

MAXBarter

David Esteve Vicente

Curso 21/22

I.E.S La Vereda

Desarrollo de Aplicaciones Web (DAW)

Tutor : MARTINEZ CARBONELL, M^a CARMEN

I.E.S. La Vereda

La Pobla de Vallbona - València





David Esteve Vicente



Índice

1. Introducción.....	4
1.1 Funcionalidades y características de la aplicación.....	5
1.2 Instrucciones de instalación.....	8
1.3 Requisitos mínimos.....	9
1.4 Módulos a los que implica.....	10
2. Estudio previo.....	13
2.1 Estudio de soluciones.....	14
3. Plan de trabajo.....	16
4. Diseño.....	18
5. Implantación.....	23
5.1 Entorno de desarrollo.....	23
5.2 Desarrollo.....	24
5.2.1 Servidor.....	25
5.2.1.1 Base de datos.....	25
5.2.1.2 Configuración.....	27
5.2.1.3 Estructura.....	28
5.2.1.4 Observaciones.....	31
5.2.2 Cliente.....	33
5.2.2.1 Estructura.....	34
5.2.2.1 Observaciones.....	39
5.3 Despliegue.....	40
6. Recursos.....	45
6.1 Herramientas de hardware.....	45
6.2 Herramientas de software.....	45
6.3 Sistemas operativos empleados.....	46
7. Conclusiones.....	46
7.1 Grado de consecución de objetivos.....	46
7.2 Problemas encontrados.....	47
7.3 Mejoras.....	47
8. Explicación contenidos de Github.....	48
9. Anexos.....	49
10. Bibliografía.....	50
11. Índice de figuras.....	51

1. Introducción

Somos muchos los que compramos gran variedad de objetos, ya sea por alguna necesidad en concreto o simplemente por capricho propio. Es una realidad que muchos de esos objetos no le sacamos el potencial que tienen, dejándolos de utilizar y haciendo que ocupen espacio innecesario en nuestro hogar. ¿Qué podría hacer para dar vida a esos objetos y además cubrir alguna necesidad?

Muy sencillo, existe gente con la misma necesidad que tú, gente que no utiliza un producto pero que sí necesite alguno tuyo que tampoco le estés dando utilidad. Siempre que exista una necesidad recíproca y estéis de acuerdo ambos, podéis hacer un **trueque** y abastecer vuestra necesidad de forma fácil y beneficiosa.



Dicho esto surge la idea de **MaxBarter** una aplicación web destinada a cubrir las necesidades descritas anteriormente. En el que se ofrecerán funcionalidades para facilitar el encuentro de usuarios interesados en hacer un trueque con sus productos.

Los usuarios tendrán la capacidad de publicar productos y crear ofertas para aquellos productos en los que estén interesados hacer un trueque.

Si el usuario al cual le han enviado la oferta está conforme y la acepta se habrá hecho un “**match**” que será notificado a ambos usuarios en su respectivo apartado. Por consiguiente, los usuarios se pondrán en contacto con los datos proporcionados en su perfil para efectuar el trueque.

1.1 Funcionalidades y características de la aplicación

Funcionalidades y características de la aplicación:

- ❏ **Gestión de autenticación.** Sistema de login y registro con el que se podrá acceder a la aplicación con el nombre de usuario y contraseña.
- ❏ **Interfaz fácil e intuitiva.** Permite al usuario hacer uso de ella de forma cómoda.
- ❏ **Publicación de productos.** Los usuarios registrados podrán publicar sus productos en la web, los cuales serán visibles para los demás usuarios.
- ❏ **Sistema CRUD de los productos.** Se les proporcionará un sistema CRUD (Create, Read, Update, Delete) de los productos a los usuarios. Por el cual podrán gestionar sus productos de forma sencilla. Dispondrán de una vista del producto a tiempo real (mostrada a los usuarios) cada vez que se quiera crear o modificar los productos.
- ❏ **Sistema de búsqueda.** Se podrá buscar por producto o por usuarios. La carga de los elementos será automática mediante la acción de scroll en la página. En cada apartado se dispondrá de filtros para facilitar la búsqueda al usuario. Los productos se podrán filtrar por título, comunidad autónoma o categoría.

- ❏ **Sistema de ofertas.** Se podrá realizar ofertas a otros productos, las cuales podrán ser gestionadas tanto las que enviamos como las recibimos. Podemos eliminar ofertas enviadas y aceptar o rechazar aquellas recibidas.
- ❏ **Geolocalización.** Se podrá saber a qué comunidad autónoma pertenecen los usuarios a través de la información aportada en sus perfiles. Además se dará la opción de poder establecer su ubicación exacta en el mapa que será mostrada en el apartado de sus perfiles.
- ❏ **Analítica y estadísticas.** Se dispondrá en los perfiles de los usuarios un apartado donde se mostrarán estadísticas de forma gráfica de datos relevantes como número de productos, de maths, ofertas rechazadas etc.
- ❏ **Sistema de matchs.** La aplicación gestionará cuando un producto ha hecho match, ya sea por aceptar una oferta o por el envío de una oferta de forma recíproca, es decir, que los dos usuarios estén interesados por el mismo producto y ambos se manden una oferta. La aplicación se encargará de hacer la lógica necesaria.
- ❏ **Personalización.** Una vez creada la cuenta el usuario tendrá acceso a modificar sus datos de contacto además de establecer su ubicación y modificarse la foto de perfil.
- ❏ **Responsive.** La aplicación está pensada para ser visualizada en todo tipo de dispositivos. Se ajustará al tamaño de los dispositivos existentes en el mercado.

- ❏ **Seguridad.** La aplicación está pensada para ofrecer al usuario una seguridad en sus datos. Las contraseñas de los usuarios permanecerán encriptadas y se otorgará una protección en las rutas para brindar confidencialidad en sus datos.
- ❏ **Sistema de Administración.** Existen usuarios administradores encargados de administrar y moderar la página. Serán dotados de permisos por lo que podrán modificar cualquier producto o información del perfil de los usuarios.
- ❏ **About Us.** La aplicación dispone de un apartado accesible por cualquier usuario, registrado o no, donde se mostrará una página de información explicando su funcionalidad.

1.2 Instrucciones de instalación.

La aplicación está dividida en dos, por un lado el back-end y por el otro el front-end, donde el back-end hará de API y se encargará de la gestión de la base de datos.

Por lo tanto, deberemos ejecutar las dos aplicaciones por separado:



Back-end:

1. Descargamos los archivos del repositorio de GitHub
“<https://github.com/DavidEsteve901/MaxBarter-Cliente.git>”.
2. Deberemos tener instalado en nuestro ordenador node.js y npm. Instalamos las dependencias haciendo uso del comando ***‘npm install’***.
3. Tener un sistema de gestión de bases de datos MySQL (recomiendo usar Xampp). Creamos una base de datos con el nombre que prefiramos y añadimos la configuración a las variables de entorno que se encuentran en el archivo *.env*.
4. Para poner en funcionamiento el servidor usamos el comando ***‘npm run dev’***.
5. (opcional) Si queremos hacer un seed a la base de datos con datos de prueba ejecutamos el comando ***‘npx sequelize-cli db:seed:all’***, después de haber cargado los modelos al arrancar el servidor.



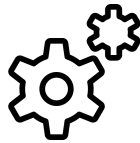
Front-end:

1. Descargamos los archivos del repositorio de GitHub
“<https://github.com/DavidEsteve901/MaxBarter-Servidor.git>”.
2. Deberemos tener instalado en nuestro ordenador node.js, npm y Angular. Instalamos Angular de forma global con ***‘npm install -g @angular/cli’***. Instalamos las dependencias haciendo uso del comando ***‘npm install’***.
3. Para poner en funcionamiento el cliente usamos el comando ***‘ng serve -o’***(-o opcional para que abra la pestaña en el navegador)



Teniendo en funcionamiento las 2 aplicaciones podremos hacer uso de la aplicación.

1.3 Requisitos mínimos



No serán necesarios grandes requisitos para el correcto funcionamiento de la aplicación ya que no hará uso de muchos recursos.

Para poder acceder a la web, su equipo debe cumplir los siguientes requisitos.

Sistema Operativo:

- ✓ *Windows*
- ✓ *Mac*
- ✓ *Linux/Ubuntu*

Navegadores web:

- ✓ *Google Chrome 72 o una versión más reciente*
- ✓ *Microsoft Edge 77 o una versión más reciente*

Conexión a internet:

- ✓ *1 Mbps como mínimo (se recomienda red de banda ancha)*
- ✓ *Conexión 3G como mínimo (se recomienda WiFi)*

Hardware:

- ✓ *Procesador de x86 o x64 bits de doble núcleo de 1,9 gigahercios (GHz) o más*
- ✓ *2 GB de RAM*

1.4 Módulos a los que implica

Para poder realizar la aplicación he tenido que hacer uso de conocimientos adquiridos de todos los módulos, ya que me han sido necesarios durante el desarrollo del proyecto. Algunos módulos han tenido más peso que otros pero todos han sido necesarios para el desarrollo en sus respectivas fases.

★ **Sistemas Informáticos.** Me ha sido útil a la hora de configurar el back-end en la máquina virtual del hosting. He necesitado acceder a ella por ssh y hacer uso de los comandos bash shell de Linux para su configuración, ya que la MV la he configurado con la distribución ubuntu de GNU/Linux.

★ **Base de datos.** Ha tenido un gran peso ya que gracias a los conocimientos adquiridos he podido crear lo que es la “base” de la aplicación donde se almacenarán todos los datos. He usado el sistema de gestión de base de datos MySQL (base de datos relacional) por lo que he seguido todos los pasos que me han enseñado para construirla de forma fiable y eficiente. Realizando la fase de recogida de necesidades, diseño conceptual, diseño lógico y diseño físico.

- ★ **Programación.** Una gran importancia a la hora de la codificación tanto para la parte del cliente como para el servidor. He usado los frameworks de Angular y Express.js donde he trabajado siempre con una programación orientada a objetos lo que me ha aportado una gran organización y legibilidad al código. He hecho uso de interfaces, clases, he optimizado el código, uso de array, estructuras de control etc.
- ★ **Lenguaje de marcas.** Fundamental para seguir una estructura de marcas. He hecho mucho uso de lo aprendido en este módulo en la parte del front-end ya que al usar el framework de angular toda mi aplicación se divide en componentes separados donde cada uno tiene su estructura html y su diseño css.
- ★ **Entorno de desarrollo.** He usado git como sistema de gestión de versiones facilitando y ahorrando muchos problemas, lo he usado desde consola y con un plugin desde el IDE de Visual Studio Code que es donde he desarrollado todo. También he usado la metodología del diagrama de Gantt donde me he representado las tareas que tenía que realizar en un periodo de tiempo en concreto.
- ★ **Diseño de interfaces.** Me ha ayudado mucho a la hora de hacer el diseño de la aplicación, he hecho uso del lenguaje de Sass para los estilos, un poco de JQuery, también he hecho uso de un poco bootstrap ya que los diseños los he personalizado a mano. Y gracias a las media queries he hecho responsive la aplicación para todos los tamaños posibles.

★ **Despliegue de aplicaciones.** Gracias a los conocimientos adquiridos he podido analizar las opciones de hosting que tenía y optar por la más adecuada a mis necesidades. He hecho uso de hosting AWS, y así poder desplegar mis aplicaciones por separado y conectarlas entre sí, por diferentes puertos y protocolos, además de usar tecnologías como docker y Nginx proxy manager para implementar el protocolo HTTPS en mi página. También he hecho uso de los DNS e implementando un dominio para mi web.

★ **Entorno cliente.** Sin duda uno de los módulos más importantes, ya que la aplicación la he desarrollado con el lenguaje de JavaScript. He hecho un uso constante de eventos, manejo del DOM, validaciones de formularios, expresiones regulares etc... También usando nuevos lenguajes como TypeScript para el tipado de la aplicación, ayudando en el desarrollo y entendimiento del código. Además he hecho uso del framework de Angular.

