UNIVERSIDAD
REY JUAN CARLOS

GRADO EN TECNOLOGÍAS DE LA TELECOMUNICACIÓN

Curso Académico 2019/2020

Trabajo Fin de Grado

APLICACIÓN WEB DE COMERCIO ELECTRÓNICO PARA UNA TIENDA DE ELECTRODOMÉSTICOS (NEVADO)

Autor : Sandra Álvarez Gancedo

Tutor : Dr. Gregorio Robles

Trabajo Fin de Grado

Aplicación web de comercio electrónico para una tienda de electrodomésticos (Nevado)

Autor : Sandra Álvarez Gancedo

Tutor : Dr. Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 2020, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2020

Un esfuerzo total

es una victoria completa

Mahatma Gandhi

II

Agradecimientos

En primer lugar quiero agradecer a mis padres, pues sin ellos y sin su esfuerzo no hubiese sido capaz de cerrar este ciclo. Muchas han sido las noches que habéis estado a mi lado, aguantando mis desvelos, mis nervios y mis miedos. Habéis sido el consuelo y el apoyo que necesitaba en mis momentos más duros, en los que únicamente un simple gesto vuestro hacía levantarme. Sois y sereís mi mejor ejemplo a seguir.

La vida me ha regalado unos segundos padres, mis tíos, los cuales me quieren y me protegen como a una hija. Me habéis acompañado en los momentos más felices, pero también en los más difíciles, incluso cuando a mis padres les faltaban fuerzas para seguir adelante. Simplemente deciros, gracias.

Mi primo, más que un hermano. Cuantos momentos vividos, siempre liándola, en las buenas y las no tan buenas, pero siempre juntos. Nos separamos en esta etapa universitaria, pero nuestras carreras eran tan iguales que nos ha permitido seguir ayudándonos.

Quiero agradecer a mis abuelos, a mi abuelo por enseñarme lo que es la vida. Por enseñarme que de todo lo malo también hay que saber sacar el lado positivo. A mi abuela, porque sé que desde algún rinconcito del cielo ha visto como lograba alcanzar todos mis sueños y mis metas. Pero sobre todo sé, que desde ahí arriba, me ha ayudado a luchar y a salir hacia delante ante el mayor reto de mi vida. Sé que estarás orgullosa de mí.

La universidad me ha aportado muchas cosas buenas, pero gracias a ella, conocí a quien es hoy uno de mis pilares, Alejandro. Se ha convertido en mi mejor amigo, mi confidente, mi compañero de viaje, mi otra mitad. Gracias por todo lo que has hecho y haces por mí. Me has ayudado en la universidad, pues sin ti, no sé si hubiese sido capaz de aprobar aquella asignatura que me traía por el camino de la amargura. Pero, sobre todo, me has ayudado en la vida, te quedaste a mi lado en el peor momento, cuando a todos nos invadía la incertidumbre, cuando ninguno sabía que iba a pasar, ahí estuviste conmigo. Me has dado fuerzas cuando mas

flaqueaba, me has sacado una sonrisa cuando más lo necesitaba, y me has dado entereza para asumir y afrontar lo que me estaba pasando. Ha habido muchos momentos malos durante estos 5 años, pero juntos los hemos superado y así seguiremos haciéndolo.

Agradecer a mis compañeros y amigos de universidad por todos los apuntes y conocimientos intercambiados.

A mis profesores, en especial a Antonio Marqués y Carlos Figueras, pues me ayudaron, me apoyaron y me dieron facilidades para poder sacar adelante la asignatura PDAC, a pesar de mis condiciones. Recuerdo, con cariño, el email de Antonio para comunicarme que había conseguido aprobar la asignatura y felicitarme por ello, pues sabía del esfuerzo realizado.

Quiero hacer mención especial a mi tutor de proyecto, Gregorio Robles, por darme ánimos para terminar el trabajo fin de grado. Pero, sobre todo, gracias por ser paciente, por entender y comprender mi situación.

Y, por último, no me puedo olvidar de todos los médicos y enfermeras que me han atendido durante estos cinco años, puesto que, sin su esfuerzo, su investigación y su predisposición, no hubiese podido terminar esta etapa.

Agradecer a la vida, porque a pesar de todo he tenido suerte. Suerte por tener a la gente que tengo a mi lado, por haber podido estudiar, pero, sobre todo, suerte porque no todos pueden decir que han superado una enfermedad, como es un linfoma. Y, por qué no, gracias a ti, linfoma, porque me has enseñado a disfrutar y vivir la vida al máximo y a dar importancia a las cosas que realmente las tienen.

Resumen

El proyecto tiene como objetivo principal desarrollar una aplicación web de comercio electrónico para la venta de electrodomésticos.

Gracias a aplicaciones como la que se desarrollada en este trabajo, pequeños comercios pueden ampliar su mercado y llegar a más gente, pues simplemente accediendo a la web se puede ver la variedad de productos que hay en el catálago, conocer su precio y realizar la compra sin tener la necesidad de ir a la tienda física.

Destacar que además de la venta de electrodomésticos, la aplicación permite al administrador, dueño del comercio, gestionar los clientes, los productos, las ofertas, los pedidos y las facturas. Pensando en está gestión, se implementa un sistema de importación y exportación de ficheros excel que permite al administrador cargar productos de manera masiva sin tener que ir uno a uno, agilizando el trabajo del administrador.

La aplicación está diseñada para que cualquier usuario pueda navegar en ella, sin ningún tipo de dificultad. Se han adaptado las distintas vistas a las resoluciones de pantalla más comunes, de tal forma que se pueda manejar tanto en un móvil como en un ordenador, sin que se produzca ningún cambio de usabilidad. Esto ha propiciado que para el desarrollo de la aplicación se hayan elegido las siguientes tecnologías: HTML5, CSS3, Bootstrap, Angular 7 y TypeScript (para la parte del cliente) y NodeJS, MongoDB y JavaScript (para la parte del servidor).

Summary

The main objective of the project is to develop an e-commerce web application for the sale of household appliances.

Thanks to applications like the one developed in this work, small shops can expand their market and reach more people, because simply by accessing the web you can see the variety of products in the catalogue, know their price and make the purchase in a simple way and without having to go to the physical store.

It is worth mentioning that in addition to the sale of household appliances, the application allows the administrator, owner of the shop, to manage clients, products, offers, orders and invoices. Thinking about this management, an import and export system of excel files is implemented that allows the administrator to load for example, products in a massive way without having to go one by one, making the administrator's work faster.

The application is designed so that any user can navigate it, without any difficulty. To do this, accessibility criteria have been taken into account and the different views have been adapted to the most common screen resolutions, so that it can be used both on a mobile and a computer, without any change in usability. This has led to the following technologies being chosen for the development of the application: HTML5, CSS3, Bootstrap, Angular 7 and TypeScript (for the client side) and NodeJS, MongoDB and JavaScript (for the server side).

Índice general

Índice de figuras

Capítulo 1

Introducción

El comercio es el intercambio de mercancías y géneros en el mercado de la compra-venta.

Si nos remontamos a la antiguedad, se puede ver como los primeros hombres ya usaban un sistema de trueque para intercambiar pieles, alimentos o animales. Este sistema de trueque ha ido evolucionando conforme iba avanzando la humanidad, surgiendo el comercio tal y como se conoce actualmente.

Gran parte de esta evolución ha venido dada por el avance de la tecnología, ya que no ha supuesto un gran cambio en el concepto de comercio pero sí en la manera de comercializar. Hace unos años no se entendía el comercio sin tener que desplazarse a una tienda física.

Con la aparicieron de Internet, aparecieron las primeras transacciones comerciales online. Fue en el año 1981 cuando se realizó la primera de todas ellas. A pesar de ello, no empezó a adquirir popularidad entre las grandes compañias de comercio hasta el año 1994, cuando NetScape implantó SSL, que proporcionaba mayor seguridad en las transacciones y, cuando empezaron a surgir los primeros servicios de terceros para el procesamiento en línea de tarjetas de crédito. Muchas de estas compañias invirtieron grandes fortunas, pensando en el boom que tendría este tipo de comercio. Pero no fue hasta el año 2004 cuando empezó a adquirir popularidad, ya que la aparicion del Consejo de Normas de Seguridad para las tarjetas de pago daba mayor seguridad a la hora de realizar las compras.

A pesar de todo, las personas seguían siendo muy reticentes a comprar por internet por el miedo a que les clonasen sus datos bancarios. La creación de pasarelas de pago totalmente seguras, como la implantación, por parte de las entidades financieras, de tarjetas de crédito recargables hizo que la gente fuese perdiendo el miedo y que el comercio electrónico despe-

gará como esperaban las grandes compañias. Además, el uso de los dispositivos móviles como smartphones o tablets así como la aparición de las redes sociales, donde te permiten publicar anuncios para promocionar productos y servicios ha provocado que el comercio no se entienda sin el comercio electrónico. Teniendo en cuenta la importancia de los dispositivos móviles en este tipo de ventas onlines, se entiende que todas las tiendas quieran adaptar sus aplicaciones web a estos dispositivos y plataformas.

El comercio electrónico, además de suponer un canal adicional de ventas, tiene la ventaja de permitir nuevos mercados y ampliar posibilidades de negocio, lo que hace que aquellas tiendas pequeñas en las que todavía no se ha implatado este tipo de comercio quieran hacerlo. De esta necesidad, surgió la idea de realizar este proyecto. Pues, me permitía ampliar mis conocimientos tecnológicos.

1.1. Estructura de la memoria

La memoria comienza con un resumen, el cual explica de manera breve en que consiste el proyecto así como las tecnologías que se han usado para desarrollarlo. A continuación, se sitúan los agradecimientos y los dos índices, el índice general y el índice de figuras. Por último, van los 6 capítulos en los que se divide la memoria y los apéndices.

- **Capítulo 1: Introducción.** Se describe de manera general en que consiste el proyecto. Especificándose cuál es la estructura de la memoria.
- **Capítulo 2: Objetivos y motivaciones.** Se describen los objetivos que se pretenden alcanzar con el desarrollo y finalización del proyecto. Se detallan las motivaciones por las cuales se ha decidido realizar una aplicación web de comercio electrónico.
- **Capítulo 3: Estado del arte.** Se explican las teconologías usadas en el desarrollo de la aplicación.
- **Capítulo 4: Diseño e implementación.** Se describe de manera detallada la arquitectura, el diseño e implementación del sevidor, el diseño de la base de datos y el diseño e implementación del cliente.
- **Capítulo 5: Resultados.** Se analiza si se ha conseguido alcanzar el resultado esperado.

- **Capítulo 6: Conclusiones.** Se detallan las conclusiones obtenidas con el desarrollo de la aplicación, los conocimientos que se han afianzado y adquirido. Y, por último, las posibles líneas futuras que se podrían desarrollar para mejorar la aplicación.

- **Apéndice A: Manual de instalación.** Se explica todo lo que se debe instalar para que la aplicación funcione correctamente en cualquier equipo, siempre en un entorno local.

- **Apéndice B: Manual de usuario.** Se describen los pasos que el cliente debe seguir para la finalización de la compra online.

Capítulo 2

Objetivos

2.1. Objetivo general

El objetivo principal de este trabajo Fin de grado es el desarrollo de una aplicación web de comercio electrónico.

A través de la aplicación se podrán realizar compras de productos electrónicos sin necesidad de ir a una tienda física y, se llevará a cabo la administración de dicha tienda online por parte del dueño o administrador del comercio.

2.2. Objetivos específicos

Para cumplir con el objetivo principal es necesario alcanzar los siguientes objetivos especificos:

- La aplicación web debe ser responsive, esto quiere decir, que se debe adaptar a cualquier tipo de resolución de pantalla sin perder funcionalidad. Pues, los clientes realizarán sus compras a través de diferentes navegadores y dispositivos móviles.
- La interfaz de la aplicación web deber ser sencilla, de tal forma que realizar una compra sea rápido y sencillo para cualquier usuario, incluso para aquellos que no tienen ningún tipo de conocimiento en informática y comercio electrónico.
- Puesto que se van a tratar datos personales susceptibles, la aplicación debe cumplir con los requisitos básicos de protección de datos. Además, debe ser robusta y segura a la hora

de realizar los pagos.

- La competitividad con otras aplicaciones web de comercio electrónico de electrodomésticos vendrá determinada por una interfaz atractiva y por funcionalidades como la calificación y valoración de los productos por parte de los clientes.
- La aplicación debe ser lo más modulable posible. Esto permitirá que ante un cambio en los requerimientos y necesidades de la tienda, la aplicación se adapte de manera fácil y rápida.
- Diseñar un sistema de administración que permita al dueño del comercio llevar un control de los clientes, los productos, los pedidos y las facturas. Así como, de los proveedores de productos y del transporte de los mismos.
- Implementar un sistema de importación y exportación de ficheros excel que permita el manejo de grandes volúmenes de datos.
- Implementar un sistema de mensajería que permita la comunicación entre los clientes y el administrador.
- Generación de las facturas en PDF para que se puedan enviar junto a los productos.

2.3. Motivaciones

En la actualidad, las tiendas online están en pleno auge y, son un pilar fundamental para el ámbito comercial, cada vez más personas deciden realizar sus compras a través de internet. Este hecho, junto con el motivante de realizar un proyecto ambicioso, fue lo que hizo que desarrollase esta aplicación.

Este desarrollo me permitía poner en práctica la mayoría de los conocimientos adquiridos en el transcurso del grado. También, me permitía adquirir nuevos conocimientos que me servirán para adentrarme en el mundo laboral del comercio electrónico. Suponía una motivación más enfrentarme a nuevos retos aprendiendo tecnologías nuevas como NodeJs, Angular, TypeScript o MongoDB y me servía para demostrarme que soy capaz de resolver cualquier conflicto que se plantee durante el transcurso del desarrollo con esfuerzo y dedicación, aumentando así mi confianza en afrontar nuevos proyectos de cara a mi carrera profesional.

Capítulo 3

Estado del arte

Para el desarrollo de este trabajo fin de grado se han utilizado diferentes tecnologías, las cuales se explicarán a continuación.

3.1. HTML5

HTML5 (Hyper Text Markup), es la quinta y última versión de HTML, lenguaje estándar utilizado para la definición y estructuración de los contenidos de las páginas web.

Se trata de una versión muy diferente a versiones anteriores, ya que su desarrollo se dió por discrepancias entre diferentes miembros del consorcio W3C. En el año 2004, WATWG (Web HyperText Application Technology Working Group) sacó una primera versión, para el asombro de todos. Pero, fue en el año 2009 y, en colaboración con WATWG, cuando W3C comenzó con el desarrollo de la especificación de HTML5.

Fue creado con la intención de sustituir a la versión anterior de HTML (HTML4). Pero, también se pretendía que la aparición de esta nueva versión relevará otros lenguajes como XHTML1 y DOM Nivel 2.

Esta versión, además de incluir nuevas etiquetas y atributos, proporciona una plataforma de desarrollo para la creación de aplicaciones web complejas. Entre las nuevas funcionalidades se encuentran los siguientes puntos:

- Incorporación de elementos multimedia para reproducir sonidos y videos desde el propio navegador. Estos elementos multimedia, son las etiquetas $< audio >$ y $< video >$.

- Integración de gráficos vectoriales escalables (SVG).
- Incorporación del elemento $< canvas >$, el cual permite dibujar y recrear animaciones en el navegador.
- Almacenamiento local en el lado del cliente, con $localStorage$ y $SeassionStorage$.
- Incorporación de etiquetas que permiten la estructuración de los documentos HTML, sin recurrir al uso de $< div >$. Entre estas etiquetas, se encuentran $< header >$ y $< footer >$, muy útiles para la creación de la cabecera y el pie de la página HTML.
- Incorporación de $MathML$ para el manejo de fórmulas matemáticas en los documentos web.

