del software.

- **Next.config.js**
  Permite personalizar el comportamiento de la aplicación, optimizar su rendimiento, gestionar rutas y redirecciones, y configurar variables de entorno, proporcionando una gran flexibilidad y control sobre la aplicación.

- **Package-lock.json**
  Es esencial para gestionar de manera confiable y segura las dependencias en proyectos Node.js.

- **Package.json**
  Configura y describe tu proyecto Node.js, facilitando la gestión de dependencias, automatización de tareas y asegurando la compatibilidad del entorno.

- **Postcss.config.js**
  Permite definir y configurar una variedad de plugins que mejoran y transforman el CSS de la aplicación. Este archivo facilita tareas como la auto prefijación, la mitificación y el uso de características avanzadas de CSS, adaptándose a diferentes entornos de desarrollo y producción.

- **README.md**
  Es un documento fundamental que proporciona información esencial sobre un proyecto de software, incluyendo instrucciones de instalación, uso, contribución y licencia, lo que facilita su adopción y colaboración por parte de otros desarrolladores.

- **Tailwind.config.js**
  Es un archivo de configuración en proyectos que utilizan Tailwind CSS, permitiendo personalizar y extender las configuraciones predeterminadas del framework para adaptarse a las necesidades específicas del proyecto.

- **Tsconfig.json**
  Es un archivo de configuración en proyectos TypeScript que define cómo se compila el código TypeScript a JavaScript, así como varias opciones de configuración para el compilador de TypeScript.

- **.next**
  Almacena archivos y datos generados durante el proceso de compilación y ejecución de la aplicación, incluyendo archivos compilados, activos estáticos, imágenes optimizadas y otros datos necesarios para el funcionamiento de la aplicación. Esta carpeta es esencial para el funcionamiento interno de Next.js y no debe ser modificada manualmente.

- **.vscode**
  Almacena configuraciones específicas del proyecto que son necesarias para el desarrollo en Visual Studio Code.

- **Components**
  Almacena los componentes que se usan en las diferentes páginas del proyecto.

- **Config**
  Contiene archivos de configuración personalizados creados por los desarrolladores para manejar aspectos específicos del proyecto.

- **Context**
  Almacena archivos relacionados con la gestión del estado global de la aplicación utilizando React Context API, lo que facilita la reutilización de datos a lo largo de todo el proyecto de forma sencilla.

- **Hooks**
  Almacena archivos que contienen hooks personalizados de React, lo que facilita la reutilización, modularidad y mantenimiento del código en la aplicación.

- **Icons**
  Almacena los iconos de la página web para su reutilización desde cualquier página o componente.

- **Layouts**
  Almacena archivos que contienen platillas para la creación de diferentes páginas con el mismo formato de una forma más simple y eficaz.

- **Pages**
  En este se encuentran las diferentes páginas del proyecto web, están organizadas al igual que como se accederá a las diferentes páginas en el navegador, por ejemplo si tienes en tu carpeta de páginas un fichero about.js si en tu navegador entras en /about estarás accediendo a este archivo.

- **Public**
  Almacena todos los elementos estáticos de la página como podría ser el logo, o algunas imágenes.

II. DISEÑO DEL SITIO WEB

En este apartado vamos a enseñar algunas de las partes graficas de las funcionales del Frontend.

1. Home de la página

El diseño del home de la página web sigue una estructura clara y bien organizada, dividiendo el contenido en secciones distintas que facilitan la navegación y la búsqueda de productos. Aquí hay un desglose detallado de las diferentes partes de la página:

Encabezado:

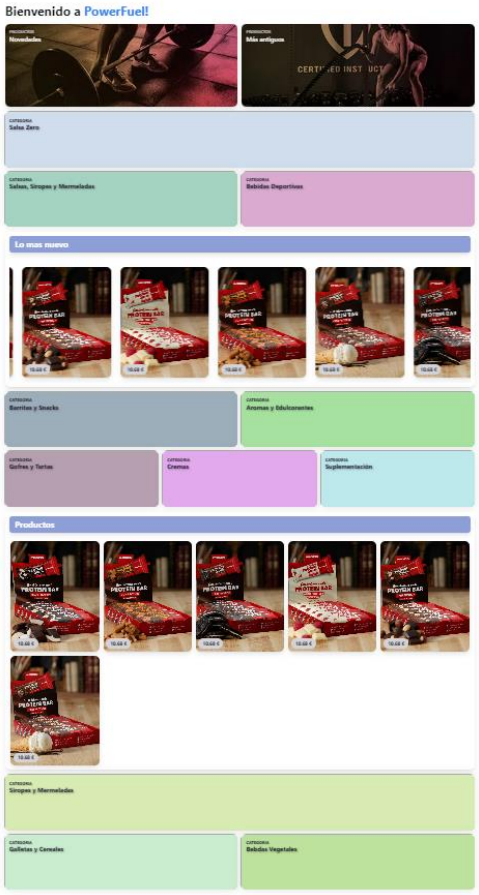

Ilustración 33 Home de la página

- El encabezado da la bienvenida al usuario.
- Sección de Novedades y más antiguos: Justo debajo del encabezado, hay dos tarjetas que parecen ser para las noticias más recientes y antiguas.
- Categorías Principales: Hay varias tarjetas grandes que representan las categorías principales de productos. Estas categorías ayudan a los usuarios a encontrar rápidamente los tipos de productos que están buscando.
- Lo más nuevo: Esta sección de carrusel muestra una serie de productos destacados con imágenes y precios visibles.
- Subcategorías: Justo debajo de los productos nuevos, se encuentran más categorías y subcategorías.
- En la sección de productos, hay una serie de productos aleatorios.
- Más categorías: Al final de la página, se muestran más categorías.