★ **Entorno servidor.** Módulo con gran peso en la aplicación, con lo que he podido estructurarme en métodos los diferentes accesos a la base de datos para modificar, editar o actualizar los datos. Pudiendo trabajar con los datos adaptándome a las necesidades. He utilizado el framework de express.js por lo que hago uso del lenguaje de JS y donde el servidor hará de API al frontend. Está pensado para adaptarse a nuevas necesidades de forma sencilla con enrutamiento de los métodos y seguridad gracias a los middlewares.

2. Estudio previo

Una vez tenida clara la idea de mi aplicación, me dispuse a plasmar las ideas que me iban surgiendo en forma de bocetos en mi libreta. Donde dibujaba y diseñaba tanto la parte del diseño como su parte lógica, dándome una idea de cómo se estructuraría la aplicación y qué pestañas tendría para resolver las necesidades que había pensado.



Figura 1: Estudio previo

Hice un estudio de qué tecnologías utilizar y procedimientos a seguir para tener una aplicación bien estructurada, organizada, fácil de mantener y de entender. Pensé varias tecnologías a usar, una de ellas fue usar el framework de **Laravel**, framework que usa el lenguaje de php y donde gestionaria el back-end y el front-end con su sistema de enrutamiento, modelo vista controlador, usaría el sistema de plantillas Blade para la parte del front-end, uso del ORM de eloquent para el manejo de la base de datos y otras tecnologías para el front-end como Alpine.js.

Otra opción era usar el framework de php **Symfony** el cual usamos en clase y ya tenía conocimientos de cómo funcionaba y no tendría que dedicar más tiempo a saber su funcionamiento.

Por otro lado, en clase también aprendimos un poco del framework **Angular**, framework el cual me pareció muy interesante por la forma de estructurar las páginas y la modularización en componentes que sigue, que con su correcto uso descompone la

aplicación haciéndola más entendible y más fácil de mantener ya que al organizarlo por separado ya sabes donde buscar.

Finalmente me decanté por usar **Angular**, ya que tenía muchas ganas de aprender a usarlo y he visto que es muy utilizado en el mundo del desarrollo web, teniendo un gran potencial. Angular al tratarse de un framework de front-end, necesitaba desarrollar el back-end con otra tecnología. Tras estudiar las opciones que tenía, me decanté por usar el framework de Node.js **Express.js**, framework que usa el lenguaje de JS y con el que podría hacer la lógica y gestión de los datos con la base de datos de forma independiente y comunicar ésta con mi front-end. También encontré un ORM para facilitar el manejo de la base de datos llamado **Sequelize**.

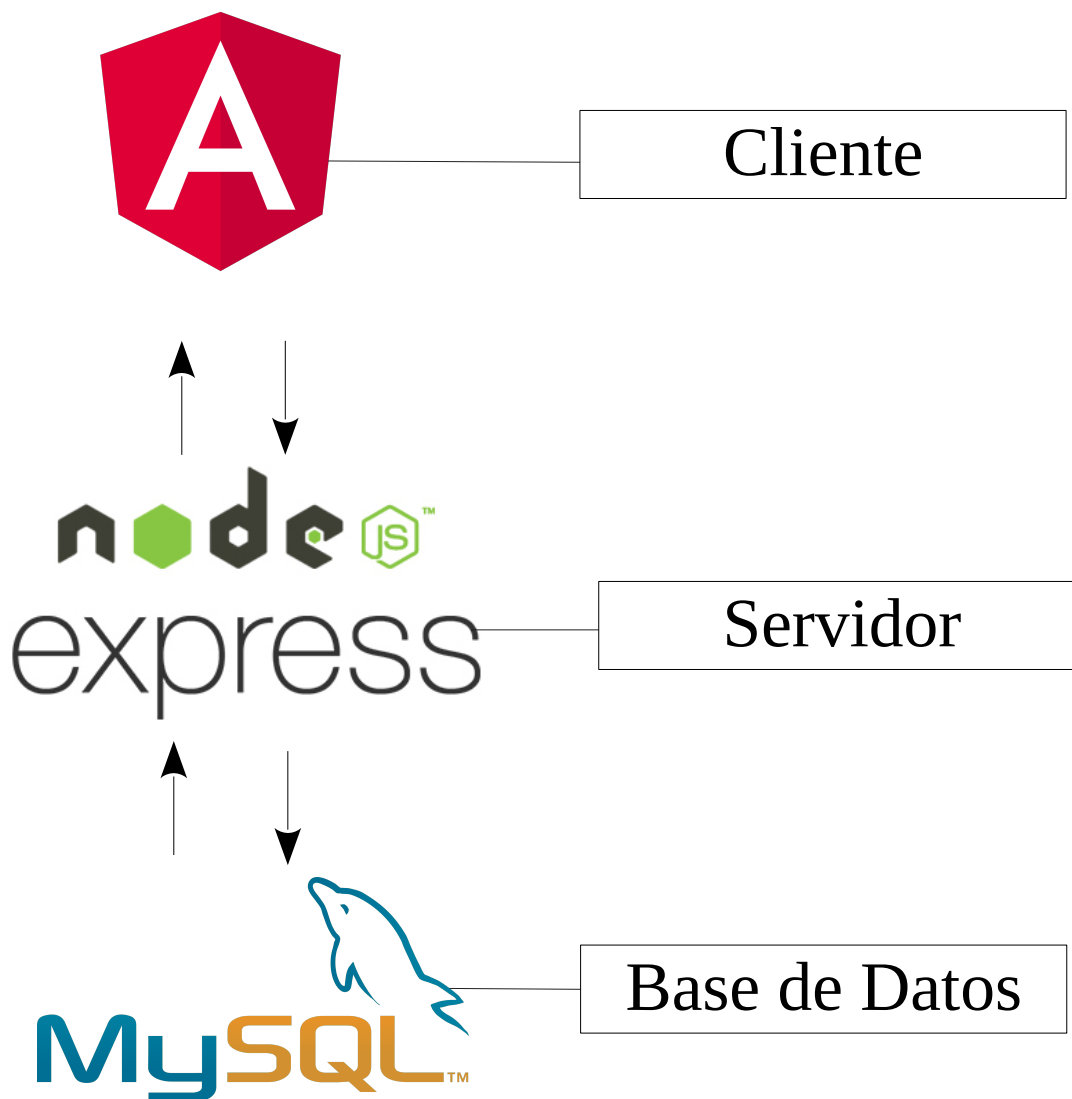
2.1 Estudio de soluciones

Tras decidir que quería usar Angular para mi aplicación, existía el problema de no saber qué hacer para desarrollar el back-end. Angular siendo un framework de front-end no disponía de herramientas para realizar el back-end cómo en las anteriores opciones que pensé.

Tras estudiar qué podía hacer, me topé con la opción de hacer la aplicación por separado usando el framework Express.js para el servidor. Es decir, tendría una aplicación que se encargaría de gestionar el cliente y otra del servidor. Las aplicaciones están conectadas mediante peticiones HTTP, por lo que mi servidor haría

de **API REST**. De esta manera el cliente mandaría peticiones a mi back-end y establecería una comunicación, siendo 2 aplicaciones por separado.

Otro inconveniente que he tenido que asumir es el **tiempo de estudio** de estas tecnologías, ya que no tenía conocimientos de ellas y he tenido que aprenderlas desde cero, estudiando cómo organizarse y todo lo que conlleva para cada una.



3. Plan de trabajo

Al tener un tiempo límite para desarrollar el proyecto, es muy importante tener una organización del desarrollo. Estableciendo un tiempo previsto por cada objetivo que te marques en la aplicación.

Una muy buena opción de establecer una organización es mediante la metodología de trabajo del **diagrama de Gantt**. El cual consiste en dividir los objetivos aportando un tiempo estimado por cada uno, pudiendo ver el progreso que llevas durante el desarrollo.

Por lo cual comencé con el estudio de los objetivos que quería establecer. Me planté 5 objetivos/fases:

- Planning
- Design
- Implementation back-end
- Implementation front-end
- Testing

Donde establecí las fechas estimadas para su finalización. Hice un excel programado para controlar los días que me faltaban por cada objetivo y así ver el estado de los mismos plasmados en una gráfica. De forma **automática** se cambia el progreso de los objetivos y facilitar su seguimiento.

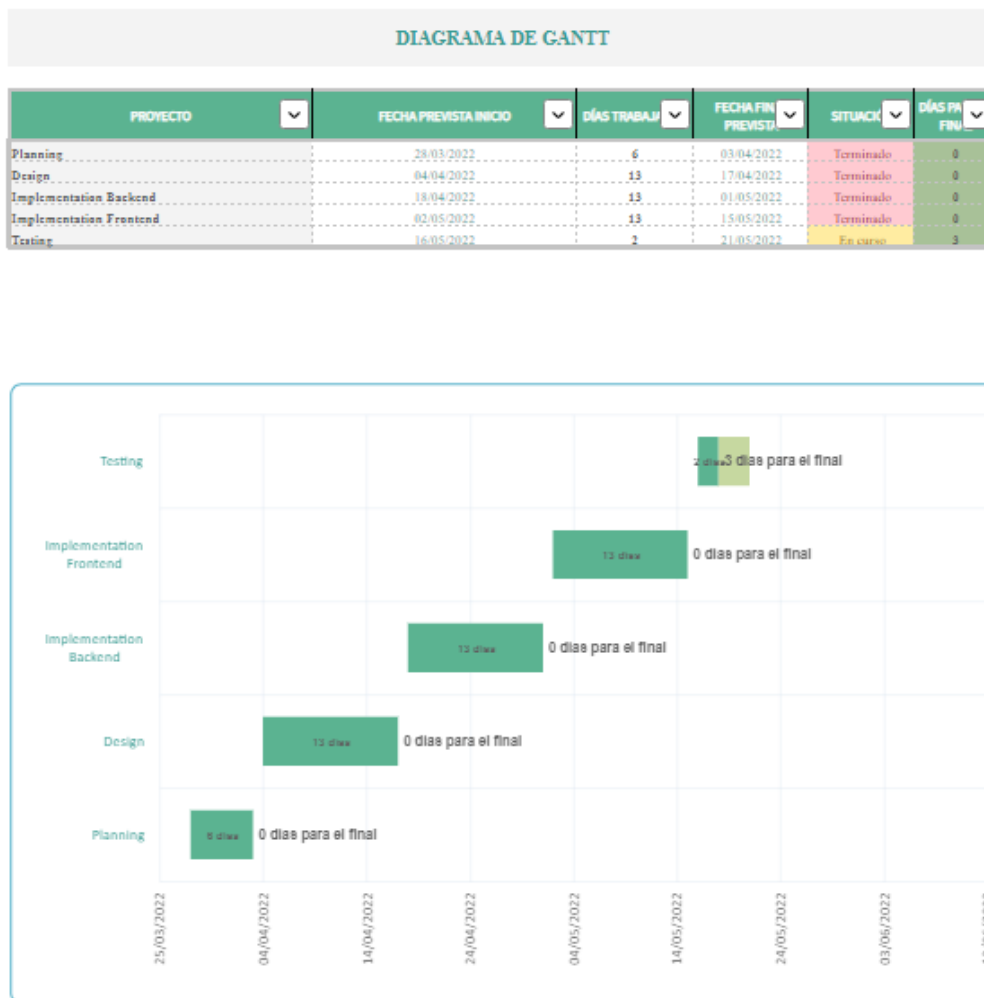


Figura 2: Excel programación de tareas



El enlace al excel se encontrará en los anexos, ***Para acceder al vínculo necesitas ser usuario de consellería, ya que lo hice con la herramienta de excel de microsoft y no deja ponerlo público***

4. Diseño

Como mi aplicación está enfocada al público su diseño es un punto muy importante, debiendo ser atractiva e intuitiva. Tras estudiar qué opciones tenía para crear un diseño basado en mis ideas iniciales y las que iban surgiendo, me topé con la herramienta de **Figma**. Un editor de gráficos vectorial y una herramienta de generación de prototipos, principalmente basada en web, con múltiples características que me garantizaban obtener los resultados que buscaba.