3.2. CSS3

CSS3 (Cascading Style Sheets), es la última versión de CSS. Es un lenguaje de programación que permite definir y crear el aspecto visual de los documentos web.

Surgío de la necesidad de separar el contenido de los documentos HTML5 de su estilo, permitiendo crear documentos web más simples. Además, permite que un mismo documento web tenga varias hojas de estilo, facilitando así, por ejemplo, el diseño responsive, ya que una misma información necesita estilos distintos para los diferentes tipos de pantalla. Del mismo modo, que un documento puede tener varias hojas de estilo, una misma hoja puede ser reutilizada en varios documentos web, simplificando el trabajo de los desarrolladores.

Es un lenguaje simple, que se basa en reglas para definir los aspectos visuales del documento web. Como resumen, su propio nombre es capaz de propocionar la información necesaria que define el lenguaje:

- Cascading: El estilo que se aplica al elemento padre del documento web, también se aplica a los elementos hijos que contenga, propagándose en forma de cascada.
- Style: Es un lenguaje que define el estilo y el aspecto visual de los documentos HTML.
- Sheets: Los estilos se añaden en ficheros, denominados hojas, cuya extensión es .css. Estas hojas pueden ser utilizadas en diferentes documentos HTML, permitiendo así la simplicidad y flexibilidad en los documentos web.

Algunas de las novedades más importantes que introduce esta versión de css son: transiccio-nes, sombras en cajas, gradientes, esquinas redondeadas, múltiples imágenes en fondo, opacidad o tranparencia de los colores, entre otras.

El encargado de definir y mantener las especificaciones de este lenguaje es Worl Wide Web Consortium (W3C).

3.3. JavaScript

JavaScript es un lenguaje de programación interpretado, no necesita ser compilado para poder ejecutarse. Es orientado a objetos y eventos. JavaScript se creó para funcionar en un entorno limitado, es decir, el código JavaScript no puede acceder a los recursos del propio ordenador. A raíz de esto, los navegadores también definieron sus propias normas de seguridad, las cuáles determinan que un script no puede establecer conexión con páginas que pertenezcan a diferentes dominios.

Su nombre puede indicar que tiene relación con Java pero la realidad es muy distinta, ya que tanto la semántica como los propósitos son diferentes. Está basado en prototipos en vez de usar clases, que simulan muchas de las características que tienen las clases en los lenguajes de programación orientados a objetos, como por ejemplo la herencia.

Unas de las principales características que lo difiere de otros lenguajes, es el ámbito de las variables. En Javascript, este ámbito es la función donde han sido declaradas y no el bloque en el cuál se declaran. Con la versión 6 se introdujo esta mejora, permitiendo que la variable fuese usada en el bloque,siempre y cuando se definiese utilizando el término let.

Es un lenguaje imperativo y estructurado.También, es débilmente tipado y dinámico, es decir, una misma variable puede adoptar dos tipos de datos diferentes en distintos momentos, sin tener que redefinir la variable. Está formado en su totalidad por objetos, puediéndose modificar las propiedades de dichos objetos en tiempo de ejecución, dontándolo así del dinamismo que lo define.

Es un lenguaje rápido y ligero, por lo que se convierte en uno de los lenguajes más usuados en el mundo del desarrollo de aplicaciones web. Su uso, principalmente, está en el lado del cliente pero, con la aparición de tecnologías como NodeJS, el uso de JavaScript en el lado del servidor está sufriendo un repunte significativo. El hecho de ser un lenguaje multiplaforma,

que permite utilizarse tanto en Windows, como en Linux como en Mac, así como en cualquier navegador hace que aumente su popularidad y su uso.

Surgío de la necesidad de los desarrolladores de crear páginas web complejas y dinámicas, ya que HTML únicamente permitía crear documentos web estáticos. Fue desarrollado por Brendan Eich de la compañia NetScape y, su primer nombre fue Mocha. Poco tiempo después, cambiaría su nombre original y pasaría a llamarse LiveScript. En el año 1995, NetScape y Sun MicroSystems (creadores del lenguaje Java) reintroducirían el lenguaje en el mundo de la programación, como Javascript. En el año 1997 se adopta como un estándar ECMA, denominándose ECMAScript. Desde entonces ha ido evolucionando y mejorando, creándose librerías y frameworks que facilitan la programación. En el año 2015, se publicó ECMAScript 2015 ó JavaScript 6, introduciendo características propias de las clases utilizadas en la programación orientada a objetos. La última versión hasta el momento, es la versión 8, segunda versión que tiene un proceso de desarrollo abierto.

3.4. TypeScript

TypeScript es un lenguaje de programación de alto nivel orientado a objetos, de código abierto y libre.

Su desarrollo estuvo a cargo de Microsoft y, en el, participó Anders Helsberg (diseñador y creador de C-Sharp, Delphi o Turbo Pascal). Surgió de la necesidad de los desarrollores de crear aplicaciones robustas y de gran tamaño que, con JavaScript no se podían mantener debido a su escasa escalabilidad.

Su uso está permitido tanto en el lado del cliente, Angular 2, framework de JavaScript, usa TypeScript como lenguaje, como en el lado del servidor con NodeJS.

TypeScript, es un ”Superset”de JavaScript, es un lenguaje escrito encima de otro lenguaje. Esto significa, que TypeScript se compila a JavaScript, permitiendo que cualquier código desarrollado en JavaScript funcione sin tener que modificar el código ni la aplicación. Mejora y arregla algunos problemas que tiene JavaScript y, es por ello, que pone a disposición de los desarrolladores las funcionalidades disponibles en JavaScript ES6 y JavaScript ES7. Añade objetos basados en clase, facilitando la programación orientada a objetos. En su código se puede incluir ficheros de definición que contienen información de las librerías JavaScript. Además,

incluye un tipado estático. El poder definir variables y funciones tipadas sin perder la esencia de JavaScript ayuda al desarrollador a evitar errores en tiempo de ejecución.

Su publicación fue en el año 2012 y actualmente, se encuentra en la versión 2.0, la cual introduce numerosas características con respecto a la primera versión, entre las que destaca la capacidad de evitar la asiganción de varibales con un valor nulo.

Su compilador fue desarrollado en TypeScript, compilado a JavaScript y con licencia Apache 2.

3.5. Bootstrap 4

BootStrap 4, es la versión más actual y estable de Bootstrap. Es el framework más popular de CSS. Es un conjunto de herramientas que permite a los desarrolleres mejorar el aspecto visual de los documentos HTML y de las aplicaciones web de manera rápida y sencilla.

Su nombre original fue Blueprint, fue creado y diseñado por Matt Otto para la compañia Twitter como herramienta de trabajo de uso interno. En el año 2011, Twitter lo libera con licencia MIT y pasa a ser de código abierto.

El uso de este framework en las aplicaciones web permite generar diseños que se adaptan dinámicamente a las diferentes resoluciones de pantalla y a los distintos dispositivos, favorenciendo, el diseño responsive. Para ello, Bootstrap dispone de un sistema de cuadrilla o rejilla estándar de 940 píxeles pero, dispone de cuatro variaciones de tamaño para adaptarse a los distintos dispositivos. Estas cuatro variaciones son sm, md, lg y xl.

Es modular y, consiste en un conjunto de hojas de estilo LESS que combinado con plantillas HTML y con extensiones JavaScript implementan los diferentes componentes de Bootstrap.

Incluye los elementos más usados en los documentos HTML, como son formularios, barras de progreso y de navegación, cuadros, mensajes de alertas o botones con características especiales entre otros. Todos ellos tienen un estilo predefinido, pero que se puede modificar de manera sencilla.

Para usar bootstrap en las aplicaciones web es necesario incluir en el documento HTML la hoja de estilo de Bootstrap CSS. De la misma forma, si se quiere utilizar algunos de los componentes JavaScript predefinidos de BootStrap se debe incluir la librería JQuery de JavaScript.

La versión 4 significa una mejora en diversos componentes permitiendo un diseño más

responsive. Se han modificado componentes como la navegación, que incluye la nueva clase $flexbox$. Se ha creado un nuevo componente Card que sustituye a los componente Wells, Panlles y Thumbnails. Se han modificado los elementos de paginación y las modales. Y, se incorpora el componente Grid. Además, de estas significativas mejoras, se han incluido numerosas clases de uso común.

Es compatible con todos los navegadores modernos, Firefox, Chrome, Safari, Opera, EDGE, Internet Explore 10 pero, no es compatible con las versiones anteriores a internet explorer 10.

3.6. Angular

Es un framework de código abierto diseñado por Google. Permite crear aplicaciones web de una sóla página, lo que se denomina SPA (Single Page Application). Utiliza TypeScript y HTML para el desarrollo de las aplicaciones web.

Angular permite separar el frontend del backend y, sigue un modelo MVC (Modelo-Vista-Controlador), arquitectura de software que separa los datos de la aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos. Además, Angular dota a las aplicaciones de escalabilidad pues, mantiene el código ordenado. De esta forma, las modificaciones y las actualizaciones de las aplicaciones son rápidas y sencillas.

Una de las principales ventajas de este framework y de las páginas SPA es la velocidad de carga entre las diferentes vistas de la aplicación. Cuando se cambia de vista no se recarga la página sino que éstas se cargan de manera dinámica, rápida y reactiva.

La arquitectura de una aplicación Angular se basa en clases de 4 tipos distintos (módulos, componentes, servicios y directivas). Estas clases se identifican a través de decoradores, los cuales permiten cargar los metadatos necesarios que determinan a Angular como debe usar dichas clases.

Módulos.

Los módulos suministran el contexto de compilación de los componentes, es decir, es aquí donde se definen los diferente componentes que va a formar la aplicación, las dependencias, las clases que actúan como servicios en la aplicación y las rutas de navegación que establecen las vistas de la aplicación.

Toda aplicación angular tiene un módulo principal, que es el raíz. Este módulo suele recibir

el nombre de AppModule y, es el que inicia el sistema de arranque de la aplicación.

Al igual que en Javascript, los módulos de Angular permiten importar y exportar funcionalidades proporcionadas por otros módulos, por lo que, una aplicación puede tener más de un módulo y, cada uno de ellos ser independiente. La manera en que se organizan estos módulos ayuda a la realización de aplicaciones más complejas y a la reutilización de código. Además, pemite que la carga inicial sea mínima, ya que cada módulo se carga a petición, es decir, cuando se va a utilizar.

Componentes.

Las aplicaciones Angular tienen un componente que es nexo de unión entre los diferentes componentes que forman la aplicación y, el módelo de objeto del documento de la página (DOM). Los componentes son los que tienen la lógica y los datos de la aplicación. Son los que controlan las diferentes plantillas HTML que se cargan cuando se modifican las URLs dentro de la aplicación. Estas plantillas HTML son las vistas que forman la interfaz de usuario.

De la misma forma que las aplicaciones diseñas con Angular pueden tener más de un módulo, también pueden tener subcomponentes, que se pueden relacionar entre sí mediante dos tipos de vinculación de datos, eventos y propiedades. A través de los eventos, la aplicación responde a las entradas del usuario actualizando los datos. A través de las propiedades, se envían valores calculados de los datos de la aplicación a los HTMLs que forman las vistas. Esta vinculación de datos se puede producir en ambas direcciones, por tanto, igual que los datos de la aplicación pueden modificar los HTMLs, los cambios en el DOM pueden modificar los datos de la aplicación.

El decorador @Component es el que determina que una clase actúe como un componente.

Servicios

Los servicios son clases en las que se definen datos o funcionalidades generales que no están asociadas a una determinada vista. Son usados para compartir datos y operaciones entre componentes. También, es donde se suele realizar toda la operativa de las peticiones a la API de las aplicaciones. Deben llevar el decorador @Injectable, pues es el que permite obtener los metados necesarios para que otras clases puedan inyectar sus dependencias.

Directivas

Las directivas son clases en las que se definen los términos claves que se usan en las plantillas. Cuando se carga una vista, Angular procesa estos términos clave que modifican el HTML

y el DOM de la aplicación. Por tanto, una plantilla, además de utilizar HTML para definir la vista, usa marcas propias de angular que son estas directivas.

Existen dos clases de directivas, las de atributo, que modifican el comportamiento del componente, y las estructurales, que únicamente modifican la apariencia.

Angular dispone de un enrutador, módulo que contiene un servicio para definir las rutas de navegación de la aplicación. Este módulo se ajusta a las convenciones de los navegadores. Es decir, tanto si se pone la URL en la barra de direcciones, como si se pincha un enlace en la aplicación, como si se hace clic en el botón de retroceso o avance, se navegá a la página correspondiente. El enrutador es el que muestra u oculta una vista. Cada vez que se modifica la URL, el enrutador se encarga de añadir la ruta en el historial del navegador, de ahí que los botones de retroceso y avance funcionen. Cada ruta de navegación está asociado a un componente.

Angular dispone de varias versiones, aportando todas ellas mejoras en dicho framework. La primera versión de Angular, se conoce como AngularJS y, es la más distinta a las demás. El resto son actualizaciones de Angular 2. Cuando se definió la segunda versión se decidió modificar el nombre del framework y dejarlo como Angular. La última versión estable es la versión 8.

3.6.1. Angular CLI

Angular CLI (Command Line Interface) es una herramienta de línea de comandos creada por el equipo de Angular. Es muy útil a la hora de iniciar una aplicación web diseñada con Angular, ya que con un sólo comando, se genera el esqueleto de carpetas y archivos necesarios de la aplicación. Además, contiene herramientas predefinidas que ayudan al desarrollo y mantenimiento de este tipo de aplicaciones. Al crear una aplicación web con Angular Cli, dentro de la estructura de archivos, se crea un archivo de configuración, en el cúal se añaden las dependencias necesarias para que la aplicación web compile y ejecute. Este archivo de configuración se va modificando conforme se van creando los componentes, servicios o directivas. Para ello, Angular Cli tiene comandos que permiten crear estas clases de manera sencilla, creando los distintos archivos que conforman un componente, un servicio o una directiva. Entre las herramientas predefinidas destaca el compilador, el sistema de testing y el servidor web.

3.6.2. Angular Material

Angular Material es una librería creada por Google para las aplicaciones web desarrolladas con Angular. Al igual que Boostrap ayuda a diseñar aplicaciones web atractivas, con estilos y formas únicas, ya que contiene multitud de componentes con estilos prefijados. Estos componentes permiten a las aplicaciones web ser rápidas, consistentes y versátiles. Además, con el uso de esta librería las aplicaciones web se adaptan mejor a la mayoría de resoluciones de pantalla y de dispositvos, facilitando el diseño responsive de las aplicaciones. También, con su uso se crean aplicaciones web accesibles, ya que los componentes cumplen con los estándares de accesibilidad. Además, esta librería facilita la internacionalización de la aplicación, puesto que permite la utilización de diferentes idiomas. Es compatible con todos los navegadores modernos.

3.6.3. Angular Editor

Es un componente de Angular, que permite escribir texto con formato y estilo propio dentro de un documento HTML. El estilo y el formato lo determina el usuario. El componente transforma el texto escrito, con el formato y el estilo definido, al lenguaje HTML visualizándose en pantalla conforme el usuario determina. Es compatible únicamente con las últimas versiones de Angular (de Angular 6 en adelante).

3.7. Node JS

Node js fue creado en 2009 por Ryan Dahl como sistema de desarrollo de aplicaciones red escalables. Es un entorno de ejecución orientado a eventos asíncronos que utiliza el motor V8 de Google, desarrollado en C++ y de código abierto, para interpretar el código javascript, del mismo modo que hace el navegador Chrome.