2. Inicio de sesión y Registro de los usuarios

En este proyecto, el inicio de sesión es muy importante ya que, sin él, el usuario no podría realizar las compras de los productos. La interfaz de inicio de sesión y registro es fundamental para la experiencia del usuario, ya que es el punto de entrada principal para los usuarios registrados y potenciales clientes. A continuación, se detallan los elementos y el diseño de esta interfaz:

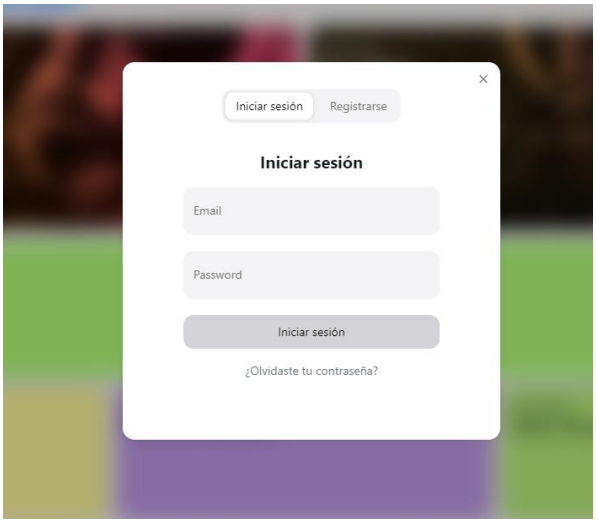

Ilustración 34 Inicio de sesión.

- **Ventana Emergente (Modal)**
  La ventana de inicio de sesión y registro se presenta como una ventana emergente (modal) que se superpone al contenido de la página principal. Este enfoque permite a los usuarios acceder fácilmente sin salir de la página actual.

- **Pestañas de Navegación**
  En la parte superior de la ventana emergente, se encuentran dos pestañas:
  - **Iniciar sesión:** Para usuarios que ya tienen una cuenta.
  - **Registrarse:** Para nuevos usuarios que desean crear una cuenta.

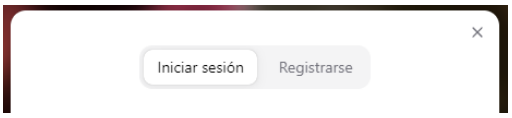

Ilustración 35 Pestaña navegación.

- **Formulario de Inicio de Sesión**
  La pestaña de "Iniciar sesión" contiene los siguientes elementos:
  - **Campo de Email:** Un campo de entrada donde los usuarios deben introducir su dirección de correo electrónico.
  - **Campo de Password:** Un campo de entrada para la contraseña del usuario.
  - **Botón de Iniciar Sesión:** Un botón que, al ser presionado, envía las credenciales para su verificación.

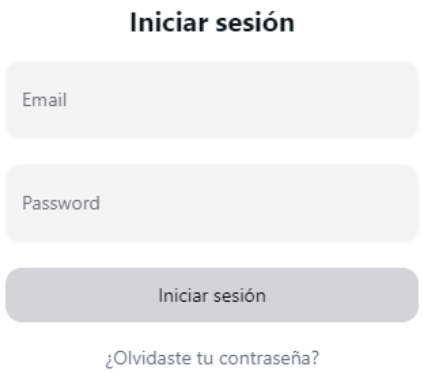

Ilustración 36 Login.

  - **Botón de ¿Olvidaste tu contraseña?:** Un botón que, al ser presionado, te lleva a un modal para introducir tu correo electrónico.

- **Formulario de Registro**
  En la pestaña de "Registrarse", el formulario incluye:
  - **Campo de Nombre y Campos de apellidos:** Campo de entrada para el nombre y los apellidos del usuario.
  - **Campo de Email:** Campo de entrada para la dirección de correo electrónico.
  - **Campo de Password:** Campo de entrada para la contraseña, con requisitos mínimos de seguridad.
  - **Confirmación de Password:** Un campo adicional para confirmar la contraseña.

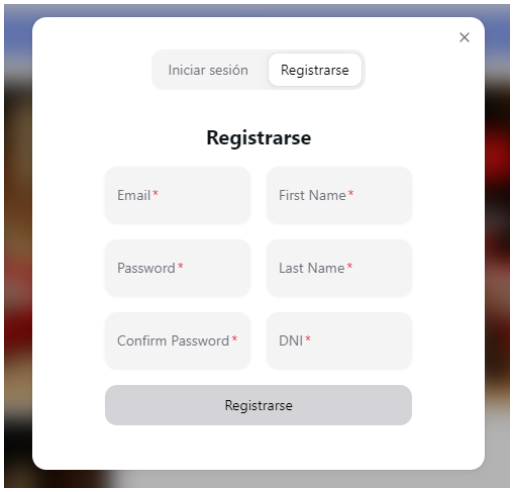

Ilustración 37 Registro.