Figma me dio la capacidad de crear el prototipo de mi aplicación, personalizar cada página a mi gusto y crear dibujos propios usados en la web (ejem. Logotipo, iconos de flechas etc).

Mi objetivo era crear un estilo característico y moderno en el que destacase el color naranja (color principal de la marca) y se viese presente en los diferentes apartados de la web. Otro punto que tuve en cuenta era que tenía que ser muy intuitiva y fácil de utilizar, por lo que el flujo de botones y de navegación en la web debía ser adecuado para el usuario.

Una vez teniendo los bocetos en mano y habiendo aprendido a cómo usar Figma comencé a diseñar cada pestaña de la web. Otra ventaja que me aportó Figma es que puedes estructurar los diseños como componentes, lo cual me vendría perfecto para el framework de Angular y en la fase de desarrollo sólo tendría que pasarme los diseños ya creados en Figma a cada componente.



Se puede acceder a la vista de figma mediante el siguiente link:

<https://www.figma.com/file/ItxXfYSrNz9D294LAci0de/Proyecto-MaxBarter?node-id=0%3A1>

Desde el link se puede acceder a mi plantilla de figma y ver los diseños, donde apporto comentarios explicando la funcionalidad de cada pestaña.

➤ Diseño de las pestañas

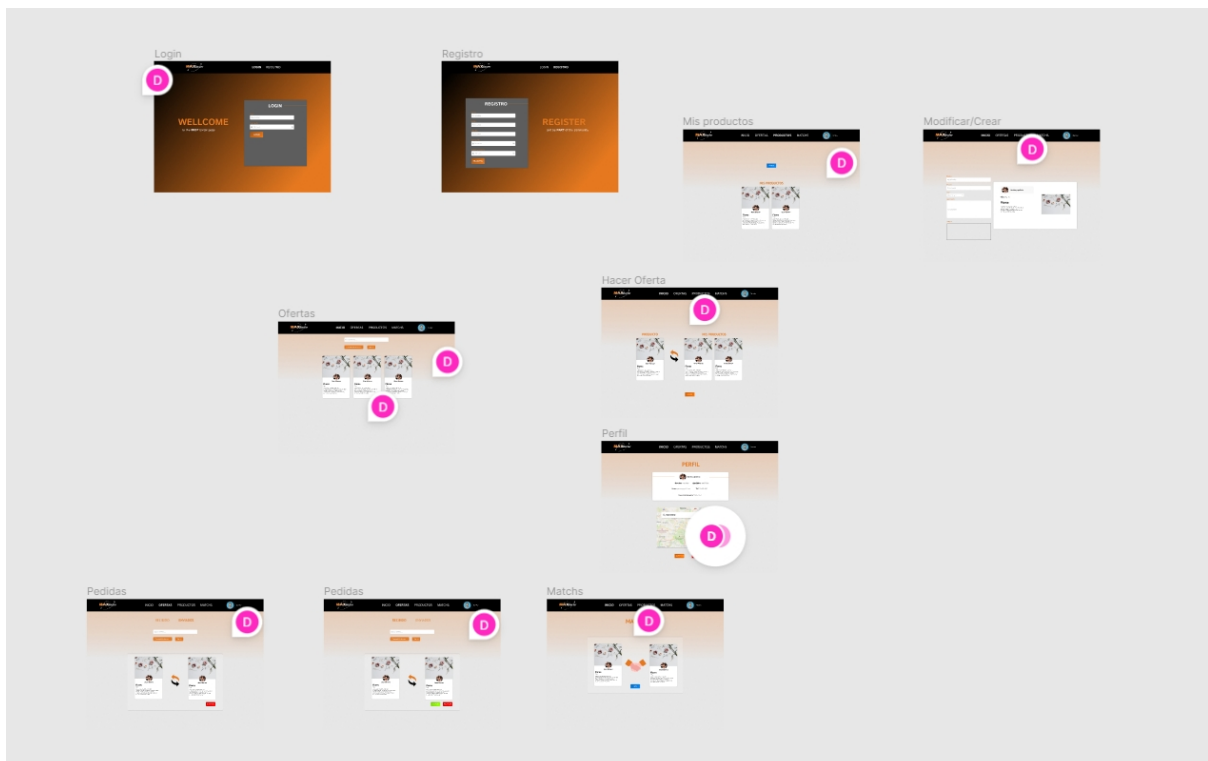


Figura 3: Diseño pestañas

➤ Diseño de los componentes

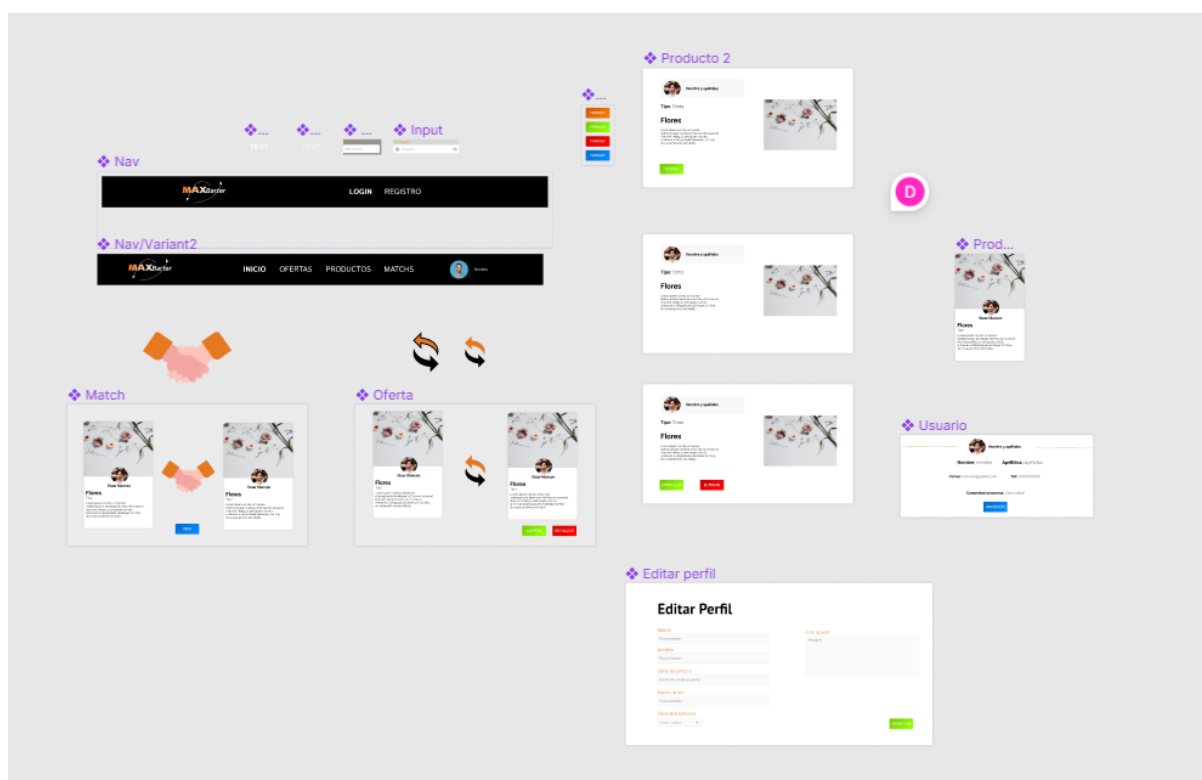


Figura 4: Diseño componentes

El prototipo también dispone de una versión donde se podrá visualizar de manera **interactiva** cómo funciona el flujo de botones y la **navegación** en la web. El Cliente podrá hacerse una idea de cómo será la aplicación.

➤ Prototipo interactivo

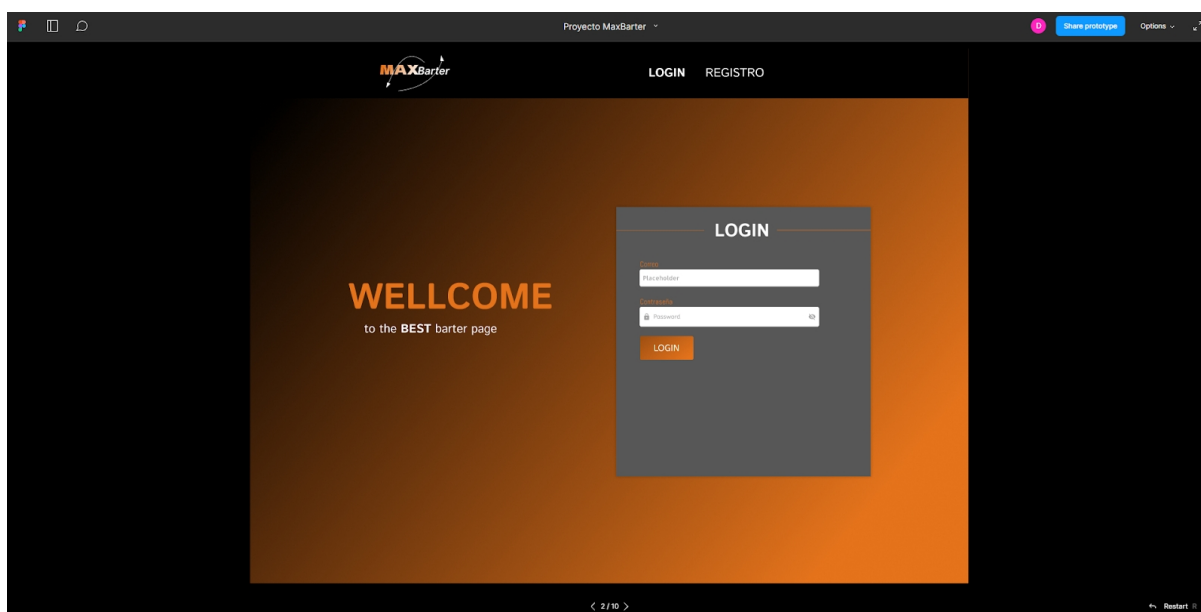
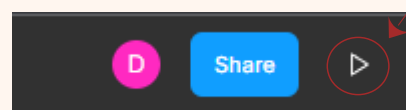


Figura 5: Prototipo interactivo



Para poder visualizar el diseño interactivo habrá que hacer click en icono de play, situado en la esquina superior derecha del diseño