Se basa en sistemas como Event Machine de Ruby y Twisted de Python. Arranca cuando se ejecuta el script de entrada al bucle de inicio de eventos y, finaliza cuando no hay más devoluciones de llamadas callbacks, utilizando así un único hilo. De este modo, cada nuevo evento se añade al bucle de eventos que se ejecuta al inicio, evitando el bloqueo de procesos que se produce en los sistemas de concurrencias basados en la utilización de hilos de los sistemas

operativos. Estos procesos usan un hilo por cada conexión provocando la ineficiencia de los recursos utilizados. A pesar de que Node.js esté pensado para usar un único hilo se pueden crear hilos diferentes o subprocesos a través de Apis propias de Node como API_child.process.fork().

Por otro lado Node es un buen aliado para desarrollar aplicaciones web, ya que a la hora de implementar el protocolo Http para Node se pensó tanto en la latencia de transmisión como en la transimisión por streaming de datos.

3.8. Npm

Es el sistema de administración de paquetes del entorno de ejecución de NodeJs. Contiene un cliente, con el que ejecutando una línea de comando, se instalan los paquetes necesarios para el funcionamiento de los proyectos NodeJs. Además, contiene una biblioteca o base de datos con más de 477000 paquetes disponibles para usarse. Todos estos paquetes están guardados con formato CommonsJs y, contienen un archivo para almacenar metadatos con formato Json. Fue desarrollado por Isaac Z. Schlueter y se desarrolló por completo en JavaScript.

Al instalar un paquete a través de Npm, se instala bajo el directorio del proyecto /.node_modules. Permite instalar con un único comando todas las dependencias del proyecto, siempre y cuando se ejecute el comando en la raíz del proyecto. Npm utiliza el archivo con formato Json, denominado Package.json, para guardar todos los metadatos relevantes a los módulos, paquetes y dependencias del proyecto. En el archivo package.json se guardan las versiones compatibles para los diferentes módulos, permitiendo hacer un versionado del módulo instalado. Gracias a esto, Npm hace que la instalación de proyectos NodeJs, descargados de Git sea sencilla.

3.9. Express.js

Express es el framework más usado para aplicaciones desarrollas con Node.js. Es un framework flexible, rápido, ligero y minimalista. Pero, a pesar de ello, existen numerosos plugins o middlewares compatibles con Express. Estos plugins permiten resolver los problemas que pueden surgir a la hora de implementar aplicaciones con Express. Por todo esto, express sirve de base de otros muchos frameworks de Node.js.

Express permite:

- Crear aplicaciones Node de manera sencilla y rápida, ya que proporciona el esqueleto de las aplicaciones.
- Modificar ajustes de la aplicación como la localización de las plantillas que se van autilizar o el puerto en el que se va a ejecutar.
- El uso de cualquier base de datos, siempre y cuando la base de datos esté soportada por Node.
- La gestión de cookies, usuarios y sesiones a través de su middleware.
- Configurar y establecer las rutas entre el front y la base de datos.
- Mejorar las funcionalidades HTTP que tiene Node y, que están basadas en Connect. Existen métodos que determinan la función a usar índicando únicamente el verbo de la petición HTTP.

3.10. Nodemon

Nodemon es una herramienta que permite a los desarrolladores de Node implementar sus aplicaciones de manera más sencilla, ya que amplía y mejora funcionalidades propias de Node.

Cuando se programa en Node, los cambios realizados en el código fuente no se ven reflejados inmediatamente, hasta que no se reinicia el servidor de Node no se actualiza la aplicación, por lo que cada vez que se prueba un cambio, habría que parar y arrancar el servidor, relentizando el desarrollo de los proyectos. Nodemon reinicia de manera automática el servidor cada vez que detecta un cambio en el código fuente, solucionando el incoveniente anterior. Los cambios se ven reflejados en la aplicación inmediatamente, permitiendo programar y probar nuevas funcionalidades sin demoras.

3.11. MongoDB

MongoDB es una base de datos no relacional, una base de datos NoSQL, que se basa en documentos para guardar la información.

Es una base de datos gratuita y multiplataforma (Linux, Windows, Os X y Solaris). Las distintas versiones se lanzaban bajo la licencia AGPL pero, a partir de octubre de 2018, se publican bajo la licencia pública SSPL, aunque tiene una línea comercial que contiene caractarísticas más avanzadas.

A diferencia de las bases de datos relacionales, MongoDB almacena sus datos en documentos y no en tablas. Estos documentos son de esquemas libres, lo que quiere decir que cada entrada puede tener una estructura de datos distinta, con campos que no se tienen porque repetir en otro registro. Por eso, MongoDB tiene mayor flexibilidad y escalabilidad que las bases de datos relacionales. Los documentos se guardan en formato BSON (Binary JSON), lo que permite guardar índices de array, longitudes de campos y datos relevantes, que hacen las búsquedas más rápidas y eficientes. A pesar de que se almacenan en BSON, los desarrolladores siempre trabajarán con documentos JSON.

Los documentos se agrupan en colecciones y, estas colecciones son lo que serían las tablas en las bases de datos relacionales.

MongoDB permite indexar cualquier atributo. Por defecto, el índice tiene un formato similar a un valor UUID hexadecimal. Está formado por una semilla basada en la MAC de la interfaz de la red de la máquina y por una marca de tiempo, que ocupa las primeras posiciones del índice. Gracias a esta marca de tiempo, los datos se ordenan por orden de creación. Todos los documentos tienen un índice, ya que es el único campo obligatorio dentro de un documento MongoDB.

MongoDB permite realizar:

- Consultas Ad Hoc, es decir, se puede realizar búsqueda de datos a través de campos, de rangos o a través de expresiones regulares. Estas consultas nos pueden devolver un dato concreto o pueden devolver el documento entero.

- Balanceo de carga, para ello realiza una partición horizontal de los datos. Es el desarrollador quien decide como van a ser distribuidos los diferentes datos. Para llevar a cabo este balanceo de carga, MongoDB puede ejecutarse en diferentes servidores.

- Almacenamiento de archivos.

- Operaciones similares al $GroupBy$ de SQL, a través de agregaciones. Estas agregaciones se implementan como pipeline en el que se van transformando los datos a través de etapas

hasta conseguir el dato deseado. Contiene una operación MapReduce que se suele utilizar en estas agregaciones.

- Consultas a través del lenguaje Javascript.

Como desventajas, MongoDB únicamente puede realizar operaciones concurrentes de escritura entre documentos distintos, ya que cada vez que se realiza una operación de escritura se bloquea el documento. Y, además, puede tener problemas de rendimiento si los datos superan los 100GB.

3.12. Mongoose

Mongoose es un ODM (Object Document Mapping) de MongoDB, es decir, sirve para mapear objetos de javascript a documentos propios de MongoDB.

Es una librería de javascript que se instala a través de un único comando npm en los proyectos que utilizan Node y MongoDB.

Para mapear los objetos a documentos, lo que hace es crear esquemas con datos fuertemente tipados. Estos esquemas se traducen en un modelo Mongoose y, es este modelo Mongoose el que se traduce en el documento MongoDB.

Contiene una infinidad de funciones para realizar búsqueda de datos, validación de datos, guardado y eliminado de datos, etc. Todas estas funciones se basan en las funciones más comunes que Mongo tiene para realizar estas operaciones.

Permite crear esquemas muy flexibles, de tal forma que se puede organizar la información de diferente manera, según convenga al desarrollador. Tanto es así, que permite crear esquemas con referencias a otros esquemas, asemejándose de esta forma, a lo que sería una relación entre tablas de una base de datos relacional, como por ejemplo, MySQL.

3.13. Bcryptjs

Bcrypt es una librería que se usa para la encriptación de contraseñas y datos importantes. Fue creada por Niels Provos y David Maxieres y está desarrollada con Javascript.

Utiliza un valor aleatorio, denominado salt, para la generación del hash del dato que se desea encriptar. Este valor se guarda con el hash en la base de datos. Por eso, para el mismo valor se

obtiene diferentes hashes. Gracias a esto, se evitan los ataques de fuerza bruta o los ataques denominados Rainbow table (tablas de textos y sus hash asociados). Bcrypt permite elegir el valor del SaltRound, ya que a mayor valor numérico del salt mayor seguridad, pero a mayor seguridad mayor coste de tiempo a la hora de encriptar los datos. Por lo tanto, lo idóneo es elegir un salt lo suficientemente grande para que no se pueda obtener el dato cifrado por fuerza bruta pero no lo suficientemente alto como para que el procesamiento se demore mucho tiempo. Por defecto, Bcrypt pone un salt de 10, un valor intermedio para que no se produzcan los dos problemas principales.

Bcrypt permite realizar comparaciones entre valores. Para realizar la comparación no descifra el dato a comparar, como se hace en otros sistemas, sino que cifra el valor con el salt del dato cifrado.

3.14. Passport.js

Passport es un middleware que permite la autenticación en aplicaciones Node. Es de código abierto. Se puede incorporar en cualquier aplicación basada en Express, ya que trabaja en conjunto con Express y Connect. Permite gestionar y administrar las sesiones de los usuarios. Las características que definen Passport son:

- Tiene numerosas estrategias de autenticación. Por ejemplo, permite autenticarse utilizando bases de datos, en las que se almacenan las sesiones o puede autenticarse con el inicio de sesión único de OpenID y OAuth, usados por ejemplo, en Facebook.
- Permite desarrollar nuevas estrategias de autenticación.
- Permite usar sesiones persistentes.
- Gestiona el proceso de autenticación indicando si la autenticación ha sido satisfactoria.

3.15. Git

Git es un software de control de versiones. Fue diseñado por Linus Torvalds. Es un software libre que se distribuye a través de la versión 2.0 de GNU.

Git permite el trabajo en equipo mediante su sistema de ramas, que permite el desarrollo no lineal de los proyectos. En cada rama se puede trabajar sobre una funcionalidad distinta, agilizando de esta forma el desarrollo del proyecto y, haciendo que el trabajo en equipo sea más sencillo y óptimo. Además, permite gestionar de manera eficiente aplicaciones y proyectos de gran envergadura.

Los cambios en los proyectos se realizan de manera inmediata, ya que cada desarrollador tiene una copia en local (en su equipo) del proyecto. Esto permite navegar por el historial de cambios sin necesidad de conectarse a un servidor. Es decir, se puede trabajar en local hasta finalizar una tarea y, posteriormente subirlo al repositorio. Así, todo el equipo obtiene el cambio en su proyecto local.

Cuando se usa Git en un equipo de trabajo se definen flujos que determinan como se va a realizar el desarrollo del proyecto. Es habitual que exista una rama master, que será la que se depliegue en producción. De esta rama master se obtiene una nueva rama, denominada develop. En esta rama se añaden los cambios y las funcionales nuevas. Se mezcla con la rama master una vez testeados los cambios. Para funcionalidades específicas, se crean ramas que se obtienen de la develop, denominándose features. De esta forma, se permite el desarrollo en paralelo de los distintos integrantes del grupo. Una vez que la funcionalidad esté finalizada y probada se junta con la rama develop. Cuando el resto del equipo actualiza su rama develop, obtienes los cambios de las nuevas funcionalidades sin que su trabajo se vea afectado. Cuando hay un error en producción que deber ser solucionado inmediatamente, se crea una rama a partir de la master, llamada hotfix, donde se desarrollan las soluciones al error. Una vez implementada la solución al error, se mezcla con la rama master.

3.16. Visual Studio Code

Visual Studio Code es un editor de código fuente, lanzado en 2015 por Microsoft. Es multiplataforma, pudiéndose instalar en Windows, Linux y Macos

Soporta una gran cantidad de lenguajes de programación. El resaltado de sintaxis, la capacidad de finalización del código que se escribe, la refactorización del código, el modo depuración, así como la integración con Git facilita el desarrollo de aplicaciones.

Para hacer más atractiva su interfaz y para facilitar su uso, permite personalizar los temas

del editor y los atajos del teclado.

Es un editor de código abierto bajo licencia MIT, lo que quiere decir que se puede descargar de GitHub y realizar cualquier modificación en su código. Se basa en Electron, framework utilizado para el desarrollo de aplicaciones de escritorio usando tecnologías web, utiliza Chromium para el motor gráfico y NodeJs para ejecutar JavaScript.

3.17. Mongo Compass

Mongo Compass es una herramienta gráfica, multiplataforma (Linux, Mac y Windows), que permite a los desarrolladores interactuar con un base de datos MongoDB. Las principales ventajas que tiene su uso son:

- Permite analizar los esquemas, visualizando la estructura de sus documentos, el tipo y los rangos de los campos de dichos documentos.
- Permite gestionar de manera sencilla los índices.
- Se pueden realizar todas las operaciones CRUD de manera visual. De la misma forma, se pueden realizar búsquedas o aplicar filtros para encontrar datos dentros de los documentos del esquema.
- Ayuda a resolver problemas de rendimientos, ya que se pueden obtener análisis y gráficos de lo que tarda una query en ejecutarse.
- Se pueden crear agregaciones de manera intuitiva y sencilla, gracias a los esqueletos de código y al autocompletado. Estas agregaciones se exportan a código fuente para que se incorporen dentro del código de la aplicación que usa MongoDB.

Capítulo 4

Diseño e implementación

4.1. Arquitectura general

La aplicación implementada en este trabajo tiene una arquitectura $Three - Tier$, es decir, una arquitectura de tres niveles. El primer nivel de esta arquitectura es la capa de presentación o cliente. Este nivel está compuesto por las diferentes vistas de la aplicación que forman la interfaz de usuario. Es el encargado de recoger la información que produce el usuario al interacturar en la aplicación y, el que envía, a través de peticiones, la información al segundo nivel. El segundo nivel, es el servidor donde se encuentra la lógica de negocio. Se encarga de recibir todas las peticiones de usuario enviadas por el cliente y, de su posterior respuesta. Es el único nivel que se comunica con el resto, puesto que, también puede comunicarse con el tercer nivel, capa de acceso a base de datos o servidor de base de datos. Este tercer nivel es el que maneja la información. En la figura 4.1 se representa un ejemplo de este tipo de arquitectura.

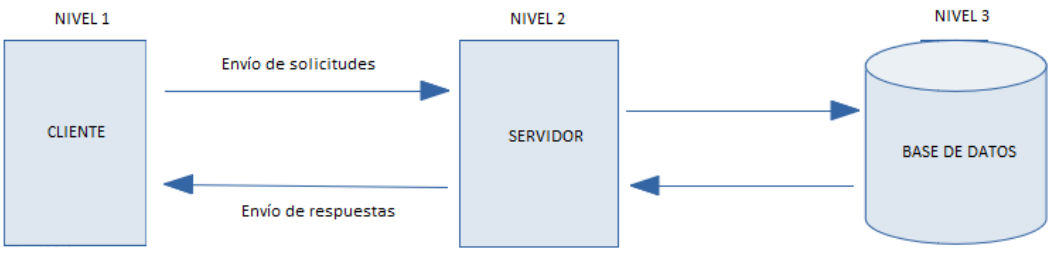


Figura 4.1: Ejemplo arquitectura Three-Tier.

Existen distintos tipos de arquitectura $Three - Tier$. La escogida para esta aplicación es

la denominada $MEAN$. Este tipo de arquitectura se denomina así porque utiliza MongoDB, Express, Angular y NodeJS para el desarrollo de las aplicaciones. En la figura 4.2 se puede observar como es la comunicación entre el cliente y el servidor en una arquitectura de tipo MEAN.