  - **DNI:** Campo para tener un documento oficial de identidad del usuario.

- **Accesibilidad y Usabilidad**
  - **Diseño Limpio y Simple:** Se utiliza un diseño minimalista para evitar sobrecargar visualmente al usuario. Los campos y botones están bien espaciados y son fáciles de identificar.
  - **Tipografía Clara y Legible:** Se selecciona una tipografía clara y legible para todos los textos, asegurando que los usuarios puedan leer fácilmente las instrucciones y etiquetas.
  - **Botones Grandes y Claros:** Los botones de acción son grandes y claramente etiquetados para que los usuarios sepan exactamente qué acción tomar.

- **Consideraciones de Seguridad**
  Implementación de validaciones en los campos de entrada para asegurar que se ingresen datos correctos y completos (por ejemplo, formato correcto de email, contraseñas que cumplan con los requisitos de seguridad).

- **Diseño Responsivo**
  La interfaz está diseñada para ser completamente responsiva, adaptándose a diferentes tamaños de pantalla y dispositivos, desde ordenadores de escritorio hasta teléfonos móviles y tabletas.

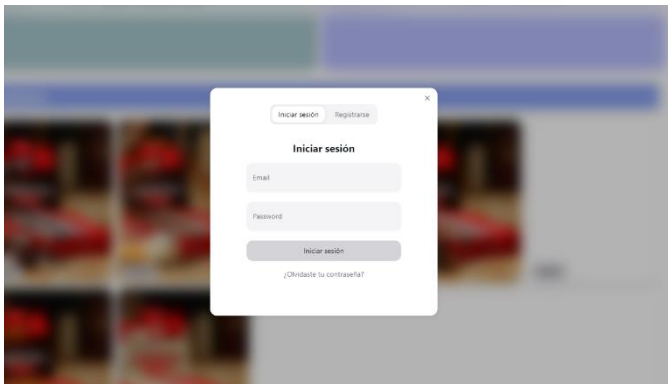

Ilustración 39 Vista monitor.

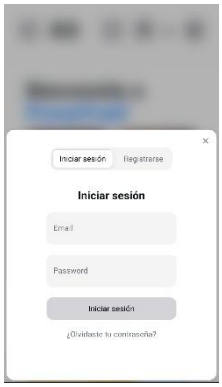

Ilustración 38 Vista móvil.

- **Mensajes de Error y Éxito**
  - **Mensajes de Error:** Si hay un problema con el inicio de sesión o registro (por ejemplo, credenciales incorrectas, email ya registrado), se muestran mensajes de error claros y específicos.

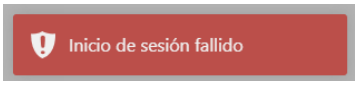

Ilustración 41 Mensaje de error.

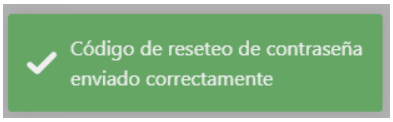

Ilustración 40 Mensaje de éxito.

- **Autenticación del usuario**

Este sistema permite a los usuarios acceder de manera segura a la plataforma mediante un token único que contiene la identificación del usuario.

- **Proceso de Autenticación**

El proceso de autenticación consta de los siguientes pasos:

  - **Inicio de sesión:** Para el inicio de sesión es necesario que el usuario complete el formulario de inicio de sesión y haber completado los patrones de seguridad.

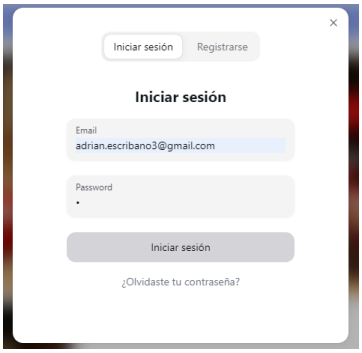

Ilustración 42 Intento de inicio de sesión.

  - **Generación del Token:** Cuando un usuario inicia sesión correctamente, se genera un token único que contiene la ID del usuario. Este token se cifra para garantizar su seguridad.

```
14  const generateAuthToken = (user) => {
15      try {
16          const payload = { userId: user.user_id, role: user.role_id };
17          return jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: process.env.JWT_TOKEN_AUTH_EXPIRATION });
18      } catch (error) {
19          console.log(`Error al generar el token de autenticación ${errorDisplay}`, error);
20      }
21  }; <- #14-21 const generateAuthToken = (user) =>
```

Ilustración 43 Generación del token de seguridad.

  - **Envío del Token al Cliente:** Una vez generado, el token se envía al cliente y se almacena localmente (en la sesión del navegador o en el almacenamiento local del cliente).

```
const getUsersByRegistrationDate = async (startDate, endDate) => {
    try {
        const users = await getUsersByRegistrationDate(new Date(startDate), new Date(endDate));
        return users;
    } catch (error) {
        console.log(`Error al intentar obtener los usuarios por fecha de registro ${errorDisplay}`, error);
    }
}; <- #194-201 const getUsersByRegistrationDate = async (startDate, endDate) =>
```

Ilustración 44 Envío del token.