Así es como se vería el flujo de botones desde la plantilla

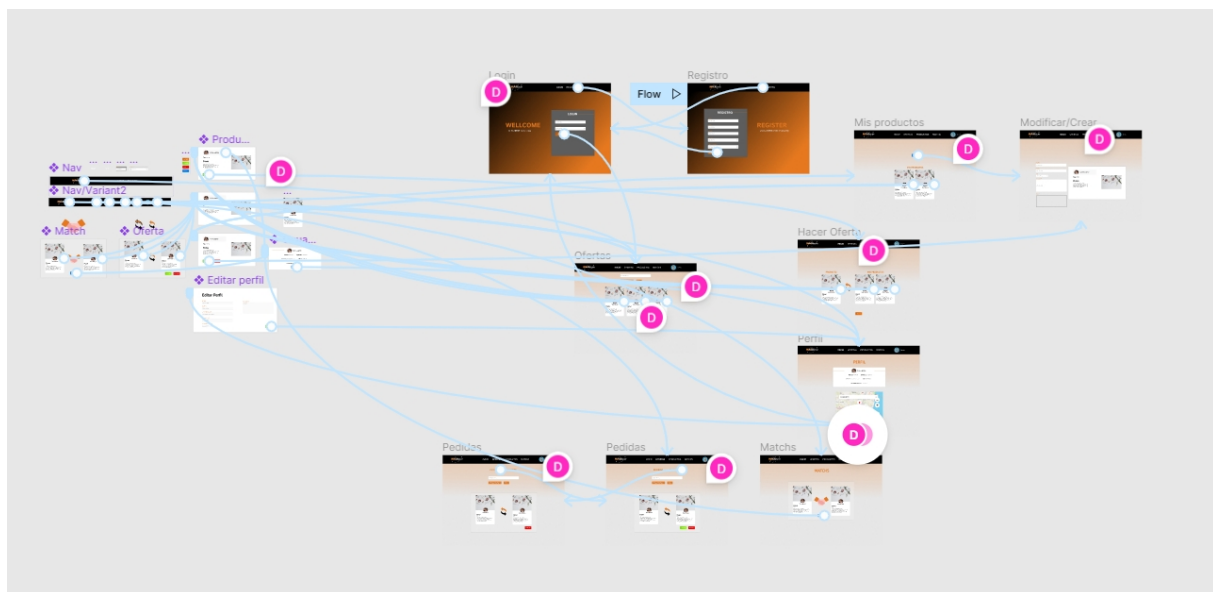
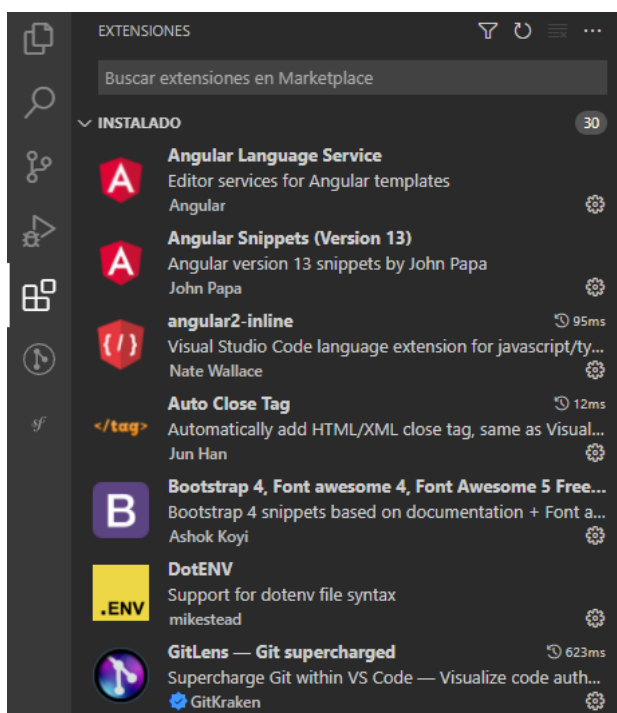


Figura 6: Flujo de navegación

5. Implantación

5.1 Entorno de desarrollo

Para el desarrollo de la aplicación he usado el IDE **Visual Studio Code**, sirviéndome de plugins/extensions que me aportan facilidades para el desarrollo.



Extensiones de autocompletado para las diferentes tecnologías que uso.

Para el control de versiones he usado **Git**, vinculándolo al repositorio remoto que se sitúa en mi cuenta de GitHub. De este modo puedo tener un control de las versiones de la aplicación y aportar seguridad durante el desarrollo en el caso de que surja algún problema. Trabajando de forma local y publicando los cambios al remoto.

5.2 Desarrollo

La aplicación se encuentra dividida en dos diferentes, una se encarga del back-end/servidor y la otra del front-end/Cliente. Ambas se comunicarán mediante peticiones **HTTP**, por lo que se desglosará la funcionalidad.

Por lo tanto, tendremos una aplicación que gestionará el apartado de cliente y otra que hará de API a la de cliente, que gestionará la parte del servidor y la base de datos. Ambas se transferirán la información en formato **JSON** mediante peticiones HTTP.

Tendremos una arquitectura **REST**, con las siguientes características:

- Arquitectura **cliente-servidor** compuesta de clientes, servidores y recursos, con la gestión de solicitudes a través de HTTP.
- Comunicación entre el cliente y el servidor **sin estado**, lo cual implica que no se almacena la información del cliente entre las solicitudes de GET y que cada una de ellas es independiente y está desconectada del resto. Lo que, por un lado, podría parecer una desventaja “la tediosa tarea de repetir los datos” es en realidad uno de sus puntos fuertes: al no almacenarlos, permite una mayor escalabilidad. No serán necesarios servidores tan potentes, capaces de almacenar todos los estados de sus clientes.
- Datos que pueden **almacenarse en caché** y optimizan las interacciones entre el cliente y el servidor.
- Una interfaz uniforme entre los elementos, para que la información se transfiera de forma estandarizada.
- Un sistema en capas que organiza en jerarquías invisibles para el cliente cada uno de los servidores (los encargados de la seguridad, del equilibrio de carga, etc.) que participan en la recuperación de la información solicitada.

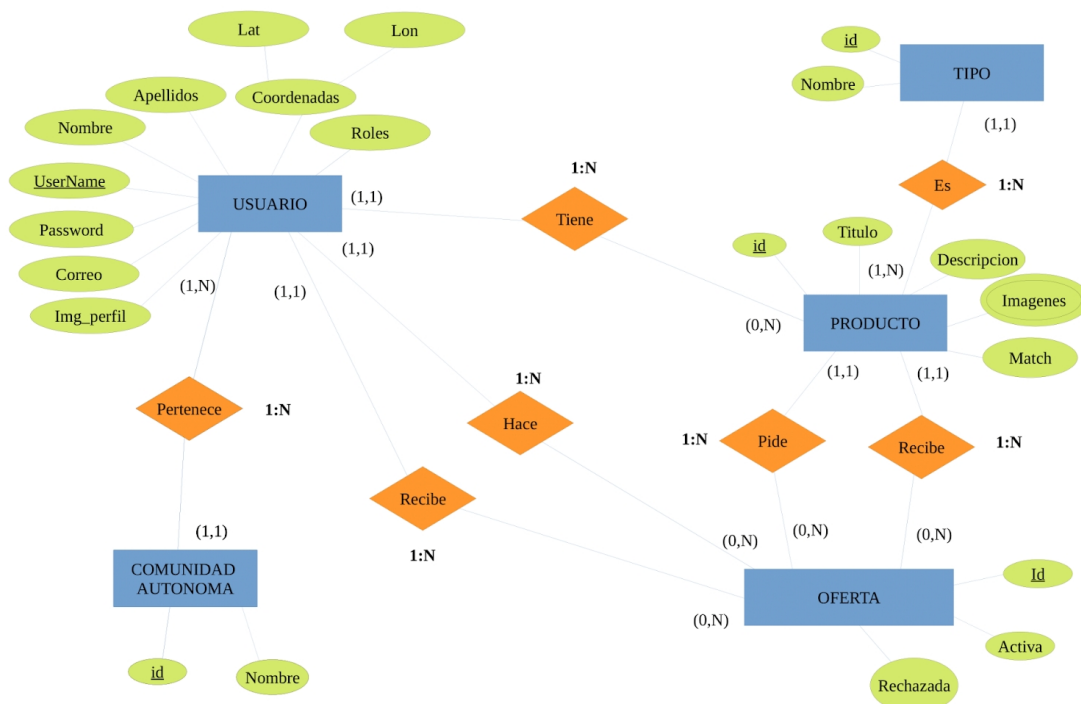
5.2.1 Servidor

Para el desarrollo de la aplicación del servidor he usado el framework **Express.js** de Node.js, siguiendo una arquitectura **MVC** (Modelo Vista Controlador). En este caso cómo el servidor hace de API el encargado de crear las vistas será la aplicación de Cliente, esta se encargará de la gestión de los modelos con la base de datos y del enrutamiento de los controladores para las peticiones http recibidas por el cliente.

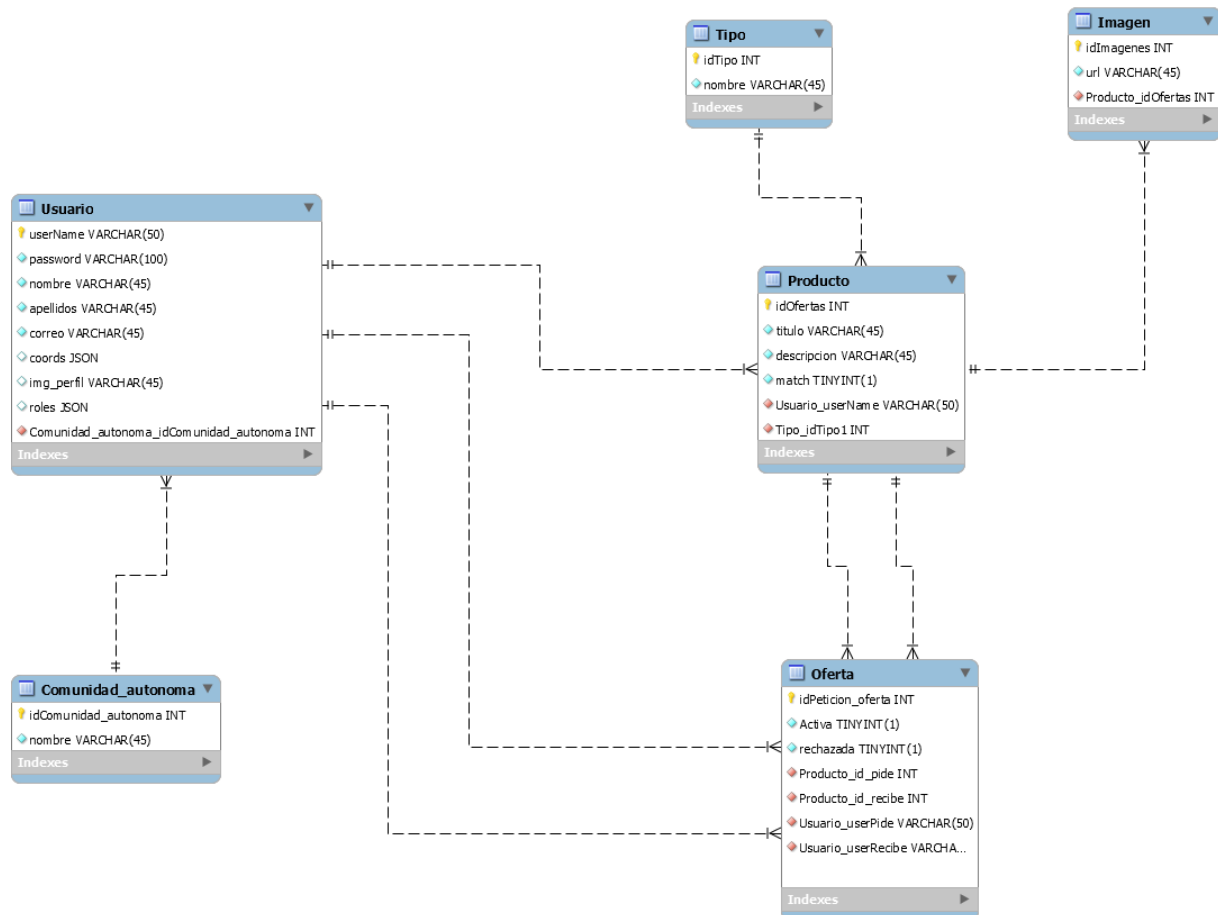
5.2.1.1 Base de datos

Antes de comenzar a desarrollar el servidor tenía que definirme qué estructura de datos tendría mi aplicación. Por lo que tras hacer un **estudio previo de las necesidades** diseñé el modelo conceptual de ER (Entidad Relación), hice su respectiva normalización y pasé al modelo lógico.