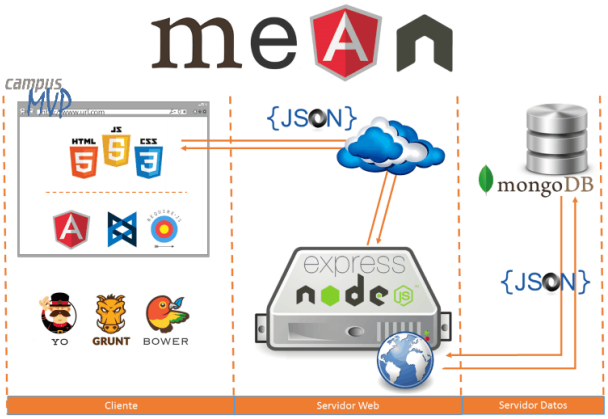


Figura 4.2: Ejemplo arquitectura de tipo MEAN.

Para la implementación del cliente se ha usado Angular, ya que permitía crear una aplicación Single-Page, haciendo uso del modelo-vista-controlador(MVC). Este tipo de aplicaciones se caracterizan por la rapidez y la fluidez de navegación, ya que no es necesario recargar la página cada vez que se cambia de vista. Angular utiliza TypeScript como lenguaje de programación, por lo que el cliente está programado con esta tecnología, además, de usar HTML5, CSS3 y Bootstrap. El cliente utiliza el servidor que proporciona Angular-Cli para servir los recuersos.

El servidor se ha implementado usando NodeJS y Express, consiguiendo así, crear una aplicación rápida y escalable. Se ha programando utilizando Javascript. Por su parte, el servidor se encuentra alojado en el servidor proporcionado por NodeJS.

Para establecer la comunicación entre el cliente (servido por Angular-Cli) y el servidor (servido por NodeJs) se ha recurrido a la utilización de un proxy. En Javascript, cuando se hacen peticiones a un dominio distinto al del origen del script, como es el caso, surge el problema de Cross Domain, el cual impide que se establezca la conexión entre el cliente y el servidor. En el caso de la aplicación, el proxy actúa de intermediario reenviando todo el tráfico que recibe Angular-Cli en el contexto $/nevado$ al servidor e intercepta las respuestas que el servidor envía al cliente. Con Angular-Cli se configura de manera sencilla, ya que para configurarlo solo se necesita un fichero de configuración y ejecutar su servidor a través del comando: *ng-serve –*

proxy-config proxy.conf.json.

La aplicación necesita una base de datos para almacenar tanto los datos de los clientes, como los datos de los productos, facturas y pedidos. Se ha utilizado una base de datos no relacional, concretamente MongoDB.

4.2. Modelo de la base de datos

El gestor de base de datos utilizado es MongoDB, que es una base de datos no relacional. Se ha utilizado este tipo de base de datos por la flexibilidad que proporciona a la hora de trabajar con documentos. Además, como se ha comentado en la sección 3.11, los documentos almacenados no tienen porque tener las mismas propiedades, lo que resulta muy útil para el almacenamiento de los productos, ya que cada producto tiene propiedades diferentes. Están indicadas en aplicaciones en las que el volumen de datos crece constantemente. En una tienda online, tanto el número de usuarios como el número de pedidos está en constante crecimiento. El formato de los documentos de las distintas colecciones de la base de datos, está definidos en el subdirectorio modelos del servidor. Las colecciones de la aplicación se describen a continuación:

- **Usuarios**: Colección donde se almacenan los usuarios registrados de la tienda online. Los campos que tienen sus documentos son:
  - **correo**: Email con el que se registra el usuario, debe ser único y siempre debe de estar presente en los documentos guardados. (String).
  - **rol**: Perfil del usuario registrado, puede ser cliente o admin. Dependiendo de este perfil se pueden visualizar unas vistas u otras. (String)
  - **bloqueado**: Indica si un usuario se ha bloqueado. El administrador de la tienda online puede bloquear usuarios en caso de detectar anomalias en su compartamiento. (Boolean)
  - **confirmacion**: Indica si un usuario está confirmado. Si el usuario no está confirmado no podrá realizar compras. (Boolean)
  - **contraseña**: Contraseña encriptada introducida por el usuario para entrar en su cuenta. Es un campo que siempre debe ir relleno. (String)

    - **clientes**: Identificador del usuario en la tabla clientes. Cuando un usuario se registra se crea un documento nuevo en la colección clientes. (ObjectId)

- **Clientes**: Colección donde se almacena la información personal de un usuario registrado. Los campos son:

    - **nombre**: Nombre del usuario registrado. (String)
    - **apellidos**: Apellidos del usuario registrado. (String)
    - **teléfono**: Teléfono personal donde dirigirse en caso de tener problemas con su pedido. (String)
    - **fechaNacimiento**: Fecha de nacimiento del usuario registrado. (String)
    - **direccionFactura**: Dirección de facturación. Calle, número, portal, escalera, piso y letra. (String)
    - **codigoPostalFactura**: Código postal de la dirección de facturación. (String)
    - **puebloFactura**: Municipio de la dirección de facturación. (String)
    - **provinciaFactura**: Provincia de la dirección de facturación. (String)
    - **direccionEnvio**: Dirección donde se envía el pedido. Sólo tendrá valor si la dirección de envío es diferente a la dirección de facturacion. (String)
    - **codigoPostalEnvio**: Código postal de la dirección de envío. Sólo tendrá valor si la dirección de envío es diferente a la dirección de facturacion. (String)
    - **puebloEnvio**: Municipio de la dirección de envio. Sólo tendrá valor si la dirección de envío es diferente a la dirección de facturacion. (String)
    - **provinciaEnvio**: Provincia de la dirección de envío. Sólo tendrá valor si la dirección de envío es diferente a la dirección de facturacion. (String)
    - **pedidos**: Número de pedidos realizados por el usuario en la tienda. (Number)

- **Artículos**: Colección donde se almacena la información de los productos.

    - **codigoArticulo**: Identificador del artículo. Es único y debe estar presente en todos los documentos guardados. (String)

- **categoria**: Grupo en el que se encuadra el artículo. (String)
- **modelo**: Modelo del artículo. (String)
- **marca**: Marca del artículo. (String)
- **pvpTarifa**: Precio máximo del artículo (Number)
- **pvp**: Precio del artículo con IVA. (Number)
- **estado**: Indica si el producto es visible o no en el catálogo. (Boolean)
- **stock**: Existencias del artículo en el almacén. (Number)
- **imagen**: Ubicación donde se encuentra la foto del producto en el servidor. (String)
- **características**: Descripción del artículo con las características principales. (String)
- **otros**: Información adicional del artículo. (String)
- **fechaPublicacion**: Fecha en la que se guarda por primera vez el artículo en el sistema. (Date)
- **fechaModificacion**: Fecha en la que algún campo del documento es actualizado. (Date)
- **autor**: Nombre de quien agrego el artículo al sistema. (String)
- **ventas**: Número de veces que el producto ha sido comprado. (Number)
- **ofertas**: Identificador del descuento que se le aplica al producto si estuviese en oferta. (ObjectId)
- **envioGratuito**: Indica si el artículo se envía sin ningún coste. (Boolean)
- **estrellas**: Número de estrellas puestas en las reseñas. (Number)
- **Usuarios**: Número de usuarios que han escrito una reseña sobre el producto. (Number)
- **comentarios**: Array de identificadores de las reseñas del producto. (ObjectId)

- **Categorías**: Colección donde se almacenan las diferentes categorías donde se agrupan los productos.

  - **nombre**: Nombre de la categoría. Es único y debe estar presente en todos los documentos. (String)

  - **imagen**: Ubicación donde se encuentra la foto, que representa la categoría, en el servidor. (String)
  - **categoriaGeneral**: Grupo donde se encuadra la categoría. (String)

- **Marcas**: Colección donde se almacenan los proveedores, marcas, de los artículos.
  - **nombre**: Nombre de la marca. Es único y debe estar presente en todos los documentos. (String)
  - **imagen**: Ubicación donde se encuentra el logo de la marca en el servidor. (String)

- **Ofertas**: Colección donde se almacenan las ofertas disponibles en la tienda online.
  - **idOferta**: Identificador de la oferta, es único y siempre debe estar relleno. (String)
  - **descuento**: Descuento que se aplica en la oferta. Siempre en % (Number)
  - **fechaInicia**: Fecha en la que se empieza a aplicar la oferta. (Date)
  - **fechaFin**: Fecha límite de la oferta. (Date)
  - **descripcion**: Breve descripción del motivo de la oferta. (String)

- **Comentarios**: Colección donde se almacenan las reseñas escritas por los usuarios.
  - **calificacionGeneral**: Puntación general que se le otorga al procucto. (Number)
  - **titulo**: Titulo de la reseña (String)
  - **comentario**: Valoración del usuario al producto. (String)
  - **recomendacion**: Indica si el usuario aconsejaría comprar el producto. (Boolean)
  - **calidadPrecio**: Puntación que se le otorga al producto en cuanto a calidad precio. (Number)
  - **alias**: Nombre del usuario que escribe la reseña. (String)
  - **email**: Correo del usuario que escribe la reseña. (String)

- **Método de Pago**: Colección donde se almacenan los métodos de pago con los que se pueden pagar los pedidos.

  - **nombre**: Nombre del método de pago (String)
  - **imagen**: Ubicación donde se encuentra el logo del método de pago. (String)

- **Transportes**: Colección donde se almacenan las empresas de transportes utilizadas para el envío de los pedidos.
  - **nombre**: Nombre de la empresa de transporte. (String)

- **Pedidos**: Colección donde se almacena la información relativa a los pedidos.
  - **idPedido**: Identificador del pedido, debe ser único. (String)
  - **fechaEntrada**: Fecha en la que se realiza el pedido. (Date)
  - **fechaEntrega**: Fecha aproximada de la entrega del pedido. (Date)
  - **estado**: Indica la situación del pedido: pendiente, preparando, entregado y finalizado. (String)
  - **total**: Precio total del pedido. (Number)
  - **formasPago**: Método de pago del pedido. (String)
  - **cobrado**: Indica si el pedido está pagado o no. (Boolean)
  - **usuario**: Identificador del usuario que realizo el pedido. (ObjectId)
  - **articulos**: Artículos que contiene el pedido. Array de objetos formados por el identificador del producto y cantidad del producto comprado. (Array)

- **Facturas**: Colección donde se almacena la información relativa a las facturas de los pedidos realizados.
  - **nFactura**: Identificador de la factura, debe ser único. (String)
  - **fecha**: Fecha en la que se crea la factura. (Date)
  - **cliente**: Identificador del usuario que realiza la compra. (ObjectId)
  - **total**: Precio total de la factura. (Number)
  - **formasPago**: Método de pago con el que se pago el pedido. (String)
  - **idPedido**: Identificador del pedido asociado a la factura. (String)

- **articulos**: Artículos que contiene el pedido. Array de objetos formados por el identificador del producto y cantidad del producto comprado. (Boolean)

- **Mensajes**: Colección donde se almacenan los mensajes enviados por parte de los usuarios al administrador.
  - **idUsuario**: Identificador del usuario que escribe el mensaje. (ObjectId)
  - **nombreUsuario**: Nombre del usuario que escribe el mensaje. (String)
  - **correoUsuario**: Correo del usuario que escribe el mensaje. (String)
  - **mensaje**: mensaje que se envía al administrador. (String)
  - **leido**: Indica si un mensaje se ha leído o no. (Boolean)
  - **fecha**: Fecha en la que se envió el mensaje. (Date)

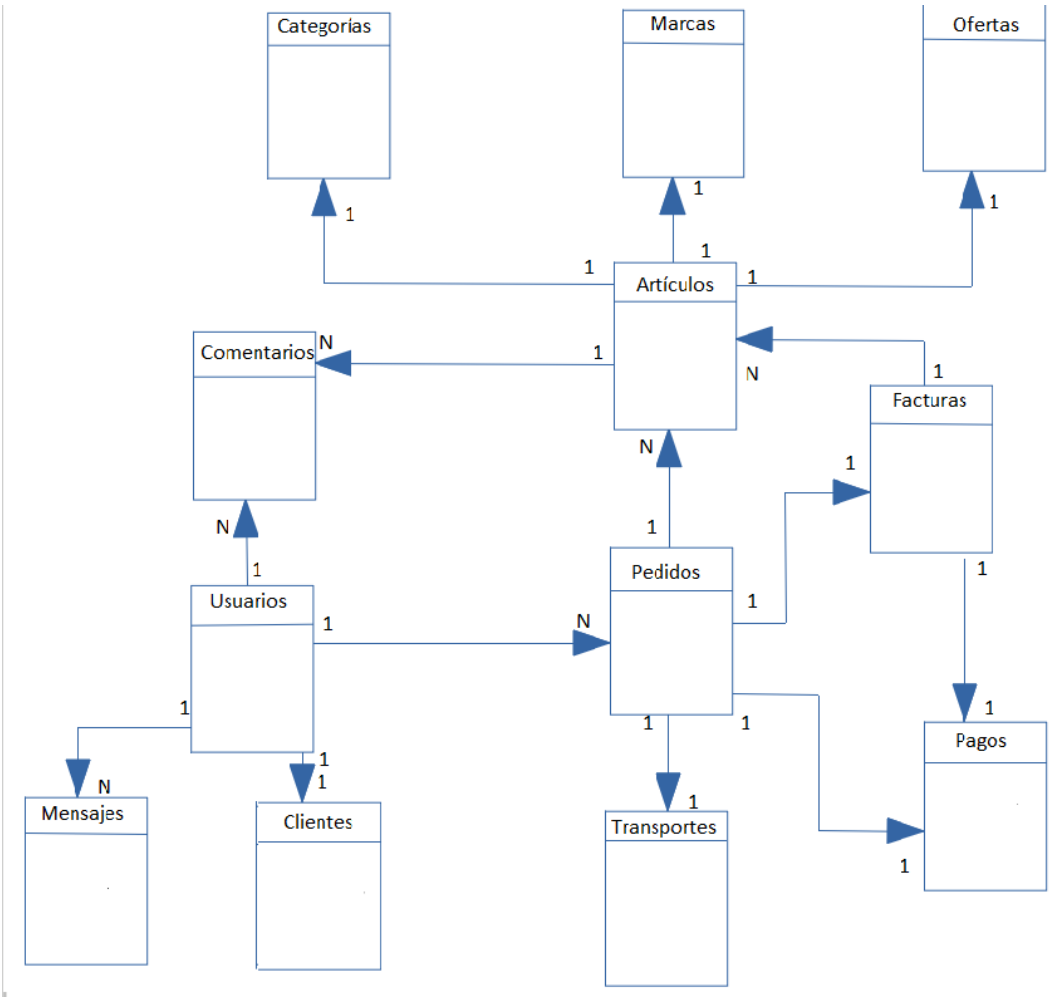


Figura 4.3: Diseño base de datos del servidor utilizando una base de datos relacional

4.3. Diseño e implementación del servidor

El servidor es el encargado de recibir las peticiones del cliente y, de su posterior respuesta. Para ello, se ha implementado un API REST. Esta API REST es la encargada de devolver los datos requeridos en cada petición del cliente. El API REST se explica en la sección 4.3.2. Por otro lado, el servidor es el que realiza las peticiones a base de datos para la obtención, el guardado, borrado y actualización de los datos almacenados en ella.

La implementación del servidor se ha realizado usando NodeJS y Express, usado este último, para la estructuración de los archivos del servidor y para el enrutamiento de las llamadas a la API REST. NodeJs está orientado a eventos asíncronos que se ejecutan en un único hilo, como se explicó en la sección 3.7. Para el manejo de los eventos asícronos es necesario recurrir a la programación asíncrona y, en este tipo de programación, es común recurrir al uso de los denominados callbacks, funciones que ejecutan otra función una vez terminada la operación. Para evitar estos callbacks y hacer el código más legible, se ha recurrido al uso de los operadores $async$ y $await$. Estos operadores tienen como finalidad simplificar la manera de programación cuando se programa de manera asícrona, ya que permiten ejecutar promesas y esperar el resultado sin utilizar $then$ y $catch$, muy usados en Javascript para este tipo de programación y, que lo único que les diferencia de los callbacks es la simplifidad en el código.