- **Envío del Token en Solicitudes Posteriores:** En cada solicitud subsiguiente realizada por el cliente (por ejemplo, al acceder a recursos protegidos), el token se adjunta en el encabezado de autorización de la solicitud.

```
tabnine: test | explain | document | ask
api.interceptors.request.use((config) => {
    const token = localStorage.getItem('auth_token');
    if (token) {
        config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
tabnine: test | explain | document | ask
}, (error) => {
    return Promise.reject(error);
}); <- #18-26 api.interceptors.request.use
```

Ilustración 45 Envío del token en solicitudes posteriores.

- **Validación del Token:** El servidor valida el token recibido en cada solicitud para asegurarse de que el usuario esté autenticado y autorizado para acceder al recurso solicitado. Si el token es válido, se permite el acceso al recurso solicitado. De lo contrario, se devuelve un error de autenticación.

```
const loginUser = async (email, password) => {
    try {
        const user = await model.getUserByEmail(email);

        if (user && await bcrypt.compare(password, user.current_password) && user.status === 'Active'){
            const authToken = generateAuthToken(user);

            return authToken;
        } <- #172-176 if (user && await bcrypt.compare(password, user.current_passw...

        return null;
    } catch (error) {
        console.log(`Error al intentar iniciar sesión ${errorDisplay}`, error);
    }
}; <- #168-182 const loginUser = async (email, password) =>
```

Ilustración 46 Validación del token.

- **Consideraciones de Seguridad**
  - **Cifrado del Token:** El token se cifra utilizando algoritmos seguros para proteger la información del usuario durante la transmisión y el almacenamiento.

```
return jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: process.env.JWT_TOKEN_AUTH_EXPIRATION });
```

Ilustración 47 Cifrado del token

```
# Clave secreta para el token de acceso JWT
JWT_SECRET='your-very-secret-key'
```

Ilustración 48 Contraseña de cifrado.

- **Expiración del Token:** Se establece un tiempo de expiración para el token para mitigar el riesgo de ataques de reutilización. Cuando un token expira, el usuario debe volver a autenticarse para obtener uno nuevo.

```
return jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: process.env.JWT_TOKEN_AUTH_EXPIRATION });
```

Ilustración 49 Expiración del token.

```
# JWT CONFIGURATION
JWT_TOKEN_AUTH_EXPIRATION='5H'
```

Ilustración 50 Tiempo de expiración.

- **Una vez expirado el token:** Una vez el token expira, se elimina del LocalStorage del usuario para que se tenga que autenticar de nuevo.

```
api.interceptors.response.use(response => {
    return response;
}, async error => {
    if (error.response && (error.response.status === 401 || error.response.status === 403) && localStorage.getItem('auth_token')) {
        toastr.error(error);
        localStorage.removeItem('auth_token');
        window.location.reload();
    } <- #40-44 if (error.response && (error.response.status === 401 || error...
    return Promise.reject(error);
}); <- #37-46 api.interceptors.response.use
```

Ilustración 51 Eliminación del token del LocalStorage.

3. Panel de Administración

El panel de administrador proporciona una interfaz centralizada para gestionar diversos aspectos de la plataforma. Estos aspectos van desde la gestión de productos y usuarios como visión del rendimiento del servidor.

- **Página de Inicio**

  La página de inicio muestra información relevante para los administradores, entre ellos, el número de usuarios, numero de productos, numero de productos sin stock, porcentaje de uso de alguno de los componentes del hardware.

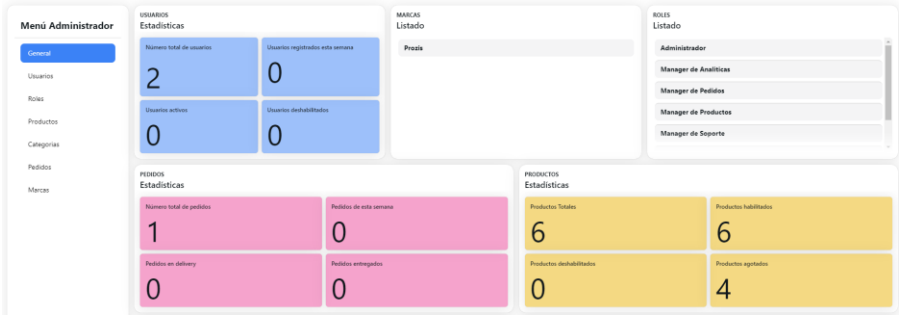

Ilustración 52 Panel de administración I.