➤ Modelo ER (Entidad Relación)



➤ Modelo lógico



5.2.1.2 Configuración

Para el desarrollo del servidor me ha hecho falta la instalación de varios módulos/dependencias con su respectiva funcionalidades

- **Express.** Framework de node.js que me permite manejo de rutas (direccionamiento), archivos estáticos, uso de motor de plantillas, integración con bases de datos, manejo de errores, middlewares etc.
- **Babel.** Herramienta que nos permite transformar nuestro código JS de última generación (o con funcionalidades extras) a un código de Javascript que cualquier navegador o versión de Node. js pueda entender.
- **Bcryptjs.** Para el hashing de las contraseñas.
- **Cors.** Para configurar los cors.
- **Dotenv.** Para el manejo de variables de entorno.
- **Helmet.** Para aportar seguridad.
- **Jsonwebtoken.** Una herramienta que nos permite **autenticarnos** con el servidor **mediante Tokens** de una forma simple y segura.
- **Multer.** Para el manejo del almacenamiento de las imágenes.
- **Nodemon.** Para a la hora del desarrollo no tener que ir reiniciando el servidor cada vez que se haga un cambio y sea automático.
- **Mysql2.** Base de datos relacional que he usado.

5.2.1.3 Estructura

Para el desarrollo y posterior escalabilidad de la aplicación es muy importante que esté bien organizada y estructurada. Me voy a centrar en la **jerarquía de archivos** de la aplicación y qué funcionalidad tienen los archivos más relevantes.

Ficheros

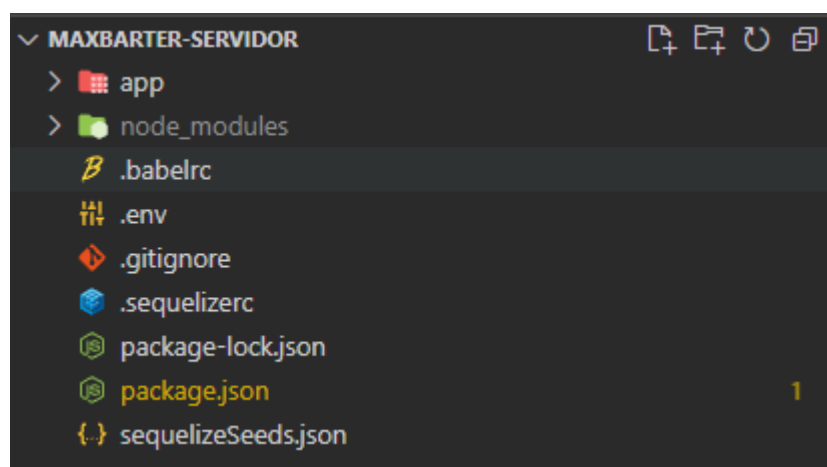


Figura 7: Ficheros servidor



App Directorio con los archivos de la aplicación.

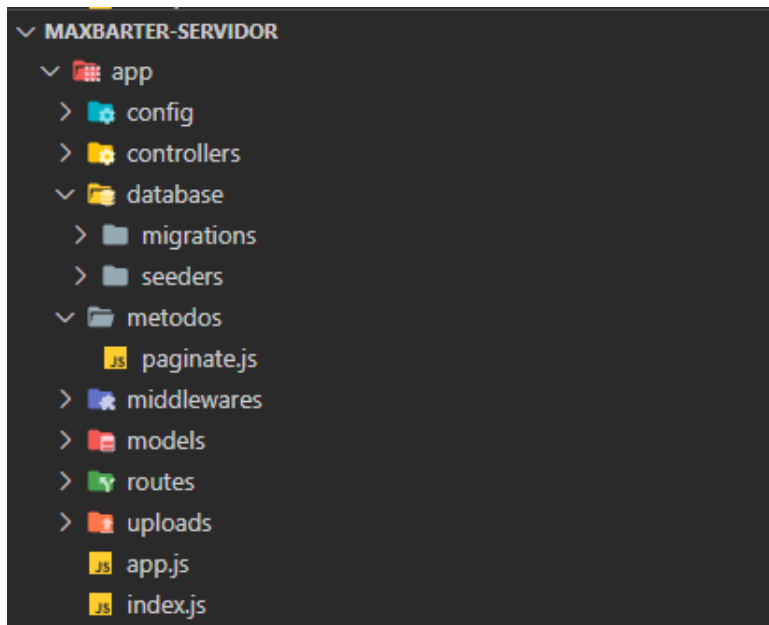


.env Archivo donde se encuentran las variables de entorno.




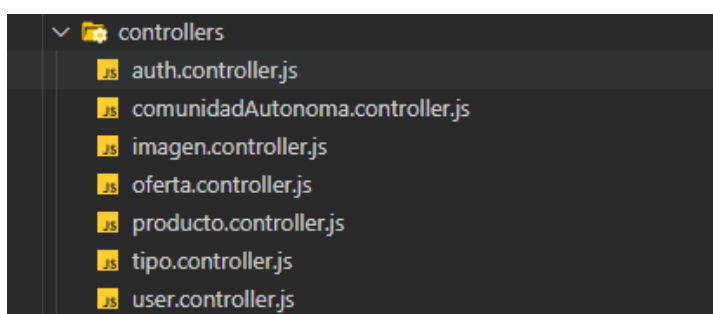
Package.json Almacena las dependencias del proyecto y donde tiene los scripts que usaremos para el arranque de la aplicación. Su finalidad es mantener un historial de los paquetes instalados y optimizar la forma en que se generan las dependencias del proyecto y los contenidos de la carpeta `node_modules/`.

Directorio app





 **Config** Donde se encuentran los archivos de configuración, en este caso la configuración a la base de datos.

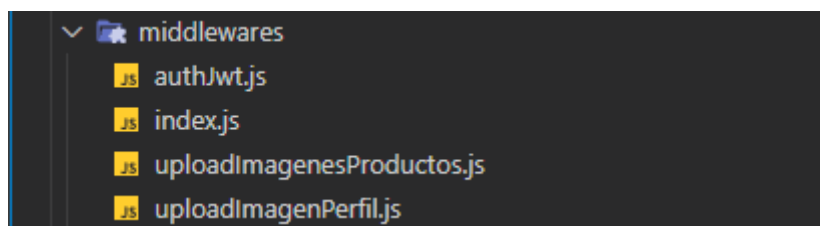
 **Controllers** Se sitúan los controladores de los modelos, cada modelo tendrá su respectivo controlador y sus métodos. Los controllers controlan la lógica de cada ruta.




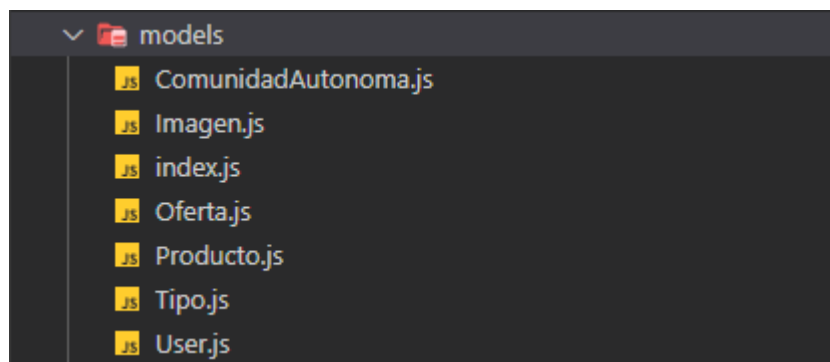
 **Database** Donde se encuentran las migraciones y los seeders para la base de datos.


 **Métodos** Directorio donde almaceno los métodos personalizados que me han hecho falta durante la aplicación. (en este caso un método genérico para que me devuelva los datos paginados, usado para el desarrollo del scroll infinito)

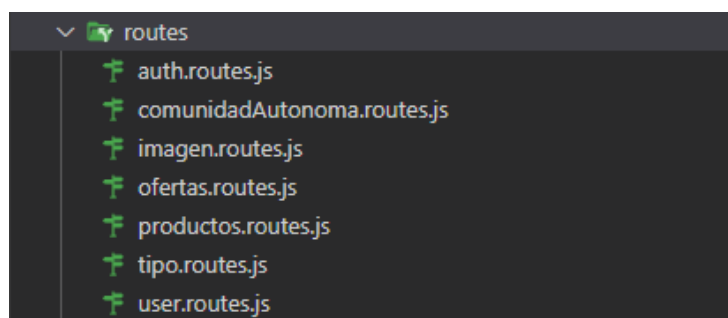
 **Middlewares** Se encuentran los middlewares que utilizo para las rutas. Middlewares para la seguridad de las rutas y la subida de imágenes.



 **Models** Se sitúan los modelos de la base de datos, con sus respectivos atributos y relaciones. Uso el **ORM sequelize** para las interacciones con la base de datos.



 **Routes** Se encuentran los archivos que definen las rutas para cada modelo donde se establece sus middlewares y controladores.





Uploads Donde se almacenan las imágenes subidas al servidor.



App.js Donde se establece la configuración de la aplicación importando módulos, definiendo las rutas etc.



Index.js Establece la conexión a la base de datos.

5.2.1.4 Observaciones

Como se ha podido apreciar la aplicación está muy modularizada para que sea fácil de entender, mantener y escalar. Todas las rutas están protegidas para que no se pueda acceder a ellas y modificar la base de datos de forma maliciosa. Cada ruta está protegida con la lógica necesaria, por ejemplo, un producto sólo podrá ser modificado si se está autenticado y el usuario es el propietario o es un administrador.

El sistema de autenticación está hecho mediante el uso de **tokens**. Para la interacción con la base de datos he usado el **ORM sequelize**, donde he creado los modelos con sus respectivos seeders.

Para facilitar las pruebas de las peticiones http que realiza el servidor he usado la aplicación **Postman**, que nos permite realizar pruebas API. Es un cliente HTTP que nos da la posibilidad de testear 'HTTP requests' a través de una interfaz gráfica de usuario, por medio de la cual obtendremos diferentes tipos de respuesta.

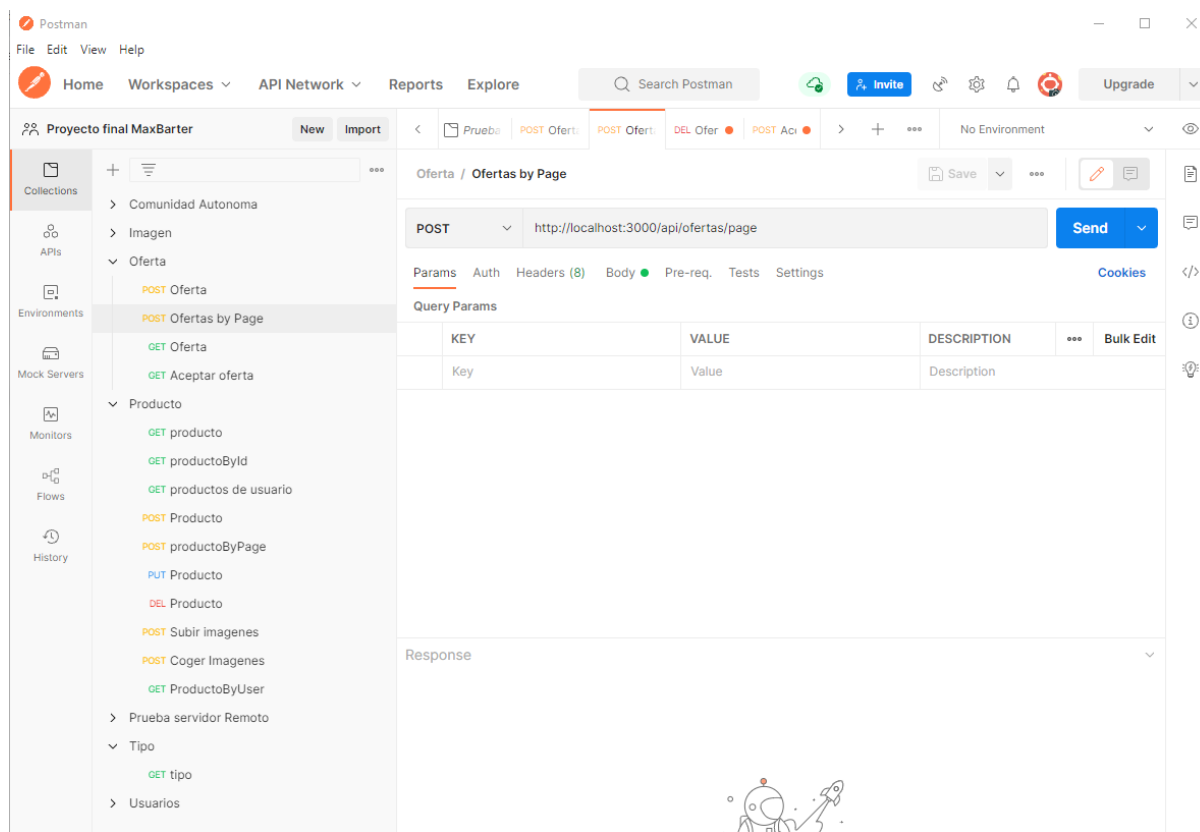


Figura 8: Postman

He estructurado en carpetas las diferentes peticiones que realiza mi aplicación, tanto las GET, POST, PUT, DELETE de cada modelo, para tener un entorno de prueba fácil de usar y garantizar que se obtengan los resultados esperados por parte de la API.