4.3.1. Estructura del servidor

Figura 4.4: Estructura de los Directorios del servidor.

En la figura 4.4 se puede observar como está organizado el servidor.

- **Server.js**, archivo donde se establece el puerto de la conexión de la aplicación y, desde el cual se inicia el servidor. Además, carga el archivo **App.js**, archivo principal del servidor.
- **App.js**, archivo principal del servidor. Es donde se establece la conexión con la base de datos y, donde se configura el middleware para servir el API REST y los datos estáticos.
- **Package.json**, archivo donde se especifican las dependencias de las librerías utilizadas en el servidor y la versión de cada una de ellas. Además, se define el nombre y la descripción de la aplicación.
- **/.node_modules**, directorio donde están instaladas las librerías usadas en el servidor. Al recurrir a Express para la organización del servidor, muchas de las librerías ya vienen

definidas. Pero, muchas otras se han instalado por necesidades de la aplicación. Para su instalación se ha usado el comando *npm install*.

- **/config**, directorio que contiene la configuración del servidor. El archivo **nevadoServer.properties** contiene los contextos de los distintos endpoints usados en el API REST. De este modo, en el caso de cambiar de contexto, únicamente se modificaría en este archivo y no en cada uno de los endpoints afectados por la modificación. El archivo **passport.js**, permite la autenticación de los usuarios. Además, permite que únicamente el usuario con rol de administrador pueda acceder a la parte administración de la tienda online.

- **/modelos**, directorio que contiene la estructura de los documentos de las colecciones usadas en la base de datos de la aplicación.

- **/routes**, directorio en el que se encuentran todos los endpoints de la API. Consta de dos subdirectorios, uno para la parte que solo puede ver el administrador de la tienda online (**/admin**) y, otro para la parte de la tienda online que ven los usuarios **/front**. Además, contiene los archivos **APIREST.JS**, **APIRESTADMIN.JS**, **APIRESTORDERS.JS** y **APIRESTUSER.JS**. Estos archivos son los que determinan a que archivo se ha de dirigir cada contexto del API.

- **/services**, contiene la lógica del negocio de la aplicación, es decir, se especifica que se debe hacer en cada llamada a la API. En estos archivos se produce el acceso a la base de datos para el guardado, borrado, búsqueda y actualización de los datos almacenados. Está organizado de la misma forma que el directorio **/routes**, un subdirectorio para la administración y otro, para la tienda online que ven los usuarios.

- **/utils**, directorio que contiene las funciones genéricas usuadas en los diferentes servicios del servidor. Entre algunas funcionalidades, destacan el formateo de fechas, la creacción de PDFs y la configuración del correo para poder enviar emails a los usuarios de la tienda online.

4.3.2. API REST de la aplicación

La API es cada uno de los endpoints que tiene la aplicación para que el cliente y el servidor se comuniquen. Se utiliza el protocolo HTTP para establecer estas conexiones, por lo que las operaciones soportadas son GET, POST, PUT y DELETE. La respuesta por parte del servidor a estas llamadas tienen un formato JSON.

Los endpoints de la aplicación implementada en este trabajo son:

- **POST /nevado/auth/register**: registra nuevos usuarios en el sistema. Antes de registrarlos, comprueba si existe el email en el sistema, en caso de existir manda un error indicando que el usuario está dado de alta. Al mismo tiempo que se crea un registro nuevo en la colección de usuarios, se crea un nuevo registro en la colección de clientes.
- **POST /nevado/auth/register/idUsuario**: Actualiza la información de confirmación del usuario.
- **POST /nevado/auth/login**: Permite el inicio de sesión. Para hacer este inicio de sesión, se comprueba el email y la contraseña. Si son erróneas devuelve un error que se muestra en pantalla.
- **GET /nevado/auth/logout**: Cierra la sesión del usuario.
- **GET /nevado/auth/forgetPassword/email**: Envía un correo al usuario, que ha olvidado la contraseña, con el enlace para restituirla. Antes de envíar el correo se comprueba que el usuario esté registrado en la aplicación. Si no lo está devuelve error y no envía el correo.
- **POST /nevado/auth/forgetPassword/idUsuario**: Permite reestablecer la contraseña del usuario.
- **GET /nevado/user/idUsuario**: Obtiene la información personal de un usuario para mostrarla en pantalla. El usuario tiene que estar autenticado para ver esta información.
- **POST /nevado/user/idUsuario**: Permite actualizar los datos personales de un usuario. Sólo el usuario y el administrador de la aplicación pueden modificar estos datos.
- **GET /nevado/orders/idUsuario**: Devuelve la lista de pedidos que el usuario ha realizado. El usuario debe haber iniciado sesión en la tienda online para ver esta información.

API parte de administración de la aplicación

La administración es lo que comunmente se denomina trastienda. Permite gestionar la tienda online, es decir, añadir productos, proveedores, ver la lista de clientes y administrar los pedidos que hay. Únicamente puede acceder a esta parte aquellos usuarios que tienen rol de administrador. Para controlar esto, se ha usado $Passport.js$. Passport permite añadir en cada endpoint la autenticación del usuario. Esta autenticación será explicada en la sección 4.3.3.

- **GET /nevado/admin**: Muestra la página principal de la parte de administración.
- **GET /nevado/admin/orders**: Devuelve la lista de pedidos realizados por los clientes de la tienda online. Estos pedidos están ordenados por la fecha de entrada. Se puede aplicar filtros, para que únicamente devuelva los pedidos que se necesitan.
- **GET /nevado/admin/orders/export**: Exporta la lista de pedidos realizados por parte de los clientes en formato excel.
- **POST /nevado/admin/orders/idPedido**: Actualiza el estado del pedido. Cuando un cliente finaliza un pedido siempre se crea con estado "pendiente". El administrador modifica el estado cuando se empieza a preparar para permitir al usuario conecer el estado de su pedido.
- **POST /nevado/admin/orders/cobrado/idPedido**: Actualiza el campo cobrado de un pedido. Está opción se implemento pensando en los pagos contra-reembolso, ya que estos pagos se realizan en el momento que el cliente ha recibido su producto en casa.
- **GET /nevado/admin/orders/pending**: Devuelve el número de pedidos que están con estado "pendiente". Este número se muestra en el header de la parte de administración, agilizando la preparación de los envíos.
- **GET /nevado/admin/orders/products**: Recupera la información de los productos que contiene un pedido.
- **GET /nevado/admin/invoices**: Devuelve la lista de facturas del sistema. Está ordenada por número de factura y, además, permite aplicar filtros para la búsqueda de facturas concretas.

- **GET /nevado/admin/invoices/export**: Exporta la lista de facturas almacenadas en el sistema en formato excel.
- **POST /nevado/admin/invoices/pedido**: Genera la factura correspondiente a un pedido. La factura siempre se genera al finalizar el pedido, exceptuando cuando el método de pago es contra-reembolso, que se generará a mano.
- **POST /nevado/admin/invoices/products**: Devuelve la información relativa a los productos que contiene la factura de un pedido.
- **POST /nevado/admin/invoices/client**: Devuelve la información que se necesita de un cliente para generar la factura.
- **POST /nevado/admin/invoices/pdf**: Genera el PDF de la factura.
- **GET /nevado/admin/invoices/idFactura**: Comprueba si la factura está generada. Sólo se puede generar una factura por pedido.
- **GET /nevado/admin/products**: Retorna la lista de productos que tiene la tienda online. Permite filtrar los productos por código, marca, categoria, modelo y precio.
- **GET /nevado/admin/products/export**: Exporta la lista de productos de la tienda online en formato excel.
- **GET /nevado/admin/products/idProducto**: Retorna la información de un producto.
- **POST /nevado/admin/products**: Añade un nuevo producto en el sistema.
- **POST /nevado/admin/products/idProducto**: Actualiza la información de un producto.
- **DELETE /nevado/admin/products/idProducto**: Elimina un producto del catálogo de la tienda online.
- **GET /nevado/admin/brands**: Devuelve la lista de marcas registradas en la tienda online.
- **GET /nevado/admin/brands/export**: Exporta la lista de marcas registradas en la tienda online en formato excel.
- **DELETE /nevado/admin/brands/idMarca**: Elimina una marca de la tienda online.

- **POST /nevado/admin/brands**: Añade una nueva marca al registro de la tienda online.
- **GET /nevado/admin/categories**: Devuelve la lista de categorías registradas en la tienda online.
- **GET /nevado/admin/categories/export**: Exporta la lista de categorías registradas en la tienda online en formato excel.
- **DELETE /nevado/admin/categories/idCategoría**: Elimina una categoría de la tienda online.
- **GET /nevado/admin/categories/products**: Devuelve los productos que tienen una misma categoría.
- **POST /nevado/admin/categories**: Añade una nueva categoría al registro de la tienda online.
- **GET /nevado/admin/clients**: Devuelve el listado de usuarios registrados en la tienda online. Además, permite filtrar el listado por nombre, apellidos, email, dni y teléfono.
- **GET /nevado/admin/clients/export**: Exporta el listado de usuarios registrados en la tienda online en formato excel.
- **DELETE /nevado/admin/clients/idUsuario**: Elimina un usuario de la aplicación. El usuario registrado puede cancelar su cuenta cuando lo desee a través de un correo electrónico al administrador. El adminitrador, por la Ley de protección de datos y política de privacidad, deberá borrarlo del registro del cliente.
- **GET /nevado/admin/payments**: Devuelve la lista de métodos de pagos disponibles en la tienda online. Se puede filtrar por el nombre del método de pago.
- **POST /nevado/admin/payments**: Añade un nuevo método de pago a la tienda online.
- **GET /nevado/admin/payments/export**: Exporta la lista de métodos de pagos disponibles en la tienda online en formato excel.
- **DELETE /nevado/admin/payments/idPago**: Elimina un método de pago del sistema.

- **GET /nevado/admin/transports**: Devuelve la lista de empresas de transporte que se utilizan para realizar los envíos. Se puede filtrar la lista por el nombre de la empresa.
- **GET /nevado/admin/transports/export**: Exporta la lista de empresas de transporte que se utilizan para realizar los envíos en formato excel.
- **POST /nevado/admin/transports**: Añade una nueva empresa de transporte.
- **DELETE /nevado/admin/transports/idEmpresaTransporte**: Elimina una empresa de transporte del sistema.
- **GET /nevado/admin/offers**: Devuelve la lista de ofertas que están disponibles en la tienda online.
- **POST /nevado/admin/offers**: Añade una nueva oferta.
- **GET /nevado/admin/offers/export**: Exporta la lista de ofertas disponibles en formato excel.
- **POST /nevado/admin/offers/idOferta**: Permite modificar las fechas en la que la oferta está disponible así como la descripción de la misma.
- **GET /nevado/admin/offers/idOferta**: Elimina una oferta del sistema.
- **GET /nevado/admin/offers/Products**: Devuelve los productos que tienen ofertas.
- **GET /nevado/admin/messages**: Retorna la lista de mensajes envíados por los usuarios.
- **GET /nevado/admin/messages/export**: Exporta la lista de mensajes envíados por los usuarios en formato excel.
- **DELETE /nevado/admin/messages/idMensaje**: Elimina un mensaje.
- **POST /nevado/admin/messages/idMensaje**: Permite poner el mensaje como leído o no leído.
- **POST /nevado/admin/messages/send**: Envía la respuesta a un mensaje a través de un correo electrónico.

- **GET /nevado/admin/import/template**: Descarga la plantilla excel que hay que usar para poder importar datos de manera masiva en la tienda online.
- **POST /nevado/admin/import/upload**: Importa los datos del archivo excel seleccionado.
- **GET /nevado/admin/import/getImportData**: Muestra el log de la importación.

API parte pública de la aplicación

La parte pública de la aplicación la forman las vistas que no necesitan un rol especifíco para interactuar en ellas. Los endpoints de esta parte son los siguientes:

- **GET /home**: Muestra la página principal de la aplicación.
- **GET /home/topVentas**: Devuelve los productos más vendidos para mostrarlos en el apartado Top Ventas de la página principal.
- **GET /home/ofertas**: Devuelve los productos que tienen ofertas para mostrarlos en el apartado ofertas de la vista principal.
- **GET /home/largeAppliances**: Muestra las subcategorías que tiene la categoría Gran electrodoméstico.
- **GET /home/smallKitchenAppliances**: Muestra las subcategorías que tiene la categoría Pequeño electrodoméstico de cocina.
- **GET /home/image**: Muestra las subcategorías que tiene la categoría Imagen.
- **GET /home/sound**: Muestra las subcategorías que tiene la categoría Sonido.
- **GET /home/telephony**: Muestra las subcategorías que tiene la categoría Telefonía y electrónica.
- **GET /home/personalCare**: Muestra las subcategorías que tiene la categoría Cuidado personal.
- **GET /home/computing**: Muestra las subcategorías que tiene la categoría Informática.
- **GET /home/product/idProducto**: Devuelve la información relativa a un producto para mostrar esta información en la vista de detalle del producto.

- **GET /home/brand**: Devuelve el logo de la marca del producto pasado como parámetro para mostrarlo en la vista detalle del producto.
- **GET /home/similarProducts**: Permite ver productos similares en la página de detalle de productos. Ofreciendo diferentes alternativas al producto seleccionado.
- **GET /home/category/nombreCategoria**: Devuelve todos los productos de una subcategoría. Estos productos se mostrarán en la vista de la subcategoría seleccionada.
- **GET /home/brands**: Devuelve los productos de una subcategoría filtrados por marca.
- **GET /home/paymentMethods**: Retorna los métodos de pago con los que el cliente puede pagar su pedido.
- **GET /home/cart**: Permite ver los productos que hay en el carrito de compra.
- **GET /home/termUse**: Muestra la página de condiciones de uso.
- **GET /home/faq**: Muestra la página de preguntas frecuentes.
- **GET /home/contact**: Muestra la página de ¿quiénes somos?.
- **GET /home/privacyPolicy**: Muestra la página de política de privacidad.
- **POST /home/review**: Añade una reseña en un producto.
- **POST /home/saveOrder**: Permite finalizar el pedido.
- **POST /home/message**: Envía un mensaje al administrador de la aplicación.

4.3.3. Registro y autenticación de usuarios

La aplicación consta de dos perfiles de usuario, un perfil cliente y un perfil administrador. El perfil cliente se asigna a un usuario cuando se registra en la apliacación. El perfil administrador únicamente lo tiene el dueño o administrador de la tienda online. Además, el perfil administrador permite acceder a la trastienda.

Proceso de registro de un usuario

Cuando un usuario decide registrarse en la tienda online debe acceder a $/nevado/login$, seleccionar la pestaña Registrarse y rellenar los campos obligatorios para el resgistro. Cuando

el usuario pulsa sobre el botón registrarse se realiza una petición al servidor, en la que viajan los datos introducidos por el usuario. El servidor comprueba si el email introducido ya se encuentra en el sistema. Para ello, realiza una búsqueda en la base de datos. Si el email ya está registrado se devuelve un error, indicando que el usuario ya está registrado. Si, por el contrario, el email no se encuentra en nuestro sistema entonces guarda los datos del usuario en la base de datos, creando un nuevo registro en la colección de usuarios. Antes de crear el registro, se deben proteger los datos más susceptibles del usuario. La contraseña es la que permite realizar compras y acceder a la información personal del usuario, por lo que no se puede guardar en la base de datos en texto plano. Según la Ley de protección de datos, en su artículo noveno, el encargado del tratamiento de los datos susceptibles debe garantizar la seguridad de los mismos adoptando las medidas técnicas y organizativas necesarias. Para cumplir con este artículo, se encripta la contraseña utilizando la librería Bcryptjs. Esta librería permite realizar una encriptación robusta frente ataques de fuerza bruta, como se ha mencionado en la sección 3.13. Para llevar a cabo la encriptación de la contraseña se realizan los siguientes pasos:

1. Se genera el salt, valor aleatorio que es usado para dar mayor robustez a la encriptación. Para generar este valor se utiliza la función *genSalt*, función propia de la librería de Bcrypt. El salt generado tiene una longitud de 10 caracteres, valor por defecto de Bcrypt. Se ha optado por este valor porque es lo suficientemente grande para evitar que ataques de fuerza bruta consigan descifrar la contraseña. Además, con este valor el proceso de encriptación no se demora mucho tiempo.