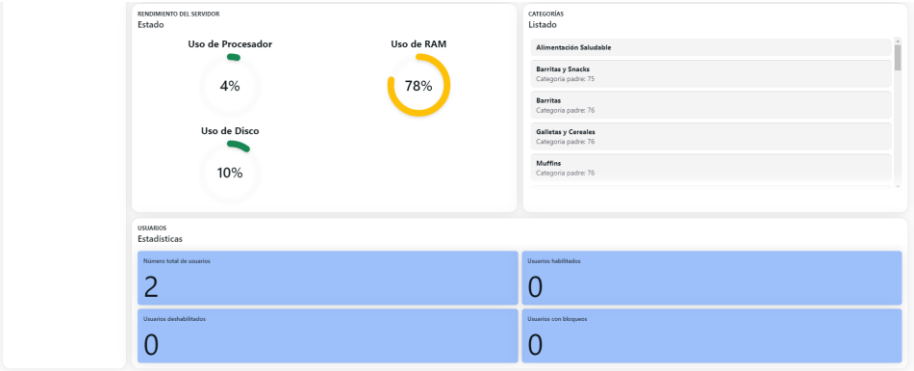

Ilustración 53 Panel de administración II.

- **Menú barra lateral panel de administrador:**
  El menú lateral proporciona acceso rápido a las diferentes secciones de administración en las que dependiendo del rol del usuario va a poder elegir unas u otras ya que al cargar el componente comprueba el rol y cargar las opciones permitidas a los usuarios de ese rol:

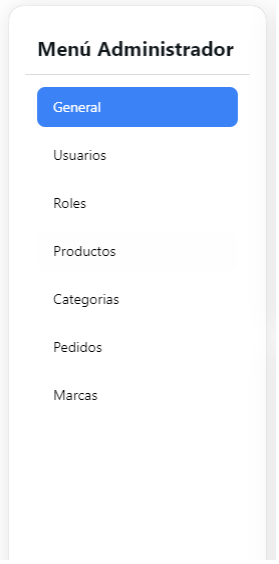

Ilustración 54 Sidebar.

  - **Usuarios:** Gestión de usuarios, incluyendo creación, edición, eliminación, asignación de roles y visualización de información de perfil.

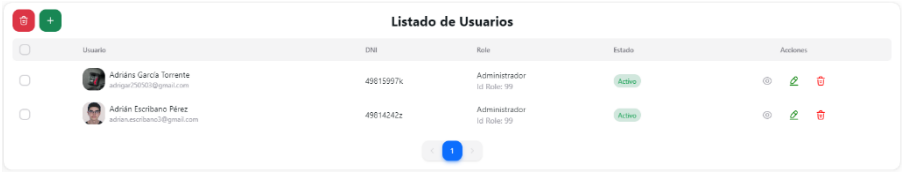

Ilustración 55 Listado de usuarios.

  - **Roles:** Administración de roles de usuario, incluyendo creación, edición y eliminación.

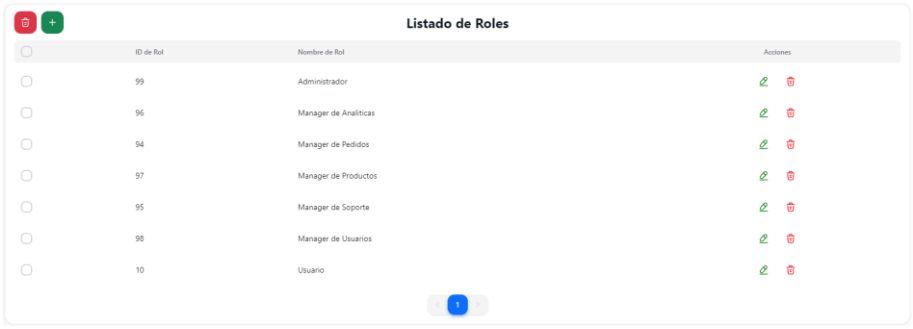

Ilustración 56 Listado de roles.

- **Categorías:** Gestión de categorías de productos, creando, editando, eliminando y organizando la estructura de categorías.

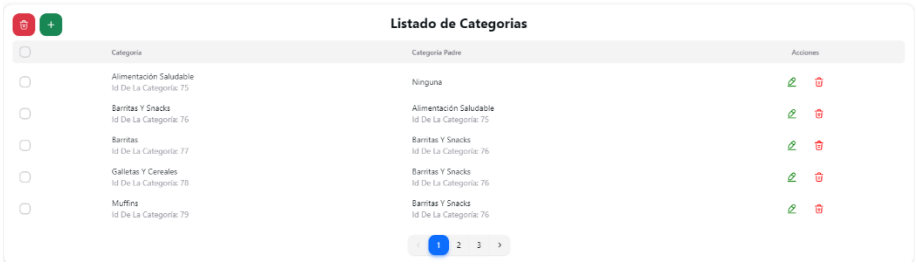

Ilustración 57 Listado de categorías

- **Productos:** Administración completa del catálogo de productos, incluyendo creación, edición, eliminación, carga de imágenes, asignación de categorías y establecimiento de precios.