Gracias al sistema de enrutamiento puedo comunicarme con la aplicación de cliente mediante las peticiones http y así puedo escalar el servidor creando aquellas rutas que me van hacer falta para las funcionalidades de mi aplicación.

5.2.2 Cliente

Para el desarrollo del cliente he usado el framework **Angular**, el cual se encargará de crear las vistas y enviar peticiones al servidor para obtener y manejar datos. Este framework ofrece la capacidad de modularizar nuestra aplicación dando gran escalabilidad y organización. Hace uso del lenguaje de Type Script, que permite escribir y generar código de JavaScript que opera de manera fuertemente tipada y orientada a objetos.

Angular se caracteriza por el uso de componentes, los cuales se componen de:



Su template



Su estilo propio



Su parte lógica

Cada sección tendrá sus ficheros de componentes, guards (parecido a los middlewares), sus páginas, servicios, su módulo y módulo de rutas. He usado el padrón de diseño **Lazy Loading**, consiste en retrasar la carga o inicialización de un

objeto hasta el mismo momento de su utilización. Esto contribuye a la eficiencia de la aplicación, evitando la precarga de objetos que podrían no llegar a utilizarse.

Angular hace uso de **observables** que sirven para hacer peticiones **asíncronas** (muy parecido a las promesas). También hago uso de **Subjects** para la comunicación entre componentes y muchas más herramientas.

Durante el desarrollo he hecho uso de varias librerías como PrimeNG, Bootstrap, aos, rellax, ngx-infinite-scroll, animate.css, font awesome, librerías de angular (formularios reactivos, ng Model...) etc. Las cuales me han ayudado a la implementación de funcionalidades y animaciones de la aplicación.

5.2.2.1 Estructura

Al igual que el servidor he hecho hincapié en la organización y estructuración de la aplicación, tiene una gran importancia ya que consigues una mayor legibilidad de la aplicación y a futuro será fácil mantenerla y escalarla. Por lo tanto, me centraré en la **jerarquía de archivos** que he seguido y explicaré los archivos más relevantes.



No profundizaré en el funcionamiento de angular porque si no se hará muy extenso y no viene al caso.

Ficheros

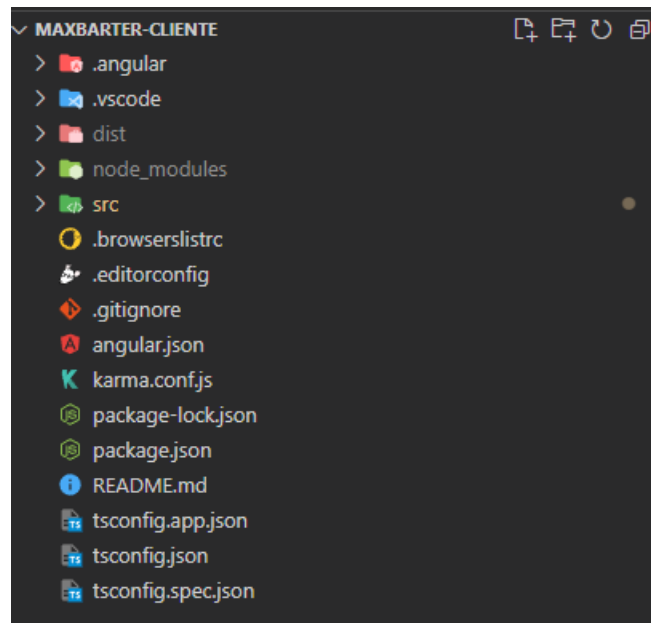



Figura 9: Ficheros Cliente

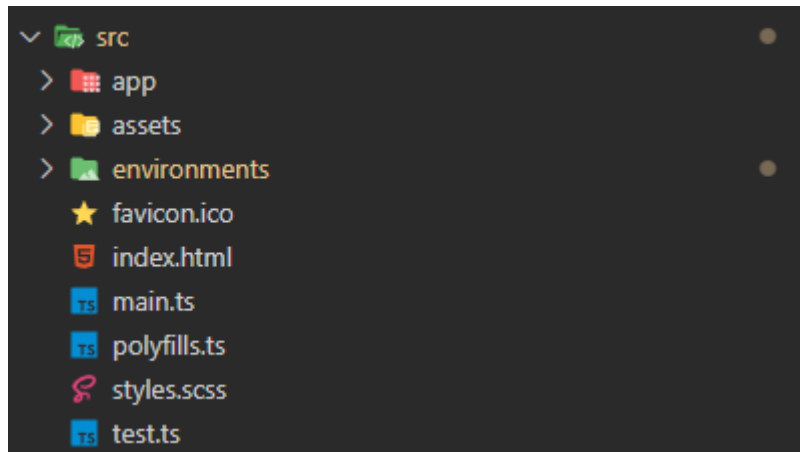
 **Package.json** almacena las dependencias del proyecto y donde tiene los scripts que usaremos para el arranque de la aplicación. Su finalidad es mantener un historial de los paquetes instalados y optimizar la forma en que se generan las dependencias del proyecto y los contenidos de la carpeta `node_modules/`.

 **Angular.json** Archivo con la configuración de angular.

 **Tsconfig.json** Archivo con la configuración del lenguaje Type Script.

 **Src** Directorio que contiene a la aplicación.

Directorio src



Assets Contiene las imágenes usadas en la aplicación.



Environments Almacena las variables de entorno, las de desarrollo y de producción.



Styles.scss Se encuentran los **estilos generales** de mi aplicación, donde he creado mis estilos personalizados de los componentes, los inputs, botones etc.

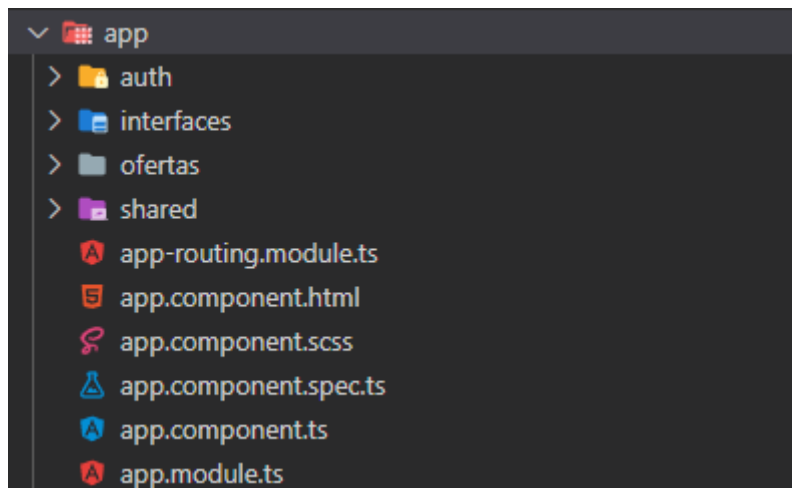


Index.html La estructura principal de la aplicación.




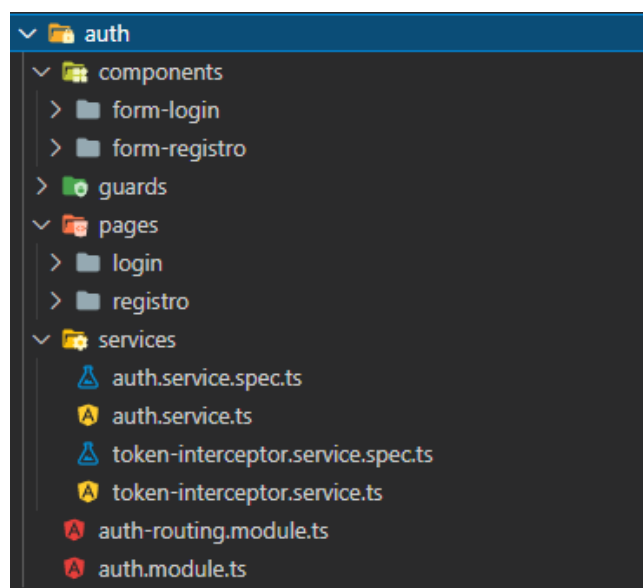
App Directorio más importante, que contiene los apartados de la aplicación.

Directorio app



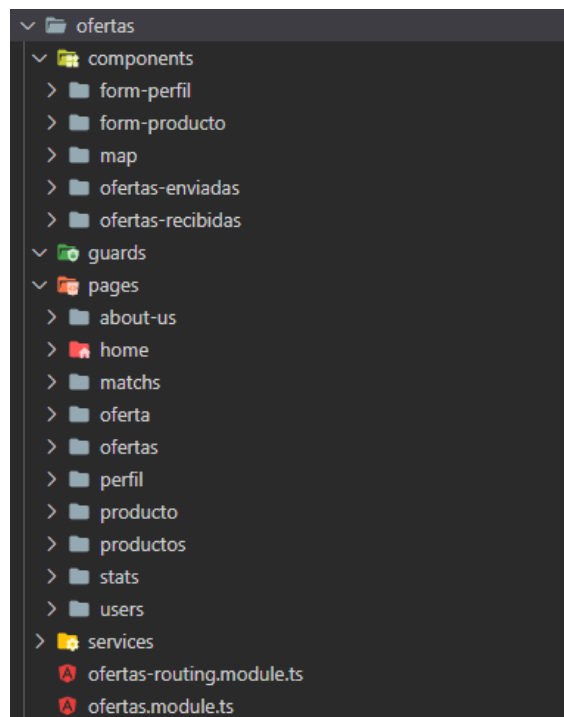
Como se puede observar, el directorio app dispondrá de archivos “padres/generales”. Dependiendo de la ruta en la que estemos se mostrarán los componentes encargados de gestionar la ruta, llamados en `app.component.html`.


 **Auth** Directorio que contendrá la sección de autenticación, con sus respectivas páginas, servicios, componentes, enrutamiento etc. La lógica correspondiente a la autenticación se encontrará en el directorio.