2. Se encripta la contraseña utilizando la función *hash*, también de Bcrypt. Para llevar a cabo la encriptación, está función utiliza el salt generado en el paso anterior.

Una vez encriptada la contraseña, se crea el nuevo registro en la colección usuarios. Además, se crea una nueva entrada en la colección clientes, que almacena la información personal del cliente. Para dar por finalizado el proceso de registro, se envía un correo al usuario. El correo contiene un enlace de confirmación de la cuenta. De este modo, se garantiza que el correo introducido por el usuario es válido y está dado de alta en un sistema de mensajería. Si el usuario no confirma el correo no podrá realizar compras en la tienda online.

Proceso de autenticación de un usuario

Un usuario debe autenticarse para poder realizar determinadas operaciones dentro de la tienda online, por ejemplo, realizar sus compras, acceder a sus datos personales o acceder a la información de sus pedidos. Para realizar esta autenticación se ha utilizado la librería $Passportjs$. Esta librería permite realizar este proceso de manera sencilla y rápida. Passport almacena los datos necesarios para la autenticación en una cookie de sesión. Para configurar dicha cookie de sesión se ha usado $express-session$, que es un módulo de Express. La configuración se realiza cuando se arranca el servidor de node, por lo que se encuentra situada en el fichero app.js. Los parámetros que se han configurado son el tiempo de expiración de la cookie, dónde se guardan las sesiones y el valor de la clave secret. Este secret es una cadena de texto que se utiliza como identificador y que permite validar la sesión. Passport tiene muchas estrategias para autenticar a un usuario, en la aplicación se ha optado por utilizar $Passportlocal$, la cual utiliza el email y la contraseña para realizar la autenticación.

Para acceder a una cuenta de usuario, se deber navegar a /nevado/login, acceder a la pestaña entrar, rellenar el email y la contraseña y pulsar sobre el botón entrar. Al pulsar sobre el botón, se realiza una petición al servidor, en la que viajan la contraseña y el email introducido por el usuario. El servidor ejecuta la estrategia de Passport que se ha implementado. Esta estrategia consiste en buscar el email en el sistema. Si no lo encuentra envía un error, indicando que el usuario no está registrado. Si lo encuentra, procede a comparar la contraseña enviada con la almacenada. Para realizar esta comparación se ha utilizado la función de bcrypt *compare*, que encripta la contraseña introducida y la compara con la almacenada. Si las contraseñas no coinciden se envía un error. Si coinciden, Passport crea una sesión de usuario que serializa y guarda en la base de datos. Además, estos datos de la sesión se añaden a la cookie, la cual se envía en cada una de las peticiones realizadas al servidor. De esta forma, no hay que enviar el email y la contraseña en cada llamada. El servidor únicamente se encargará de validar la cookie enviada.

Para acceder a determinadas vistas de la aplicación como, por ejemplo, las vistas de administración, el usuario además de estar logueado deberá tener el perfil de administrador. Para hacer esta comprobación, en cada llamada realizada bajo el contexto $/admin$ se ejecutará una función que valida que el usuario tenga este perfil. Si el perfil no coincide, no se permitirá el acceso a esta parte y se redireccionará a la página principal.

4.3.4. Sistema de subida de archivos en el servidor

La aplicación permite realizar subidas de archivos al servidor. Los archivos que se permiten subir son imágenes y archivos .zip. La subida de imágenes se produce al registrar un nuevo producto o a actualizarse. Además, se suben imágenes al añadir a la tienda online nuevos proveedores, nuevas categorías y nuevas formas de pago. Las imágenes tienen que tener un formato determinado para poder subirse($.jpeg$, $.jpg$, $.png$). Si el formato es diferente se produce un error, que se muestra por pantalla. Los archivos .zip se suben al servidor cuando se realiza la importación de productos, categorías o marcas. El sistema de importación se explica en la sección 4.3.6.

Para realizar esta subida de archivos en el servidor, se ha utilizado la librería $Multer$ de NodeJs. $Multer$ facilita el manejo de datos codificados como multipart/form-data. Para ello, se incluye la librería y se configura pasándole como parámetro el path donde se almacenarán los archivos subidos. En el route (endpoint correspondiente) se ejecuta la función *single*, pasándole como parámetro el nombre del campo del formulario donde se envió el fichero.

var multer = require('multer');

var upload = multer(dest: properties.get('nevado.server.import.folder'));

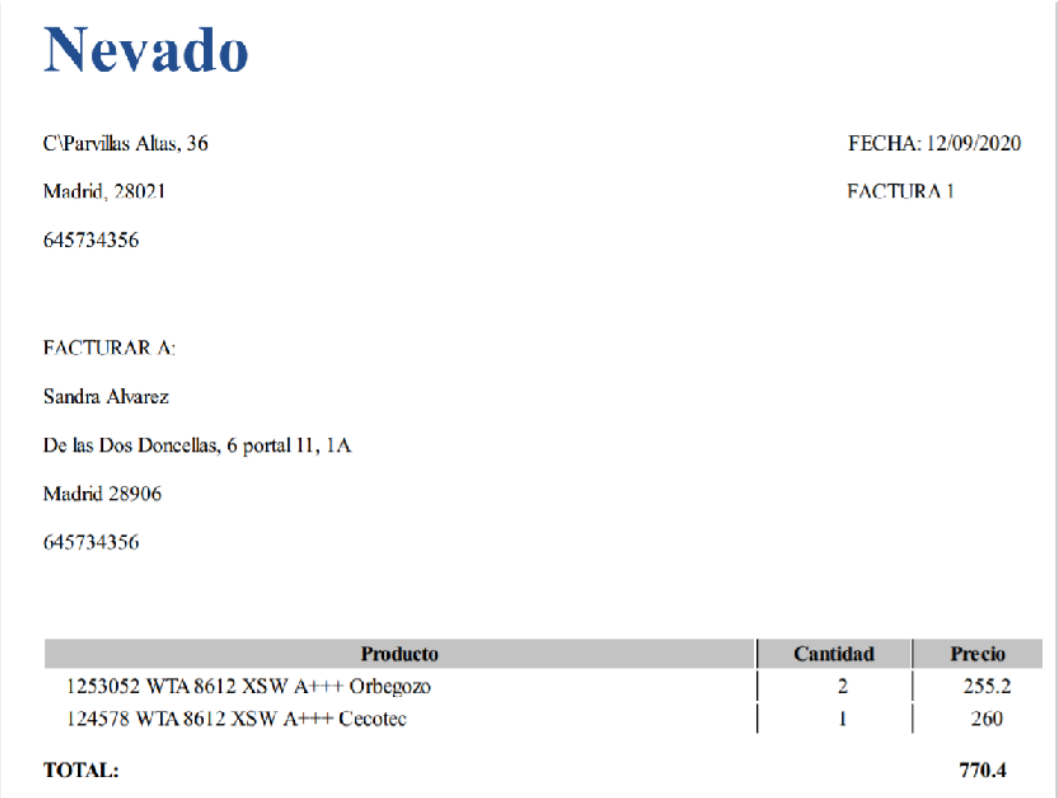
routesImport.post('/upload', passportConfig.isAuthenticated, upload.single('file'), ...);

4.3.5. Generación de PDFs

Cuando se realiza un pedido es necesario enviar la factura, ya que es esta la que actúa como ticket de compra. Por ello, se ha implementado una funcionalidad que permite generar el PDF de una factura. Para la generación de los PDFs se ha utilizado la librería $HTML - PDF$ de NodeJS. A través de un HTML, generado con la información recogida en la base de datos, se crea el PDF. Para ello, se utiliza la función *create* de la librería, a la cual se le pasa el HTML generado y se llama a la función *toFile*, a la que se le indica donde se crea el archivo .pdf y que nombre va a tener dicho archivo.

const pdf = require('html-pdf');

pdf.create(content).toFile(properties.get('nevado.server.invoices')+/- fileName+".pdf", ...);



Nevado

C\Parvillas Altas, 36

Madrid, 28021

645734356

FECHA: 12/09/2020

FACTURA 1

FACTURAR A:

Sandra Alvarez

De las Dos Doncellas, 6 portal 11, 1A

Madrid 28906

645734356

Producto	Cantidad	Precio
1253052 WTA 8612 XSW A+++ Orbegozo	2	255.2
124578 WTA 8612 XSW A+++ Cecotec	1	260
TOTAL:		**770.4**

Figura 4.5: Ejemplo PDF factura.

4.3.6. Sistema de importación y exportación

Una tienda online requiere tener en su catálogo numerosos productos. Añadir cada uno de ellos al sistema puede llegar a ser pesado y tedioso. Por eso, se ha implementado un sistema de importación, que permite cargar masivamente datos en la base de datos. En este sistema se pueden cargar productos, marcas y categorías.

Para realizar la importación el administrador de la tienda debe ir a $/admin/import$, seleccionar el tipo de datos que se van a importar, es decir, debe indicar si son productos, categorías o marcas y pulsar sobre el botón importar. Cuando se pulsa sobre el botón importar se realiza una petición al servidor, en la que viaja el archivo a importar. El tipo de archivo que se permite importar es .zip. El zip contiene un excel con los registros que se quieren añadir a la base de datos y las imágenes asociadas a cada registro. Cuando se recibe la peticion, se guarda el archivo .zip en el servidor, como se explica en la sección 4.3.4 y, posteriormente, se realiza

la importación. Para ello, se descomprime el archivo .zip, utilizando la librería $Adm-zip$. Se extrae el documento excel y se parsea dicho documento para convertirlo en un array de objetos Json. Para realizar el parseo del documento excel se usa la librería $SheetJS-XLSX$. El array va a tener tantos objetos Json como registros tenga el documento excel. A cada objeto Json se le añade la ubicación de la imagen asociada a cada registro. Antes de guardarlo en la base de datos, se realizan una serie de comprobaciones como, por ejemplo, que todos los campos obligatorios estén rellenos o, en el caso, de los productos que el precio sea numérico o la marca y categoría exista en la base de datos. Una vez terminada esta comprobación, se insertan en la base de datos todos los registros, utilizando el comando *insertMany* de Mongoose, realizando así una única query. Cuando el proceso termina, se muestra en pantalla, a través de un sistema de log, el número de registros importados y cuales de ellos han fallado, indicando el número de línea y el error producido.

Para facilitar la creación de los documentos excel y evitar errores a la hora de importarlos, se ha implemantado una opción de descarga de plantilla. Para descargar esta plantilla, basta con seleccionar el tipo de dato que se quiere importar y pulsar sobre el botón plantilla, descargando un documento excel con las cabeceras de los campos que se han de rellenar. Este sistema de descarga de plantilla es similar al sistema de descarga de los documentos excel que se encuentran en las distintas vistas de la parte de administración.

Para la creacción de los documentos excel, se utiliza la librería de NodeJS $excel4node$. El documento excel se va rellenando con los metadatos de la cabecera y los datos devueltos por la base de datos. En el caso de la plantilla, sólo se rellena la cabecera del documento excel. Este documento se envía en la respuesta al cliente y, el cliente lo descarga utilizando *saveAs* de la librería de Angular $File-saver$.

4.4. Diseño e implementación del cliente

El cliente se ha desarrollado utilizando Angular y está programado en Typescript. Está formado por diferentes vistas, que forman la interfaz de usuario. Estas vistas se cargan de manera dinámica e instántanea, ya que cuando se modifica la url no se recarga la página. Por lo tanto, el cliente se ha implementado como Single-Page. Además, con el uso de Angular, el cliente tiene una arquitectura modelo-vista-controlador.

El cliente es el encargado de recoger la información introducida por el usuario cuando interactua en la aplicación. Para atender los eventos lanzados por el usuario, se ha alojado en el servidor proporcionado por Angular-Cli.

4.4.1. Estructura del cliente

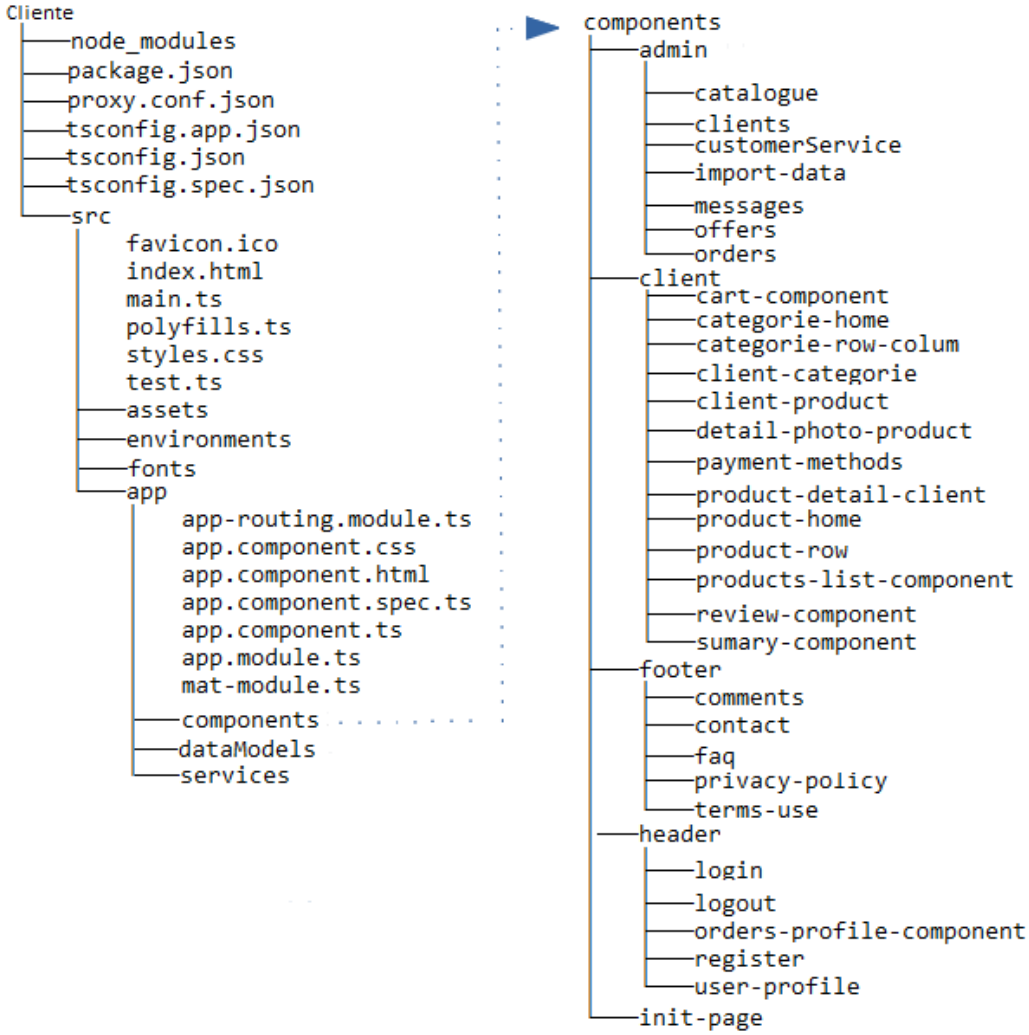


Figura 4.6: Estructura de los Directorios del cliente.

