



TRABAJO FIN DE GRADO SUPERIOR

Desarrollo de Aplicaciones Web 2021-2022

JOSÉ IGNACIO GUTIÉRREZ CERRATO

VESELIN BONTCHEV STANEV

AGRADECIMIENTOS

Nos gustaría agradecer este proyecto a todos los profesores del centro por enseñarnos y apoyarnos en estos dos años de aprendizaje. En los dos años académicos y sobre todo en el segundo, ha habido muchos momentos duros y de sacrificio. Pero sin ninguna duda nos ha merecido la pena vivir y pasar por ellos ya que hemos adquirido un aprendizaje que nos abre muchas oportunidades tanto profesionales, como personales.

A nuestros compañeros en general por estar siempre atentos y dispuestos a ayudar.

¡A todos muchas gracias!

INTRODUCCIÓN

1. Contexto

Este proyecto es un Trabajo Final de Grado (TFG) que se enmarca en los estudios de la FP de Grado Superior en Desarrollo de Aplicaciones Web del Instituto Tecnológico de Edix.

2. Objetivos

El proyecto que se ha desarrollado es una aplicación web y pretende cubrir las siguientes necesidades:

- Permitir al cliente la búsqueda rápida y fácil de oficinas de alquiler y/o coworking sin necesidad de registrarse.
- Suministrar al sector inmobiliario una herramienta de trabajo fácil de usar, pero a la vez potente y ágil en cuanto a funcionamiento. Además, la aplicación debe presentar una interfaz sencilla, y el uso de la misma ha de ser intuitivo.

3. Motivación

Hoy en día gran parte de los procesos del sector inmobiliario han dejado de realizarse de la forma tradicional, y en su lugar, han dado giro hacia la digitalización. Esta transformación ha ocurrido a un ritmo acelerado, y es solo cuestión de tiempo que todas las empresas del sector realicen sus gestiones de manera 100 % online.

A la hora de alquilar una oficina cada empresa tiene necesidades diferentes por lo que a la hora de elegir un espacio de trabajo se valoran diferentes factores como, ubicación, modalidad, espacio, precio, etc.

Los últimos años ha proliferado sobre todo el coworking. Son espacios perfectos para empresas, pero también para autónomos en general que busquen un espacio de trabajo dónde poder concentrarse mejor que en casa. A su vez la mayoría de coworkings generan un ambiente inspirador, creativo y además se concentran emprendedores, freelancers y todo tipo de profesionales con los que se pueden llegar a crear sinergias y alianzas que pueden llegar a ayudar al desarrollo del negocio o incluso crear nuevas oportunidades de negocio.

PALABRAS CLAVE

API-REST

Es un conjunto de reglas que definen cómo pueden las aplicaciones o los dispositivos conectarse y comunicarse entre sí y además permite la interacción con los servicios web de RESTfull.

JSON WEB TOKEN (JWT)

Información de verificación enviada al servidor y que esta compuesta por datos JSON. A través de un algoritmo hash se genera una cadena.

SPA (Single Page Application)

Tipo de aplicación web donde todas las pantallas las muestra en la misma página, sin recargar el navegador. Todo el contenido de la web se carga al completo una primera vez y mientras se navega, los recursos solicitados al servidor se cargan de manera asíncrona, sin necesidad de recargar la página entera.

Índice

INTRODUCCIÓN	1
PALABRAS CLAVE.....	2
RESUMEN.....	4
MÓDULOS FORMATIVOS APLICADOS.....	5
HERRAMIENTAS/ LENGUAJES UTILIZADOS	7
COMPONENTES DEL EQUIPO	11
FASES DEL PROYECTO	12
Fase I: Idea del Proyecto.....	12
Fase II: Usuarios y roles, definición de requisitos	12
Fase III: Creación del entorno NodeJS. Incorporación de Express.....	14
Fase IV: Diseño de la Base de Datos	16
Fase IV: Back-End. Creación de la API-REST.....	20
Fase V: Front-End de la aplicación	25
Fase VI: Documentar el proyecto.....	34
FUNCIONAMIENTO DE LA APLICACIÓN	37
CONCLUSIONES	48
MEJORAS DEL PROYECTO.....	49
BIBLIOGRAFÍA.....	50

RESUMEN

Este proyecto expone la creación de una aplicación web destinada al sector inmobiliario y cuya función principal es la búsqueda de oficinas de alquiler y/o coworking.

La idea surge dado que los dos alumnos trabajan actualmente en el sector, aunque dedicados a funciones que no tienen relación con el alquiler de inmuebles. Sin embargo, se intuye la posibilidad de facilitar la búsqueda de oficinas por empresas o autónomos y a la par crear una aplicación que se pueda usar por cualquier inmobiliaria dedicada al alquiler de espacios de trabajo. Aunque en el sector existen otras herramientas del estilo, esta ofrece como ventajas una interfaz sencilla y un buen rendimiento gracias a las tecnologías utilizadas y cuya explicación se hará en otro capítulo.