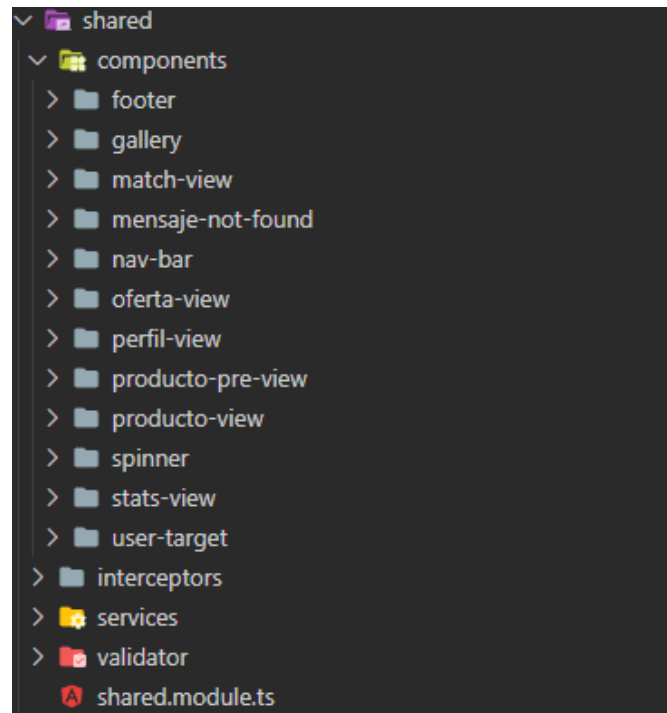


 **Interfaces** Donde almaceno las interfaces de los objetos que usaré durante mi aplicación

 **Ofertas** Directorio muy **importante**, donde gestiono gran parte de los apartados de la aplicación



 **Shared** Directorio donde almaceno todos aquellos componentes, servicios, interceptors y validaciones que vayan a ser usados en cualquier parte de la aplicación. Además, en su módulo importo los componentes de primeNg y otras librerías que utilizo, evitando hacer extenso el módulo principal de la aplicación que por normalización tiene que ser lo menos extenso posible.



5.2.2.1 Observaciones

La estructuración permite poder ser fácil de escalar y poder añadir más funcionalidades en un futuro. El sistema de enrutamiento permite modularizar la aplicación con la tecnología **Lazy Loading** donde cada ruta será gestionada por un componente. Las rutas están protegidas por guards, por lo que se deberá estar logueado para acceder a ciertas rutas

Durante el desarrollo hago uso de muchas tecnologías y librerías para conseguir los resultados que buscaba. Gracias a como está modularizada la aplicación consigo no crear una sobrecarga de componentes y aumento el rendimiento.

Hago uso de los **interceptors**, por donde gestiono las peticiones HTTP de la aplicación. Gracias a ello consigo que por cada petición siempre se mande el token de la sesión del usuario almacenado en el local storage, para que el servidor lo decodifique y gestione si se tiene permiso para esa petición. También los he utilizado para la creación del **spinner** y que solo se muestre mientras se esté procesando alguna petición.

Como he mencionado antes, todos los componentes se compondrán de sus propios estilos, estructura y la parte lógica.

5.3 Despliegue

Para el despliegue de mi aplicación era necesario disponer de un servicio de hosting el cual me ofreciese espacio suficiente para almacenar mis dos aplicaciones, en especial el servidor, donde se almacenarían todas las imágenes de los productos y perfiles. Además que me garantizase fiabilidad y un buen rendimiento.

Por lo que hice un estudio de qué hosting podía utilizar, priorizando la opción de que sea gratuito. Estuve mirado hostings como **000.webhost.com** , **freeHosting** etc. Los

cuales no me acaban de ofrecer lo que buscaba. Hasta que me topé con **AWS** (Amazon Web Services) donde puedo disponer de una gran variedad de servicios de forma gratuita durante el primer año.

Gracias la gran cantidad de servicios que ofrece, sin costos y que aseguraba usar los servidor de Amazon que funcionan muy bien y me parecen fiables, decidí despegarla allí.



Para desplegar la **base de datos** usé el servicio de **RDS**, proporciona una selección de tipos de instancias optimizadas para diferentes casos de uso de bases de datos relacionales. Los tipos de instancia abarcan varias combinaciones de capacidad de CPU, memoria, almacenamiento y redes.

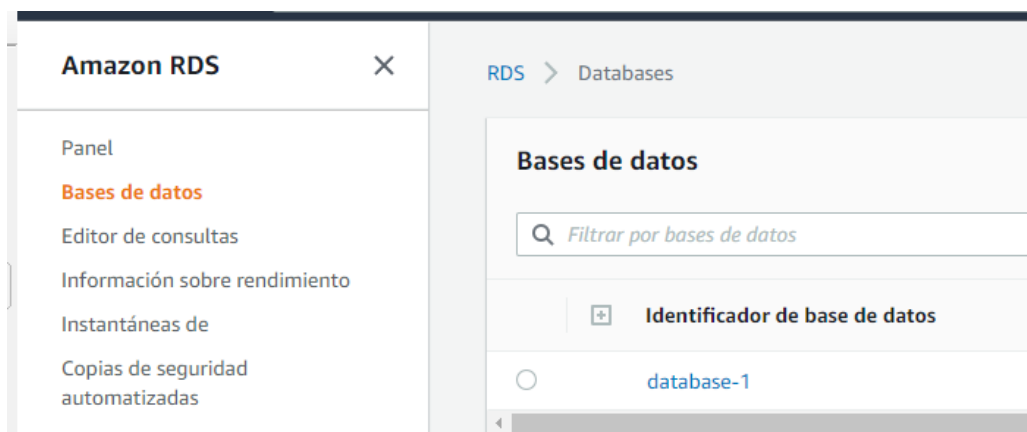


Figura 10: Servicio RDS de AWS

Para desplegar el cliente use el servicio de **S3**, un servicio de almacenamiento de objetos que ofrece escalabilidad, disponibilidad de datos, seguridad y rendimiento.



Figura 11: Servicio S3 de AWS

Para el desligue del servidor user el servicio de **EC2**, permite alquilar computadores virtuales en los cuales pueden ejecutar aplicaciones. Es decir, me proporcionan un hardware en la nube como una máquina virtual en la que puedo acceder por **ssh** y configurarla a mi antojo.

Configuré todo lo necesario para desplegar mi servidor, me traje el repositorio de Github en la máquina. Al ser un servidor en la nube y estar siempre activo, arrancando mi aplicación desde la máquina se encontrará siempre activa.

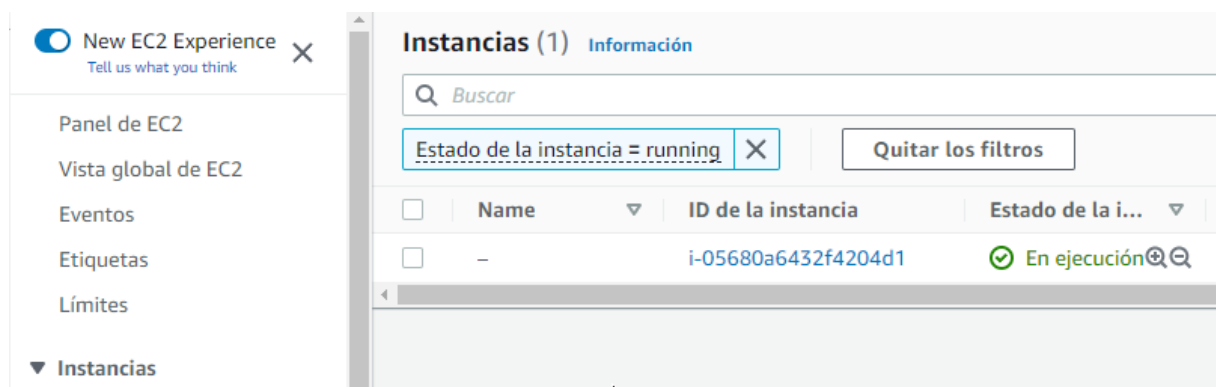


Figura 12: Servicio EC2 de AWS

Amazon me ofrece un enlace donde puedo acceder a mi cliente y al servidor, pero el enlace es muy largo y poco atractivo. Por lo que registré un dominio desde el servicio **FreeNom**, registré el dominio de **maxbarter.tk**

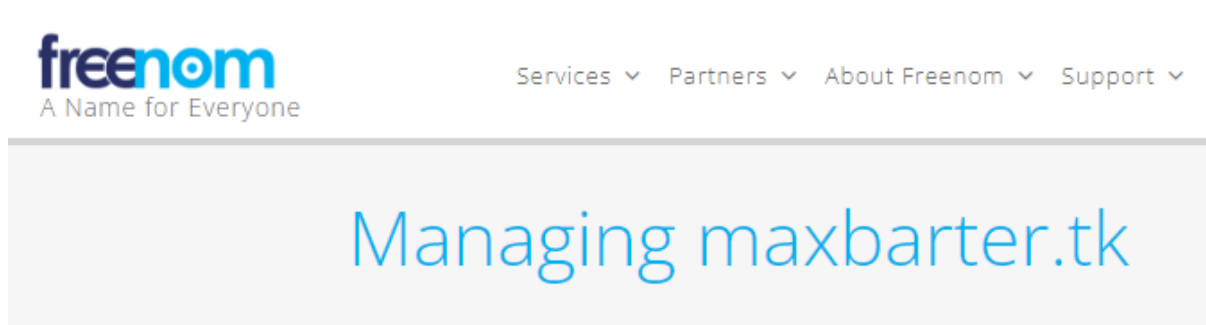


Figura 13: Dominio de freenom

Para la gestión de mis subdominios usé el servicio de Amazon llamado **Route 53**, conecta de forma efectiva las solicitudes del usuario con la infraestructura en ejecución en AWS. Por lo que cambié el servidor DNS de Freenom por el de AWS.

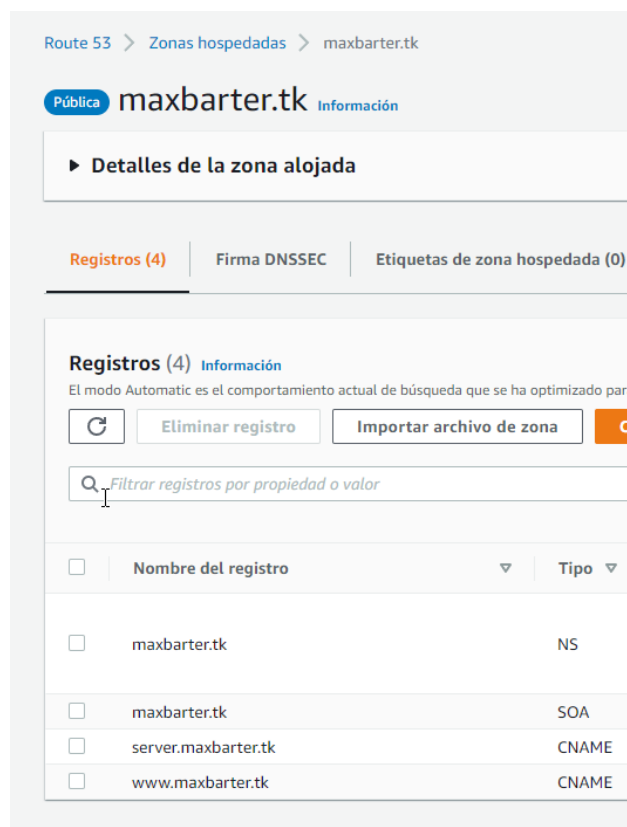


Figura 14: Servicio Route 53 de AWS

Mi dominio funciona por http, para poder añadirle el **protocolo SSL** tuve que usar mi maquina virtual del servicio EC2 e instalarme **docker** con la imagen de la interfaz de **portainer**, que permite gestionar los contenedores de docker de forma interactiva. Donde configuré los volúmenes para descargar **NGINX proxy manager**. Gestioné los proxys reversos, redirigiendo las peticiones http a https de mi dominio, generando un certificado **SSL**.