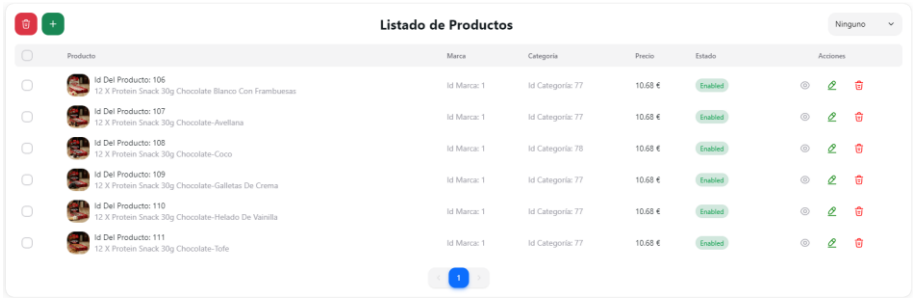

Ilustración 58 Listado de productos.

- **Marcas:** Administración completa de las marcas, incluyendo creación, edición y eliminación de las marcas.

Ilustración 59 Listado de marcas.

- **Pedidos:** Administración completa de los pedidos, incluyendo la edición y eliminación de los pedidos.

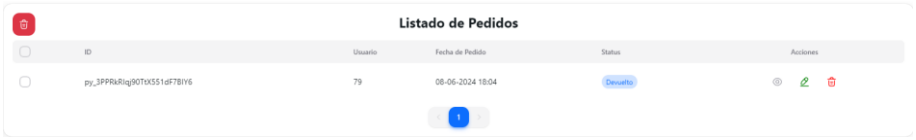

Ilustración 60 Listado de pedidos.

4. Panel de Usuario

En el panel de usuario están reflejado todos los datos del usuario, así como sus pedidos, direcciones de envió y sus notificaciones.

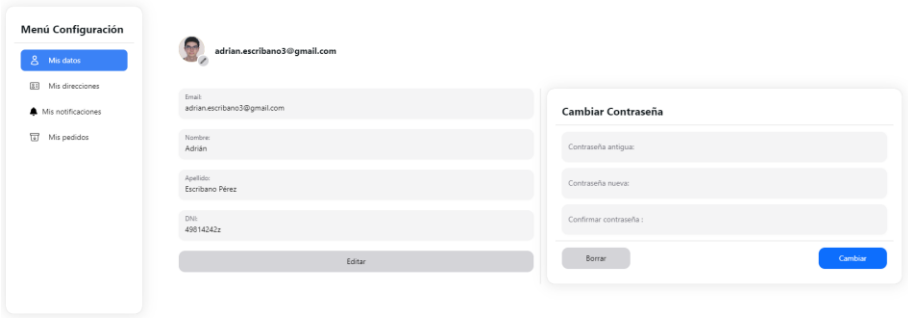

Ilustración 61 Panel de usuario.

- **Panel menú lateral**

  Contiene alguna de las funciones que el usuario puede ver y utilizar como ver sus datos, ver sus direcciones y editarlas, ver sus notificaciones y ver sus pedidos, así como poder cambiar el estado del pedido. Todas estas opciones son las que el usuario va a poder utilizar en la página de Usuario.

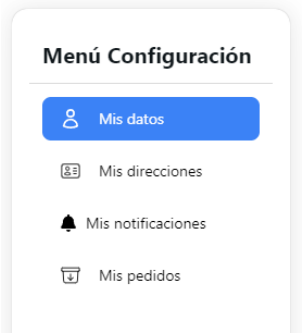

Ilustración 62 Menu de configuración.

- **Mis datos:** contiene los datos personales del usuario. En ella va a poder modificar su información, así como añadir o editar su imagen de perfil. También incluye un formulario de cambio de contraseña.

Ilustración 63 Sección 'Mis datos'.

- **Mis direcciones:** Este apartado contiene las direcciones de envió del usuario. En este menú también se puede editar, eliminar y añadir nuevas direcciones.

Ilustración 64 Listado de direcciones de envío.

- **Mis notificaciones:** En este apartado se pueden ver las notificaciones del usuario y permite poder borrarlas.

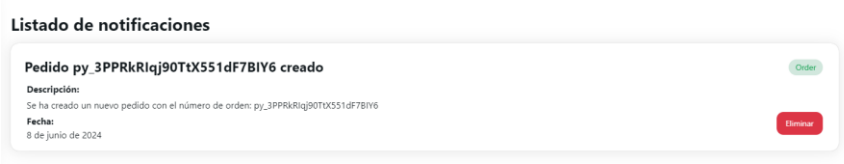

Ilustración 65 Listado de notificaciones.

- **Mis pedidos:** En esta parte del menú, se podrán ver todos los pedidos realizados por el usuario. En él se podrán cancelar o devolver los pedidos si es preciso.

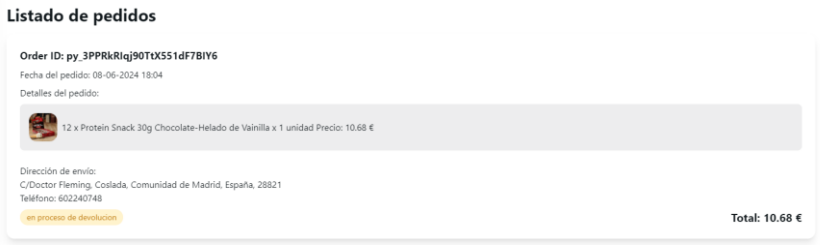

Ilustración 66 Listado de pedidos.