Asimismo, tal y como se estructura la aplicación, deja la posibilidad para una futura ampliación de nuevas funcionalidades según las necesidades de los usuarios.

En conclusión, la realización de este proyecto surge de la necesidad de crear una aplicación web intuitiva y sencilla que permita al cliente la búsqueda rápida y fácil de espacios de trabajo en territorio nacional, pero por otro lado también sirva de herramienta de trabajo a los agentes inmobiliarios.

MÓDULOS FORMATIVOS APLICADOS

Desarrollo Web en Entorno Servidor

Este módulo se ha utilizado principalmente para montar la parte Back-End de la aplicación, fundamentalmente para construir una API-REST.

Bases de Datos

Los conocimientos adquiridos en esta asignatura han ayudado principalmente entre otras cosas a integrar la información de la aplicación, en la persistencia de los datos y en los mecanismos para asegurar la integridad y la seguridad de los mismos.

Desarrollo Web en Entorno Cliente

Esta asignatura ha ayudado en la construcción de la parte Front-End y concretamente en el desarrollo de los siguientes aspectos:

- La integración de lenguaje de programación y el lenguaje de marcas.
- La incorporación de funcionalidades en los documentos web.
- La utilización de características y objetos propios del lenguaje y de los entornos de programación y ejecución.
- La utilización de mecanismos para la gestión de eventos y la interacción con el usuario.
- La incorporación de técnicas y librerías para la actualización dinámica del contenido.

Lenguajes de marcas

Este módulo se ha utilizado en la parte Front-End para crear la estructura de las diferentes páginas y codificar los diferentes documentos de texto.

La aplicación de los conocimientos de esta asignatura ha sido imprescindible para poder visualizar la aplicación web en los diferentes navegadores gracias a los lenguajes HTML5 y CSS3.

Diseño de Interfaces Web

Esta asignatura también se ha utilizado en la parte Front-End. Se han aplicado los conocimientos de los alumnos para diseñar una interfaz más efectiva, atractiva y usable.

Despliegue de Aplicaciones Web

Los conocimientos adquiridos en esta asignatura han ayudado a implantar la arquitectura web analizando y aplicando criterios de funcionalidad. Por otro lado también, evaluar y aplicar criterios de configuración para el acceso seguro a los servicios.

Entornos de desarrollo

El módulo de entornos de desarrollo se ha empleado sobre todo en la utilización de GIT.

Gracias a esta asignatura los alumnos han sido capaces de aplicar la gestión de control de versiones en los repositorios creados para el proyecto consolidando los conocimientos adquiridos.

Inglés

La asignatura de inglés se ha aplicado en la codificación del proyecto. Dado que todos los comandos y palabras claves en los lenguajes de programación se encuentran en inglés se ha decidido no hacer mezcla de dos idiomas y de esa manera crear un código más homogéneo lingüísticamente.

HERRAMIENTAS/ LENGUAJES UTILIZADOS

Para la construcción de la aplicación objeto del presente proyecto se han utilizado las tecnologías MEAN en su totalidad: MongoDB, Express, Angular y NodeJS.

JavaScript



Es el lenguaje principal de las tecnologías MEAN y en el que está basado el lenguaje TypeScript. Se ejecuta del lado del cliente y es uno de los lenguajes más demandados en los últimos años. Es un lenguaje de alto nivel, no tipado o de tipado débil e interpretado. Permite crear elementos dinámicos e interactivos, mejorando ampliamente la interacción de los usuarios con una página web.

TypeScript



Es un lenguaje orientado a objetos y construido a un nivel superior de JavaScript. Es de programación libre y de código abierto desarrollado y mantenido por Microsoft. Su principal característica es que permite definir el tipo de variables que se van a usar y además es casi obligatorio usarlo si se trabaja con un framework.

Otro aspecto a destacar es que se convierte a JavaScript y se ejecuta en cualquier lugar donde se ejecute JavaScript. También soporta ficheros de definición que contengan información sobre los tipos de librerías JavaScript existentes.

MongoDB



Es un sistema de base de datos NoSQL, orientado a documentos, es decir, en lugar de guardar los datos en registros, guarda los datos en documentos. Estos documentos son almacenados en BSON, que es una representación binaria de JSON.

Una de las diferencias más importantes con respecto a las bases de datos relacionales, es que no es necesario seguir un esquema.