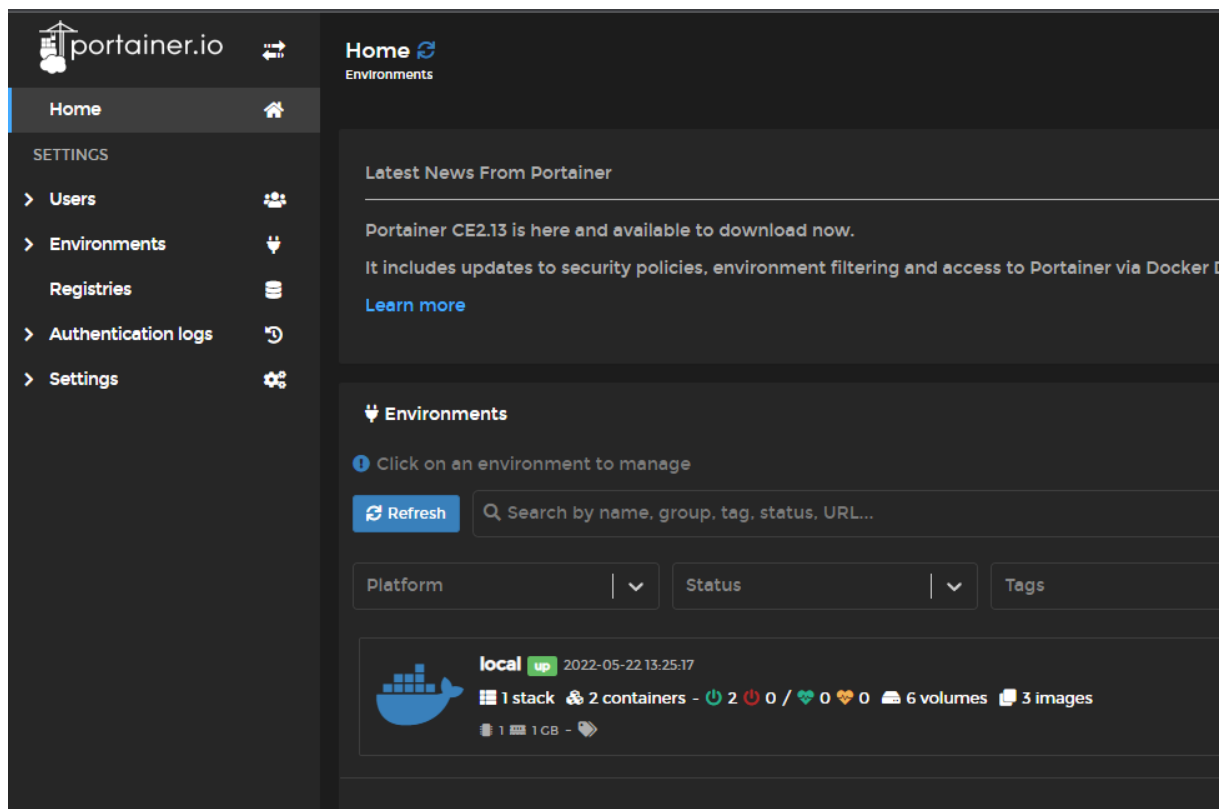


Figura 15: Portainer

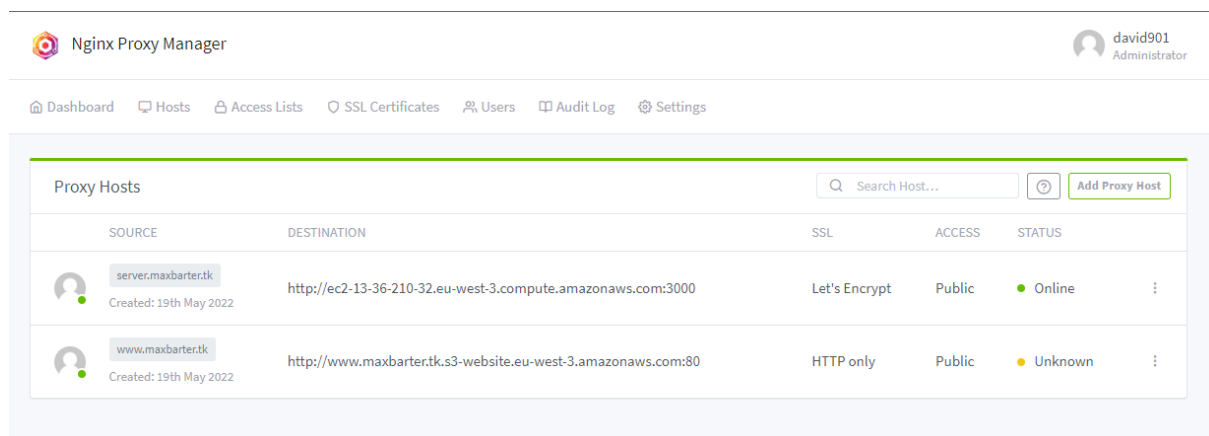


Figura 16: Nginx Proxy Manager

6. Recursos



6.1 Herramientas de hardware

Para desarrollar el proyecto he hecho uso de mi ordenador de sobremesa con buenas prestaciones. También he utilizado un portátil en aquellas ocasiones que no disponía del ordenador de sobremesa. Por lo general, no ha sido necesario tener unas buenas herramientas de hardware, pero sí que se agradece ya que ahorras más tiempo a la hora de compilar los cambios y ver los resultados durante el desarrollo.

6.2 Herramientas de software

He necesitado usar muchas herramientas de software durante el desarrollo del proyecto. Destacaré las más relevantes ya que también he usado otras herramientas para resolver pequeños problemas pero que no han tenido el mismo peso.

Software utilizado:

 IDE Visual Studio Code Git GitHub MySQL WorkBenche Xampp Figma Docker

6.3 Sistemas operativos empleados

Para el desarrollo he usado el sistema operativo de **Windows 10**, donde me he instalado las herramientas necesarias. Por otro lado, he usado la distribución de Linux **Ubuntu** para la máquina virtual del servicio EC2 de AWS, donde me he instalado los paquetes necesarios para desplegar el servidor.

7. Conclusiones

7.1 Grado de consecución de objetivos

Se ha alcanzado el objetivo perseguido, crear un sistema de trueques con el que los usuarios puedan satisfacer sus necesidades y dar vida a los objetos que han quedado

inutilizados. Consiguiendo crear las funcionalidades necesarias para una buena operatividad, con un destacable atractivo al público.

Facilitando el encuentro de posibles usuarios que mantengan un interés mutuo por algún objeto.

7.2 Problemas encontrados

El desarrollo del proyecto ha sido una tarea **compleja**, debido a que en gran parte de las tecnologías empleadas no disponía de ningún conocimiento previo. No obstante, me han generado muchos **problemas** que he tenido que ir resolviendo y aprendiendo. Asimilando conceptos que desconocía pero que me han sido muy útiles e interesantes de aprender.

Además del tiempo dedicado al estudio de las nuevas tecnologías y su implementación en la práctica.

Algunos de los problemas que me surgían eran cómo gestionar la imágenes, el sistema de autenticación, desplegar la aplicación etc. Los cuales he ido resolviendo gracias al esfuerzo dedicado.

7.3 Mejoras

Me he centrado en que la aplicación sea fácil de escalar para añadir nuevas funcionalidades a la aplicación. Debido al límite de tiempo para la entrega del

proyecto no he podido implementar todas aquellas funcionalidades que me hubieran gustado, ya que serían muchas y era inviable hacerlas todas.

Una de las mejoras que quiero implementar a futuro es crear un sistema de **chat en vivo** para la comunicación entre usuarios. Utilizando la tecnología de **socket.io** que usa el protocolo **Websocket** con lo que me permitiría una comunicación bidireccional a tiempo real entre el cliente y el servidor.

8. Explicación contenidos de Github

Como la aplicación está dividida en dos, tendremos un repositorio para Cliente y otro para el Servidor.

 Cliente: <https://github.com/DavidEsteve901/MaxBarter-Cliente.git>

 Servidor: <https://github.com/DavidEsteve901/MaxBarter-Servidor.git>

En ambos encontraremos un documento **pdf con la documentación del proyecto**, la explicación de sus contenidos está desarrollada en sus respectivos apartados en el punto de *Implantación*.

9. Anexos



Enlace a la aplicación desplegada: www.maxbarter.tk



Enlaces a los repositorios de github:

Cliente: <https://github.com/DavidEsteve901/MaxBarter-Cliente.git>

Servidor: <https://github.com/DavidEsteve901/MaxBarter-Servidor.git>



Enlace al diseño en figma:

<https://www.figma.com/file/ItxXfYSrNz9D294LAci0de/Proyecto-MaxBarter?node-id=0%3A1>



Enlace al prototipo interactivo:





















<https://www.figma.com/proto/ItxXfYSrNz9D294LAci0de/Proyecto-MaxBarter?page-id=0%3A1&node-id=25%3A884&starting-point-node-id=25%3A884>



Enlace al excel del diagrama de Gantt ***deja acceder si eres usuario de consellería, ya que lo hice con las herramientas de excel de microsoft y no deja ponerlo público***

https://gvaedu-my.sharepoint.com/:x:/g/personal/davestvic_alu_edu_gva_es/EZpnqs4T6upGno9Cmzdym0wBcTFJUS2NlsezHxkM4enc3A?e=YR4kGm

10. Bibliografía

-  Visual Studio Code url: <https://code.visualstudio.com/docs>
 -  Angular url: <https://angular.io/docs>
 -  Npm url: <https://www.npmjs.com/>
 -  Node.js url: <https://nodejs.org/es/>
 -  Express.js url: <https://expressjs.com/es/>
 -  Sequelize url: <https://sequelize.org/>
 -  Postman url: www.postman.com
 -  Bootstrap url: <https://getbootstrap.com/>
 -  PrimeNg url: <https://www.primefaces.org/primeng/>
 -  FontAwersome url: <https://fontawesome.com/>
 -  jQuery url: <https://jquery.com/>
 -  Docker url: <https://www.docker.com/>
 -  Nginix url: www.nginx.com
 -  Figma url: <https://www.figma.com/>
 -  MySQL Workbench url: <https://www.mysql.com/products/workbench/>
 -  Xampp url: <https://www.apachefriends.org/es/index.html>
 -  StackOverflow (Joel Spolsky y Jeff Atwood) url: <https://es.stackoverflow.com/>
- (La más importante sin duda)
-  Animate.css url: <https://animate.style/>
 -  Aos url: <https://michalsnik.github.io/aos/>
 -  Rellax url: <https://dixonandmoe.com/rellax/>

11. Índice de figuras

Figura 1: Estudio previo.....	13
Figura 2: Excel programación de tareas.....	17
Figura 3: Diseño pestañas.....	19
Figura 4: Diseño componentes.....	20
Figura 5: Prototipo interactivo.....	21
Figura 6: Flujo de navegación.....	22
Figura 7: Ficheros servidor.....	28
Figura 8: Postman.....	32
Figura 9: Ficheros Cliente.....	35
Figura 10: Servicio RDS de AWS.....	41
Figura 11: Servicio S3 de AWS.....	42
Figura 12: Servicio EC2 de AWS.....	42
Figura 13: Dominio de freenom.....	43
Figura 14: Servicio Route 53 de AWS.....	43
Figura 15: Portainer.....	44
Figura 16: Nginx Proxy Manager.....	45



Estudio previo : <https://www.helioesfera.com/estudio-previo-de-consumo-a-instalacion/>

Imagen Angular: https://es.m.wikipedia.org/wiki/Archivo:Angular_full_color_logo.svg

Express.js : <https://medium.com/@aarnlpezsosa/introducci%C3%B3n-a-express-js-a1ebe16dbcf4>

Imagen MySQL: <https://1000marcas.net/mysql-logo/>

Logo Html: https://commons.wikimedia.org/wiki/File:HTML5_logo_and_wordmark.svg

Logo Sass: <https://1000marcas.net/sass-logo/>

Logo TypeScript: https://es.m.wikipedia.org/wiki/Archivo:Typescript_logo_2020.svg

Logo AWS: https://es.m.wikipedia.org/wiki/Archivo:Amazon_Web_Services_Logo.svg



IES la Vereda

Desarrollo de Aplicaciones Web (DAW)

**PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO TÉCNICO
SUPERIOR DE DESARROLLO DE APLICACIONES WEB**

TEMA:

“Creación de una aplicación destinada al trueque/intercambio de productos.”

AUTOR:

David Esteve Vicente

TUTOR/A:

M^a Carmen Martinez Carbonell

2021/2022