5. Formulario de Ayuda

En este formulario cualquier tipo de usuario podrá mandar una incidencia de cualquier tipo solo especificando su correo para que luego se ponga en contacto alguno de los administradores a través del correo. También se tendera que especificar el tipo de incidencia.

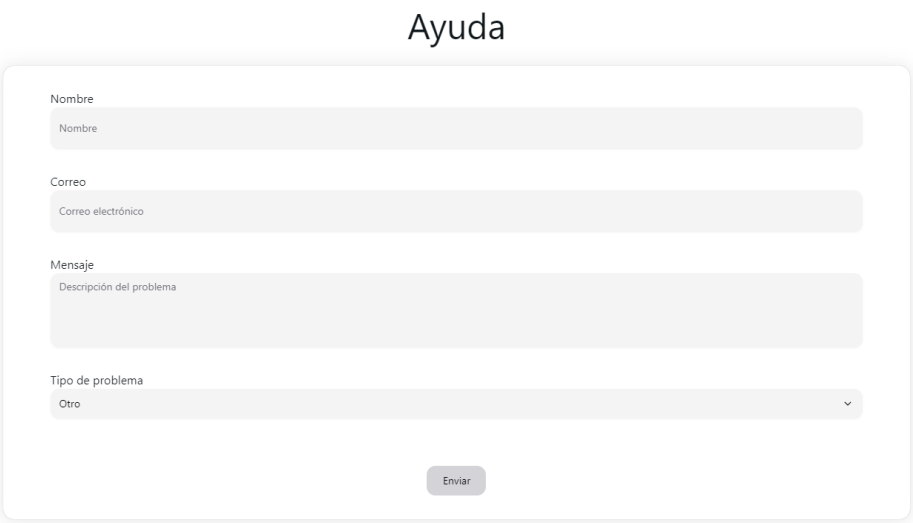

Ilustración 67 Formulario de ayuda.

III. FUNCIONALIDADES DEL CÓDIGO

1. Modelos de la base de datos

En este apartado se van a mostrar todos los modelos de base datos que utiliza Sequelize para hacer las consultas a base de datos. Estos modelos representan en JavaScript como son las tablas de la base de datos y así poder tener un mayor control de que se va a ejecutar sobre ellas.

- **Database**

  Especifica de que lenguaje es la base de datos y mucha de otras configuraciones.

```
const sequelize = new Sequelize(process.env.DATABASE_NAME, process.env.DATABASE_USER, process.env.DATABASE_PASSWORD, {
    1 reference
    host: process.env.DATABASE_HOST,
    1 reference
    dialect: 'mysql',
    1 reference
    timezone: '+02:00', // Aquí puedes especificar tu zona horaria

    1 reference
    logging: false,
    1 reference
    pool: {
        1 reference
        max: 5,
        1 reference
        min: 0,
        1 reference
        acquire: 30000,
        1 reference
        idle: 10000
    }, <- #10-15 pool:
    no references found for waitForConnections
    waitForConnections: process.env.DATABASE_WAITFORCONNECTIONS === 'true',
    no references found for connectionLimit
    connectionLimit: parseInt(process.env.DATABASE_CONNECTIONLIMIT),
    no references found for queueLimit
    queueLimit: parseInt(process.env.DATABASE_QUEUELIMIT)
}); <- #4-19 const sequelize = new Sequelize
```

Ilustración 68 Uso de sequelize.

- **Marcas:** Así son los modelos de las tablas que utiliza Sequelize

```
class Brand extends Model { }
Brand.init({
    no references found for id_brand
    id_brand: {
        45 references
        type: DataTypes.INTEGER,
        15 references
        primaryKey: true,
        10 references
        autoIncrement: true
    },
    no references found for brand_name
    brand_name: {
        45 references
        type: DataTypes.STRING,
        2 references
        unique: true
    }
}, {
    14 references
    sequelize,
    13 references
    modelName: 'Brand',
    13 references
    tableName: 'brands',
    13 references
    timestamps: false
});
```

Ilustración 69 Modelo de marcas.

2. Funcionalidades de los modelos

En este apartado vamos a mostrar cómo funcionan todas de las funcionalidades que tiene cada modelo. En este caso vamos a enseñar como se agregan direcciones a los usuarios:

- **Router**

  En este archivo se especifican todas las rutas que va a usar este modelo para poder llamar a las funciones según las peticiones HTTP.

```
router.route('/')
    /**
     * Endpoint para añadir una nueva dirección.
     * Endpoint to add a new address.
     *
     * @route POST /
     * @param {Object} req.body - El objeto con los datos de la nueva dirección.
     *                            The object with the new address data.
     * @returns {Object} 201 - El objeto de la dirección añadida. | The added address object.
     * @returns {Error} 500 - Error al añadir la dirección. | Error when adding the address.
     */
    .post(async(req, res) => {
        try{
            const newAddress = req.body;
            newAddress.user_id = req.user.userId;
            const address = await addAddress(newAddress);
            no references found for message | 1 reference
            res.status(201).json({ message: 'Dirección añadida correctamente', address });
        }catch(error){
            no references found for message
            res.status(500).json({ message: 'Error al añadir la dirección' });
        }
    }) <- #16-25 .post
```

Ilustración 70 Router.