En la figura 4.6 se puede observar como está organizado el cliente.

- **/node_modules**, directorio donde se encuentran instaladas las dependencias del cliente. Se denomina así, porque utiliza el comando *npm install* para descargar e instalar las dependencias, ya que Angular-Cli usa NodeJS para la ejecución de su servidor.

- **package.json**, archivo donde se especifican las dependencias utilizadas en el proyecto. Estas dependencias se declaran junto con la versión utilizada.

- **proxy.conf.json**, archivo de configuración del proxy utilizado como intermediario entre el cliente y el servidor.

- **tsconfig.json**, archivo de configuración del compilador de los archivos Typescript.

- **/src**, directorio donde se encuentra la lógica del cliente, en el se sitúan los componentes, las directivas y los servicios utilizados para interactuar en la interfaz de usuario. Está formado por:

    - **index.html**, archivo HTML principal de la interfaz de usuario. Se compone del componente raiz que inicia la aplicación y del que parten los demás componentes. En este HTML se cargan los estilos generales utilizados y los scripts necesarios para el correcto funcionamiento del cliente.
    - **main.ts**, archivo Typescript principal. En el se estable el módulo principal del cliente, siendo este módulo **AppModule**
    - **styles.css**, archivo donde se definen los estilos generales de la interfaz de usuario.
    - **/fonts**, directorio donde se almacenan las fuentes utilizadas en la interfaz de usuario.
    - **/enviroments**, directorio donde se establece la configuración de los distintos entornos de desarrollo. En el caso de esta aplicación, no se ha configurado ningún entorno, ya que siempre se ha trabajado en local.
    - **/assets**, directorio donde se almacenan los recursos estáticos como el logo o las imágenes del carousel implementado en la página “home”de la aplicación.
    - **/app**, directorio que contiene los componentes, servicios y directivas utilizados para implementar el cliente. Está compuesto por:
        - **app.module.ts**, módulo principal del cliente. En este archivo se especifican todos los componentes que forman la interfaz de usuario.
        - **app.component.html**, archivo HTML principal. Se compone de dos componentes fijos para todas las vistas, el “header” y el “footer” y una parte variable

donde se cargan los distintos componentes. Estos componentes varían según se modifique la URL.

- **app.routing-module.ts**, archivo donde se declaran los componentes que se cargan en la parte variable del componente principal y cuando se han de cargar, es decir, se asocia el componente a la URL.

- **mat.module.ts**, archivo donde se declaran los módulos de angular material que se pueden utilizar.

- **/components**, directorio que contiene los componentes que forman el cliente. Se divide en en cuatro subdirectorios **/admin** (componentes de la trastienda) **/client** (componentes de la tienda vista por los usuarios) **/footer** (componentes del footer) **/header** (components del header) **/init-page** (componentes que forman la vista "home" de la tienda). Cada uno de los componentes tiene asociado un archivo .html para crear el html de la vista, un archivo .css para definir los estilos de la vista y un archivo .ts que es el controlador de la vista.

- **/dataModels**, directorio que contiene los modelos de datos utilizados en el cliente. En este directorio se encuentra el archivo **properties.ts**, que contiene los contextos de las peticiones realizadas al servidor. De este modo, si uno de los contextos se modifica únicamente habría que modificar este fichero y no todos los archivos donde se encuentra el contexto a modificar.

- **/services**, directorio que contiene los servicios utilizados en los controladores de las vistas. Estos servicios se pueden usar desde cualquier componente y sirven para compartir información y datos de un componente a otro. Además, es en estos servicios donde se definen las peticiones a la API. Se divide en tres subdirectores **/admin** (servicios usados en la trastienda), **/front** (servicios usados en la tienda que ve el usuario), **/faq** (servicio usado en el componente preguntas frecuentes). Además contiene los archivos **auth-service.service** (contiene las peticiones a la API del registro), **user-service.service** (servicio donde se definen los valores de las variables guardadas en el localStorage).

4.4.2. Vistas

Las vistas de la aplicación forman la interfaz de usuario. Cada vista está formada por uno o más componentes. Cada uno de estos componentes tiene asociado un controlador, una hoja CSS y un documento HTML. Por este motivo, las aplicaciones desarrolladas con Angular son escalables, ya que hacer una modificación en uno de los componentes no supone tener que modificar el resto.

Para aumentar la escalabildad de la aplicación se han creado componentes reutilizables, es decir, componentes que se usan en varias vistas de la aplicación. Crear estos componentes reutilizables significa aumentar la mantenibilidad de las aplicaciones, ya que a la hora de producirse un error o una actualización en uno de los componentes reutilizables supone modificar únicamente el componente reutilizable, evitando modificar cada uno a uno los componentes afectados por el error o la actualización. Los componentes que se reutilizan en la aplicación son:

- **Client-product**: Tarjeta resumen del producto. Muestra la información más importante del producto, marca, modelo, precio y envío gratuito, si lo tuviese. Permite añadir el producto al carrito de compra pulsando al botón comprar, sin necesidad de pasar por la vista de "Detalle del producto". Se utiliza n veces en el home de la aplicación, ya que se usa para mostrar los productos más vendidos y los productos que tienen oferta. Se utiliza tantas veces como productos tenga la subcategoria en la vista "Filtrado de productos". Además, se utiliza en la vista "Detalle del producto" para mostar los productos de la sección "Artículos relacionados".

Figura 4.7: Ejemplo de una tarjeta resumen del producto (Componente client-product).

- **Product-row**: Fila de tarjetas resumen del producto. Permite mostrar varias tarjetas resumen en una misma fila. Se utiliza en el home de la aplicación, tanto en top ventas como en ofertas, en la vista “Detalle del producto” en el apartado Artículos relacionados y en la vista ”Filtrado de productos”.

Figura 4.8: Ejemplo Fila de tarjetas resumen del producto (Componente product-row).

- **Product-detail-client**: Detalle del producto. Permite visualizar la información del producto de manera detallada. Se utiliza cada vez que se pulsa sobre la imagen o el modelo del producto del compenente client-product.

Figura 4.9: Ejemplo Detalle del producto (Componente Product-detail-client).

- **Client-categorie**: Tarjeta Subcategoría. Permite visualizar el nombre y la imagen con la que se caracteriza una subcategoría. Se utiliza tantas veces como subcategorias tenga la categoría general.

Figura 4.10: Ejemplo Tarjeta subcategoria (Componente Product-detail-client).

Por otro lado, cada vista tiene asociada una ruta. Esta asociación se realiza en el archivo app-routing.module.ts. Cuando se cambia de ruta se cambia de vista, mostrándose la vista correspondiente a dicha ruta.

Las vistas que tiene la aplicación son:

4.4.3. Uso Angular Material

Angular Material es una librería de Angular creada por Google. Permite crear componentes de manera rápida y sencilla. Sus componentes se adaptan a los distintos tamaños de pantalla, por lo se suelen utilizar para crear diseños responsive. En la aplicación se ha utilizado en:

- Las listas de productos, clientes, pedidos, facturas, marcas, categorías, métodos de pago, empresas de transporte y mensajes. Para ello, se ha utilizado *mat-table*.
- Los diálogos de confirmación de borrado de productos, clientes, marcas, categorías, métodos de pago, empresas de transporte y mensajes. Usando para ello *mat-dialog*.
- Los Campos de los fórmularios utilizados para crear productos, marcas, categorías, métodos de pago y empresas de transporte. Y en los campos de los formularios donde se muestra la información de los pedidos, clientes, facturas y mensajes. Para crear estos campos se ha utilizado *mat-form-field*.
- Los checkbox de las vistas de información de producto, del carrito, de las reseñas, entre otros. Para ello, se ha utilizado *mat–checkbox*

- La lista de categorías que tiene una categoría general. Usando *mat-grid-list*.
- La lista de productos de una categoría se ha creado usando también *mat-grid-list*.

4.4.4. Descarga de archivos

Para la descarga de los archivos excel generados en la trastienda y que contienen la lista de productos, clientes, pedidos, facturas, marcas, categorías, empresas de transporte, métodos de pago y mensajes, se ha utilizado la librería $file - saver$. Para realizar la descarga, únicamente hay que llamar a la función *saveAs* de dicha librería y, pasarle como parámetros los bytes del archivo a descargar y el nombre con el que se va a guardar el archivo descargado.

4.4.5. Integración con Paypal

Existen tres métodos de pago para realizar los pagos de los pedidos: tarjeta bancaria, contrareembolso y Paypal. Para realizar el pago a través de Paypal, se ha realizado una integración con la API de Paypal. De esta forma, se usa su pasarela de pagos y se evitan errores de seguridad al no tener que trabajar ni guardar los datos bancarios de los clientes.

Para realizar la integración se han seguido los siguientes pasos:

1. Se crea una cuenta de Paypal para la tienda online en el sitio web de desarrolladores.
2. En la cuenta creada anteriormente se crea un sandbox que permita realizar y recibir los pagos. Al crear el sandbox se obtiene un identificador que se usará para la configuración de Paypal en el componente.
3. En el componente de los métodos de pago se hace la configuración para poder utilizar Paypal. Para ello, se crea un objeto paypalConfig en el que se indica el entorno, el identificador del sandbox, el pago (moneda en que se realiza el pago y cantidad) y una función denominada *onAuthorize*. Dicha función se ejecuta al pulsar sobre el botón pagar de la pasarela de pagos de Paypal, es decir, es la que efectúa el pago.
4. Se añade el script que permite llamar a la API de paypal. Este script se ejecuta al pulsar sobre el botón Paypal, que aparece al seleccionar Paypal como método de pago del pedido.

5. Se añade el botón de Paypal en el HTML del componente “métodos de pago”.

Esta integración permite al usuario efectúar el pago en el sitio web de Paypal, ya que al pulsar sobre el botón “Paypal pagar” se abre la página de inicio de sesión de Paypal. En esta página el usuario deberá introducir las creedenciales correctas para acceder a su cuenta. Una vez dentro, el usuario puede elegir la forma de pago que más le conviene, ya que Paypal permite realizar los pagos a través de su propia cuenta Paypal, a través de tarjeta bancaria o a través de transferencia. Cuando el usuario finaliza el pago, pulsando sobre “Pagar ahora”, se cierra el sitio web de Paypal y se redirecciona al “Home” de la tienda online, mostrando al usuario un mensaje indicando que su pedido ha finalizado correctamente. La figura 4.11 muestra este proceso.

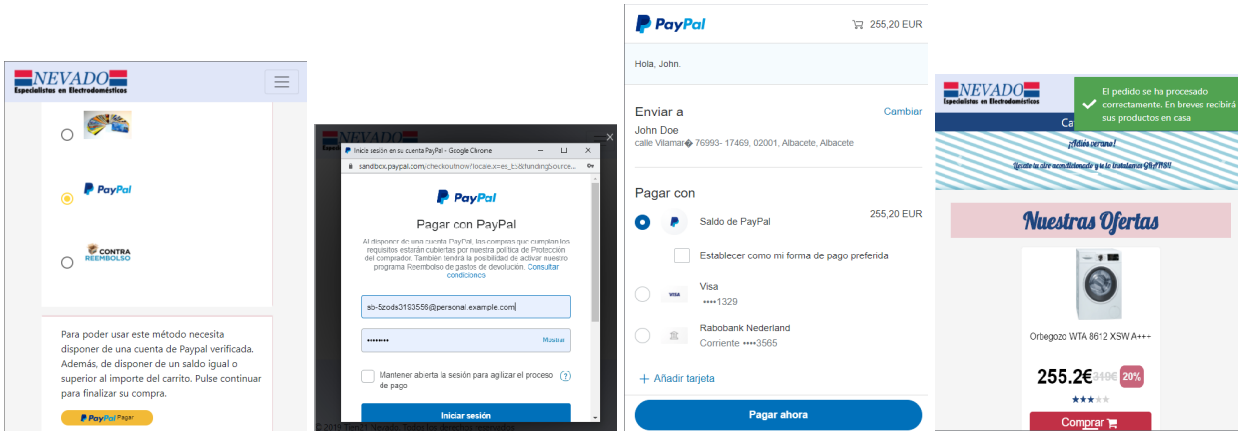
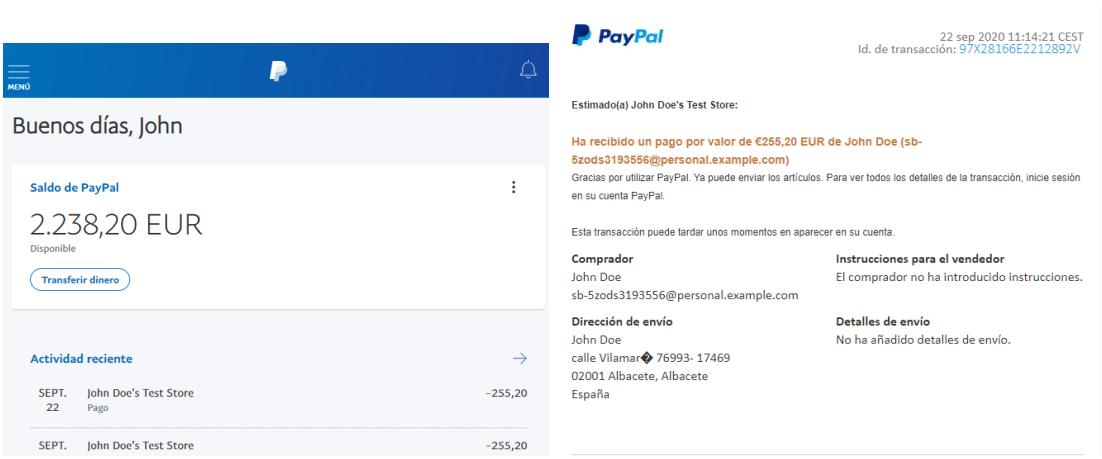


Figura 4.11: Proceso de pago con paypal

En la figura 4.12 se puede observar como el pago se descuenta de la cuenta del cliente y como se recibe en la cuenta de la tienda.



(a) Pago (b) Recibí

Figura 4.12: Ejemplo cargo en la cuenta del cliente y recibí en la cuenta de la tienda

4.4.6. Sistema de avisos

Se ha creado un sistema de avisos mediante el cual el usuario es informado del estado de las operaciones realizadas en la tienda online. Este sistema consiste en mostrar un popup con el mensaje en el margen superior derecho de las vistas. Este popup se muestra una vez finalizada la operación. Si la operación ha finalizado de manera errónea el popup será de color rojo y, si por el contrario, ha finalizado correctamente el popup será de color verde. La figura 4.13 muestra un ejemplo del formato de los avisos.

Figura 4.13: Ejemplos mensajes de aviso

Para implementar este sistema de avisos se ha utilizado el servicio $ToastrService$ de la librería de Angular $ngx-toastr$. Con el uso de este sistema, únicamente se ha de llamar a las funciones *error*, cuando se produce un error y *success*, cuando el proceso es correcto. A ambas funciones hay que pasarles por parámetros el mensaje que se muestra en pantalla y el tiempo que el popup es visible.

4.4.7. Diseño responsive

Actualmente, el acceso a las aplicaciones web se realiza desde diferentes dispositivos: ordenadores, tablets, móviles. Cada uno de estos dispositivos tiene una resolución y un tamaño de pantalla distinto, por lo que la visualización de las páginas no es la misma. Por esta razón, las aplicaciones web tienen que adaptar sus diseños a los diferentes tamaños de pantalla.