- **Controller**

  La función a la que llama el archivo router, está localizada en este archivo. En este archivo se realiza toda la funcionalidad de las peticiones pertinentes. Si está función necesita datos de la base de datos, se conecta a un archivo modelo que contiene las funciones de Sequelize de cada modelo.

```
/**
 * Agrega una nueva dirección.
 * Adds a new address.
 *
 * @param {Object} newAddress - El objeto de dirección que se quiere agregar. | The address object to be added.
 * @returns {Promise<Object>} El objeto de dirección agregado. | The added address object.
 * @throws {Error} Si ocurre un error al intentar agregar la dirección. |
 *                 If an error occurs trying to add the address.
 */
const addAddress = async (newAddress) => {
    try {
        const address = await model.insertAddress(newAddress);
        return address;
    } catch (error) {
        console.log(`Error al intentar agregar la dirección ${errorDisplay}`, error);
    }
}; <- #12-19 const addAddress = async (newAddress) =>
```

Ilustración 71 Controller.

- **Model**

  Este archivo contiene las peticiones a la base de datos o las inserciones a la misma. En este caso inserta los datos en la base de datos.

```
/**
 * Inserta una nueva dirección para un usuario.
 * Inserts a new address for a user.
 *
 * @param {Object} address - El objeto de dirección que se quiere insertar. | The address object to be inserted.
 * @returns {Promise<Object|null>} El objeto de dirección insertado, o null si no se proporcionó una dirección. |
 *                                 The inserted address object, or null if no address was provided.
 * @throws {Error} Si ocurre un error al intentar insertar la dirección. | If an error occurs trying to insert the address.
 */
1 reference
insertAddress = async (address) => {

    try {
        if (!address) {
            return null;
        }
        const existingAddresses = await UserAddress.findAll({
            8 references
            where: {
                no references found for user_id
                user_id: address.user_id
            }
        }); <- #71-75 const existingAddresses = await UserAddress.findAll
        if (existingAddresses.length === 0) {
            address.is_default = "1";
        }else{
            address.is_default = "0";
        }
        return await UserAddress.create(address);

    } catch (error) {
        console.log(error);
        console.log(`Error al intentar insertar la dirección ${errorDisplay}`, error);
    }
}; <- #65-90 insertAddress = async (address) =>
```

Ilustración 72 Model.

CONCLUSIÓN

Después de todas estas semanas de desarrollo, nos hemos dado cuenta de que para desarrollar la aplicación de estas características que teníamos en mente desde un principio, necesitaríamos unas cuantas semanas más. Estamos orgullosos porque hemos conseguido la gran mayoría de los objetivos principales como la creación de los usuarios, creación de los productos, edición de ambos, gestión de roles, gestión de transacciones bancarias, etc. También estamos contentos de todas las tecnologías nuevas para nosotros empleadas como React, NodeJS, ExpressJS, Next, etc.

En conclusión, estamos satisfechos con el trabajo en equipo y creemos que la aplicación con unos pocos cambios más podría ponerse en marcha. Alguno de esos cambios podría ser la gestión de incidencias, una mejor gestión de los pedidos, añadir algún soporte de autenticación como Google, Apple, etc. También creemos que podríamos mejorar la unión de nuestro proyecto con la aplicación de Stripe para tener una mejor gestión del stock, un generador de facturas, gestión de ofertas y cupones; y una mejor gestión de los productos a través de Stripe.

También una de las cosas que hemos intentado desarrollar y al final no ha sido posible es el despliegue de la aplicación a la web. Esta parte no ha sido posible debido a que la gran mayoría de servicios de despliegue de máquinas en la nube no ofrecen una potencia suficiente como para desplegar el servidor de Frontend. Debido a esto, tuvimos que desistir en la idea y debido a esto, las autenticaciones externas como Google, Apple, etc.… ha sido imposible de poder completarlas.

Por todo esto, creemos que con unos pocos cambios creemos que la aplicación podría ser desplegada ya que tenemos un nombre de dominio ya registrado a través de servicio que proporciona nombres de dominio.

Con todo esto, este proyecto nos ha enseñado mucho cómo funciona una web y como es el ciclo de vida del desarrollo del software desde que se idea la aplicación, como se diseñan los bocetos, como se crea una base de datos robusta, creación de los servidores, diseño de estos y casi el despliegue de la aplicación a internet.

WEBGRAFÍA

Index | Node.js v22.2.0 Documentation. (s. f.). https://nodejs.org/docs/latest/api/

Stripe | documentation Web

States, N. L. (2013). *Next generation science Standards: for states, by states*. https://ci.nii.ac.jp/ncid/BB1563791X

Archer, R. (2015). *ExpressJS: Web App Development with Node.js Framework*. https://www.amazon.com/Express-js-Web-Development-Node-js-Framework-ebook/dp/B0182K6MG0

BIBLIOGRAFÍA

Derks, R. (2019). React projects: Build 12 real-world applications from scratch using React, React Native, and React 360. Packt Publishing Ltd.

Gerchev, I. (2022). *Tailwind CSS*. SitePoint Pty Ltd.