Para que los usuarios de la tienda online puedan acceder a través de diferentes dispositivos, se ha creado un diseño adaptativo, denominado diseño Responsive. Este tipo de diseño permite visualizar las vistas de la tienda online en cualquier pantalla. En la figura 4.14 se puede observar como el diseño se adapta al tamaño y resolución de la pantalla.

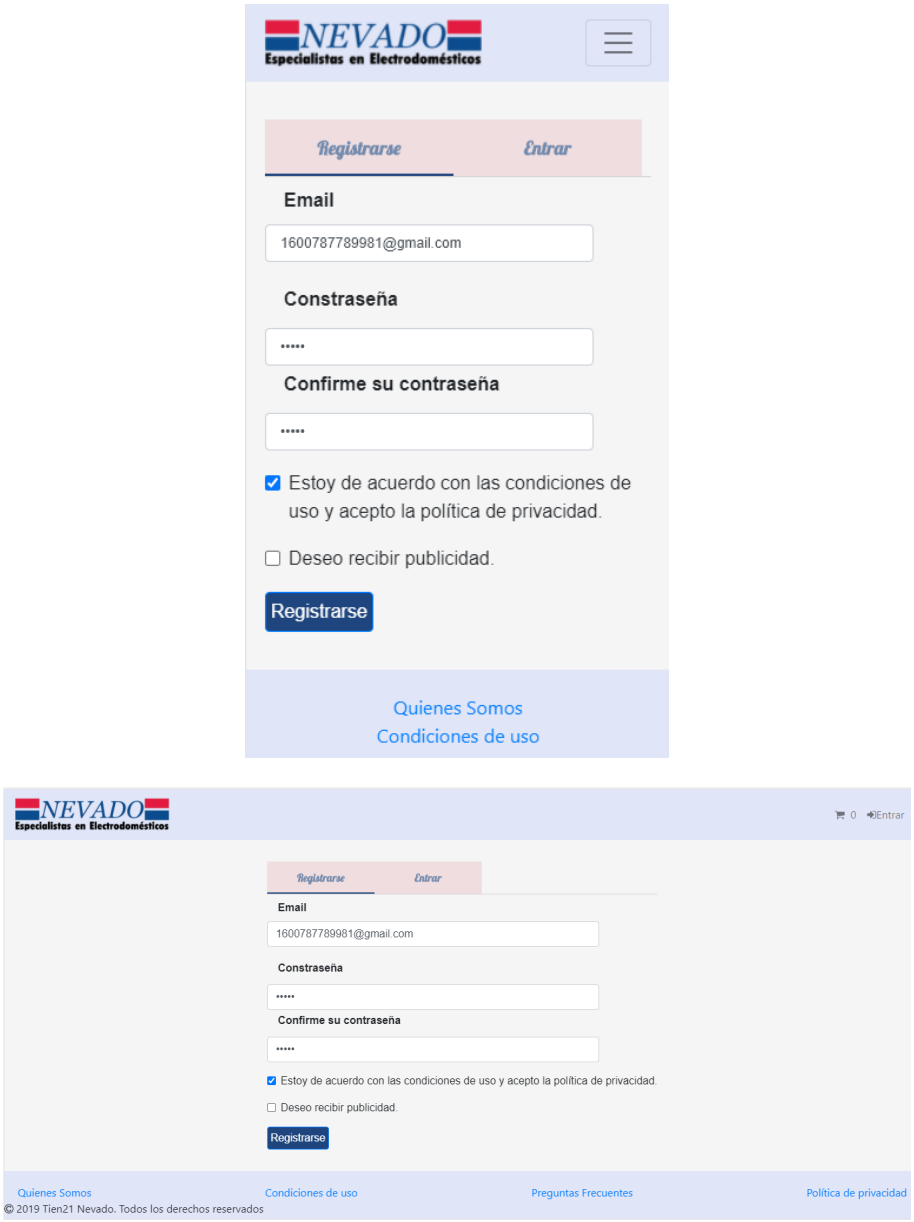


Figura 4.14: Diseño responsive

El diseño responsive se ha logrado, recurriendo al uso de componentes Bootstrap como, por ejemplo, la navbar de la cabecera de las vistas y al uso de las nuevas clases *d-flex*, *flex-column*, *flex-row*, entre otras. Estas clases permiten posicionar los elementos en filas o columnas sin tener en cuenta si el elemento es un elemento de bloque. La variación de rejilla que tiene bootstrap (sm, md, lg y xl) me ha permitido tener diferente posicionamiento para los distintos tamaños de pantalla.

Para los elementos creados sin Bootstrap se ha utilizado media queries de CSS3. Estas media queries permiten redefinir los estilos cuando es necesario, por lo que un mismo elemento puede

tener un estilo diferente para cada tamaño de pantalla.

En algunos componentes se ha recurrido a Angular, utilizando la nomenclatura *class.clase*, para definir estilos, ya que permite añadir estilos en el momento de la visualización de la vista, siempre y cuando se cumplan las condiciones que el desarrollador determina. Además, se ha recurrido al cálculo del tamaño de la pantalla en el momento de la visualización para mostrar un elemento del html u otro, ya que con Angular es sencillo obtener este valor. De esta forma, se puede crear un elemento diferente por cada tamaño de pantalla, adaptando el elemento al tamaño.

Por último, se ha utilizado la librería Angular material para ciertas vistas, por ejemplo para crear la tabla de pedidos que realiza un usuario. Esta librería permite crear componentes responsive de manera sencilla.

Capítulo 5

Resultados

El resultado del trabajo es satisfactorio, ya que se cumplido con el objetivo principal del mismo, crear una tienda online de electrodomésticos. Si se hace una comparativa con otras tiendas online, se puede decir que la aplicación desarrollada, está a la altura de las que vemos en el mercado online de electrodomésticos. En la sección 6.1 se ha detallado los objetivos especificos que se han logrado cumplir.

Capítulo 6

Conclusiones

6.1. Consecución de objetivos

El principal objetivo de este trabajo era desarrollar una aplicación de comercio electrónico, a través de la cual los usuarios pudiesen mirar las características, los precios de los productos, el stock, etc., así como realizar sus compras sin tener que ir a la tienda física. Objetivo que se ha logrado cumplir.

Con la implementación de la aplicación, se ha conseguido cumplir con los objetivos específicos que se marcaban en el capítulo 2, pues:

- La aplicación es responsive. Si se observan las imágenes X, se ve como la aplicación se adapta a diferentes tamaños de pantalla. Para cumplir con este objetivo se ha recurrido al uso de Bootstrap, CSS3 y Angular Material como se ha explicado en el capítulo 4.

- La interfaz de usuario, es una interfaz sencilla, en la que los componentes que se utilizan son muy intuitivos. La mayoría de estos componentes, se encuentran en casi todas las web de comercio electrónico, como por ejemplo, los botones denominados dropdown. Además, se ha usado gran número de iconos, que identifican las funcionalidades que realiza cada botón.

- La aplicación cumple con la normativa de protección de datos, para llevar a cabo este objetivo, se ha creado una política de privacidad, donde se informa al usuario como van a ser tratados sus datos. En cuanto a la seguridad de los datos susceptibles como son las

contraseñas, se ha usado una librería robusta para su encriptación. En la base de datos, se guarda esta información encriptada mediante un hash. El hash esta formado por un salt que dificulta conseguir la información mediante ataques de fuerza bruta. En cuanto a los pagos, se realizan a través de Apis externas a la aplicación, para no tener que almacenar los datos bancarios.

- La aplicación es atractiva y competitiva. Se ha implementado las funcionalidades básicas que deben tener las tiendas online y, además, se ha implementado un sistema de valoración y votación de productos, que no todas las aplicaciones de comercio electrónico tienen. Esto permite a los usuarios conocer las valoraciones de anteriores compradores. Se ha recurrido al uso de colores, iconos, letras y componentes llamativos para captar la atención de los clientes. De esta manera, se amplía el mercado y la aplicación puede situarse entre las más visitadas del mercado de los electrodomésticos.

- Se ha logrado crear una trastienda de administración sencilla. Permite al administrador de la tienda a gestionar pedidos, crear productos y visualizar clientes de manera rápida, evitando que los pedidos se demoren. Para agilizar la carga masiva de los productos, las marcas y las categorías, se ha implementado un servicio de importación y exportación de ficheros excel. Además se ha implementado un sistema de mensajería que permite interactuar entre los usuarios y el administrador de la tienda, pudiendo resolver cualquier duda de los usuarios.

- Con el uso de Angular, Express y Node se ha conseguido crear una aplicación web modulable y escalable, en la que el mantenimiento y la actualización de componentes es sencilla. Cada vista de la aplicación tiene su propio componente, por lo que modificar una vista significaría simplemente modificar el componente asociado a ella.

6.2. Aplicación de lo aprendido

Al realizar este trabajo he puesto en práctica muchos de los conocimientos adquiridos en el grado que estoy cursando, Grado en Tecnologías de la Telecomunicación, sobre todo, aquellos que están relacionados con el desarrollo de aplicaciones web y con la programación.

La asignatura que más me ha aportado y ayudado para hacer este trabajo, es la cursada en el tercer curso del grado, Desarrollo de aplicaciones telemáticas, ya que fue aquí donde aprendí todo lo relacionado con la implementación de aplicaciones web como CSS3, HTML5 y Bootstrap.

En el mismo año, cursé la asignatura Ingeniería de sistemas de la información. Fue aquí donde tuve el primer contacto con Javascript, lenguaje de programación utilizado para desarrollar el servidor de la aplicación web de comercio electrónico Nevado. Además, aprendí a manejar Git, que me ha permitido llevar un control de las versiones que iba teniendo del proyecto, de tal forma que si alguna funcionalidad me fallaba podía seguir con otra.

Asignaturas como Fundamentos de la programación o programación de Sistemas telemáticos han significado mucho, ya que aprendí los conocimientos más básicos de la programación.

Para implementar una aplicación como la desarrollada en este trabajo, hay que tener conocimientos sobre las peticiones HTTP y saber lo que es un API Rest. Estos conocimientos los adquirí en las asignaturas Sistemas telemáticos y Servicios y aplicaciones telemáticas.

Por último, indicar que aunque muchas de las asignaturas que cursé en esta etapa universitaria no han sido necesarias para este trabajo, sí que me han servido para aprender que todo requiere un esfuerzo y un aprendizaje y, que, sin este esfuerzo y aprendizaje no se consiguen las metas que te propones.

6.3. Lecciones aprendidas

El desarrollo de este trabajo ha supuesto un gran reto, pues es el primer proyecto que he realizado de esta complejidad. Durante el transcurso del grado, han sido muchas las prácticas que he realizado, pero en todas tenía las directrices a seguir. En este trabajo, a lo primero que me he enfrentado ha sido a la recapitulación de los requisitos que se necesitaban para implementar la aplicación. Este análisis me ha servido para conocer los puntos más importantes del comercio electrónico.

Al analizar los requerimientos, he tenido que hacer un análasis de las distintas arquitecturas que podría tener la aplicación. Esto me ha servido para tener más conocimiento de los distintos tipos de arquitecturas que tienen las aplicaciones de este negocio. Como he mencionado en el Capítulo 4, me decanté por la arquitectura MEAN porque es la que más se ajustaba a mis nece-

sidades. El haber implementado esta arquitectura me ha permitido conocer nuevas tecnologías: MongoDB, Express, Angular y NodeJs. La mayoría de estas tecnologías se basan en Javascript, pero tienen un concepto diferente a la hora de programar, sobre todo con NodeJS, puesto que, se programa de manera asíncrona. Con Angular, me he tenido que adentrar en el mundo de Typescript y, en el mundo de las aplicaciones modulares y escalables.

Durante el grado, han sido pocas las veces que he usado base de datos y, cuando las he usado ha sido de manera muy básica, con operaciones sencillas y usando siempre SQL. El utilizar MongoDB, me ha permitido aprender otra manera de guardar los datos a través de documentos, en vez de en tablas como hasta ahora. Además, esta aplicación me ha permitido realizar operaciones mucho más complejas, en las que en una misma operación tenía que tocar varias colecciones (tablas en base de datos relacionales). Para ello, he tenido que aprender a hacer agregaciones, que son funciones en las que se van transformando los datos a través de etapas hasta que se obtiene el resultado esperado.

Para realizar un diseño atractivo y que se adapte a los tamaños de pantalla más estandares, he aprendido a usar media queries de CSS3 y he profundizado en Bootstrap. Además, he conocido la librería Angular Material de Angular.

La importación y exportación de ficheros, así como, la subida de archivos al servidor de node me ha proporcionado conocimientos en las librerías adm-zip, para la decompresión de zip, excel4node para el manipulado de excels y, multer, para la subida de imágenes al servidor.

Por último, este trabajo he aprendido que los grandes logros se consiguen con constancia, dedicación y sacrificio y, que a pesar de todos los problemas que pueden aparecer en el camino, hay que levantarse y luchar por conseguirlos. Que la desesperación y los nervios nos buenos aliados. Que hay que confiar en uno mismo y no ponerse límites.

6.4. Trabajos futuros

Todas las aplicaciones web necesitan mantenimiento y la implementada en este trabajo no iba a ser menos. Algunas de las líneas a seguir para mejorar la aplicación son:

- Adaptar la aplicación para que soporte como segundo lenguaje, el inglés. De esta forma, se abriría el mercado al mundo internacional. Para realizar esta mejora, se debería utilizar la librería de angular Ngx-translate. Ngx-translate utiliza ficheros JSON donde

se encuentran los textos estáticos de la aplicación traducidos a los idiomas que se desee utilizar. Tiene que existir un fichero por cada idioma y, el formato de los ficheros debe ser igual, para que a la hora de usar el servicio TransalteService sepa el archivo y texto que se debe usar. Para los textos que varían en la aplicación como, por ejemplo, la descripción de los productos, se debería crear un campo nuevo dentro de la base de datos en el que se encuentre la traducción y a la hora de hacer la petición, mandar el idioma para saber que descripción obtener.

- Ampliar el filtrado de productos. A pesar del número de filtros que tiene la aplicación para la búsqueda de productos, se pueden añadir filtros que se adapten a las distintas categorías. Por ejemplo, si se accede a la categoría de televisores, que aparezca un filtro más para buscar por pulgadas.

- Añadir una lista de favoritos, donde el cliente pueda guardar los productos que le gustan. Así, si vuelve a entrar en la aplicación no tiene que volver a buscar los productos que desea.

- Inicio de sesión a través de Facebook y Google. De este modo, los clientes no tendrían que darse de alta en la aplicación bastaría que iniciasen sesión con sus credenciales de Facebook y Google.

En cuanto a la parte de administración se podrían introducir las siguientes mejoras:

- Añadir estadísticas y gráficos que ofrezcan información sobre los productos más vendidos, los menos demandados, así como la zona donde más pedidos se realizan. Tener esta información de manera resumida, en forma de gráficos, sería muy útil para el administrador y dueño, ya que sabría lo que debe reforzar para mejorar el servicio.

- Añadir una opción de impresión de cartelería. En la tienda física, se necesitan carteles que promocionen los productos, que contengan la información y precio de los mismos. Toda esta información está almacenada en la base de datos, por lo que con la impresión de estos carteles a través de la aplicación, se agilizaría el trabajo en tienda física.

- Mejorar el diseño responsive de la parte de administración. La aplicación en la parte de administración es responsive, pero en ciertas pantallas, los campos de las tablas salen más

amontonados, por lo que se podría reducir el número de campos a mostrar en la tabla para que dicha información se vea mejor.

Apéndice A

Manual de usuario

Esto es un apéndice. Si has creado una aplicación, siempre viene bien tener un manual de usuario. Pues ponlo aquí.

Bibliografía