Se ha elegido este modelo por su mejor implementación dadas las tecnologías utilizadas. Ofrece una gran escalabilidad y flexibilidad.

NodeJS



Se ha utilizado para la parte Back-End. Representa un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa de servidor basado en el lenguaje de programación

JavaScript.

La ventaja que tiene NodeJS es que esta diseñado a compartir un único hilo de ejecución entre todas las solicitudes y atiende a necesidades de aplicaciones altamente concurrentes. Gracias a esta característica no se producen bloqueos de procesos. Toda operación que realice entradas y salidas debe tener una función callback.

Express



Se ha utilizado en el Back-End y es el framework más popular de NodeJS. Está escrito en JavaScript y presenta un conjunto sólido de características para compilar aplicaciones de una sola página (SPA),

aunque no exclusivamente.

El framework Express hace que sea más fácil desarrollar una API-REST para manejar múltiples tipos de solicitudes, como GET, PUT, POST y DELETE.

Angular



Se ha utilizado principalmente para el desarrollo del Front-End. Es un framework para aplicaciones web muy potente y desarrollado en TypeScript, por ese motivo la utilización de este lenguaje. Mantenido por Google se utiliza para crear y mantener aplicaciones SPA (Single Page Application). Para conseguirlo Angular ofrece un sistema de routing entre sus herramientas.

Los componentes son las piezas básicas de Angular y cuyas propiedades son las usadas para el binding (enlace) de los datos. Un componente está compuesto por una plantilla html, un archivo de lógica con extensión .ts y un archivo para el CSS o para los estilos.

Bootstrap



Es un framework Front-End cuya principal ventaja es que permite la creación de sitios y apps 100% adaptables a cualquier tipo de dispositivo, es decir, totalmente responsive. Una cuestión que se ha considerado de suma importancia teniendo en cuenta que a día de hoy son cada vez más los usuarios que acceden a Internet a través de sus teléfonos y tablets.

HTML5 /CSS3



Lenguaje de marcado y lenguaje de estilos que casi siempre van de la mano y que se han utilizado para el desarrollo básico de los documentos HTML.

Visual Studio Code



Es un IDE y editor de código fuente. Se ha optado por él ya que es la elección principal para desarrolladores JavaScript y es ideal para las tecnologías utilizadas. Además, se ejecuta en Windows, macOS y Linux.

Entre muchas otras funciones incluye soporte para la depuración, control integrado de GIT, resaltado de sintaxis y la opción de instalar una gran cantidad de plugins que ayudan en la labor del desarrollador. Es gratuito, desarrollado por Microsoft y de código abierto.

Postman



Nace como una herramienta que principalmente permite **crear peticiones sobre APIs** de una forma muy sencilla, para realizar pruebas con las mismas.

Es una herramienta muy usada por desarrolladores para comprobar el funcionamiento de una API. Posee un conjunto de utilidades para la gestión y monitorización de las mismas, permitiéndote crear y enviar peticiones http, definir colecciones, gestionar documentación, tener entornos colaborativos, establecer variables locales y globales, crear servidores de mockups o sandbox, entre otras cosas. Nos ha ayudado a generar las peticiones y probarlas, lo cual nos ha facilitado la creación de la API.

GIT



Git es un sistema de control de versiones distribuido gratuito y de código abierto creado por Linus Torvalds. Se ha diseñado para manejar todo, desde proyectos pequeños hasta proyectos muy grandes, con rapidez y eficiencia.

Como características principales destacan la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.

Swagger



Es la tecnología más popular para las documentaciones de API-REST. Swagger fue desarrollado por Reverb, pero actualmente se caracteriza por su neutralidad como código abierto al abrigo de la Fundación Linux,

llamada Open API Initiative.

Compodoc



Es una herramienta que facilita la generación de documentación para proyectos Angular. Tiene muchas funcionalidades como realización de búsquedas, elección de temas y la generación de documentación en varios idiomas.

COMPONENTES DEL EQUIPO

José Ignacio Gutiérrez Cerrato

Back-End:

- API-REST
- Controlllers: oficinas, profile.
- Rutas: oficinas, profile.
- JWT
- Front-End:
- Componentes Angular: home, login, carrouselimagenes, oficina, oficina-detail, all-profiles, create-profile-secondary, oficinas-filtro, oficinas-modality, serviciosoficinas.
- Servicios.
- Documentación Back-End con Swagger

Vesselin Bontchev Stanev

Back-End:

- API-REST
- Instalación librería nodemailer.
- Controlllers: email, comercial.
- Rutas: email, comercial.

Front-End:

- Componentes Angular: employee, employee-detail, formulario, google-maps, oficina, oficina-detail.
- Footer.
- Models.
- Documentación Front-End con Compodoc.

FASES DEL PROYECTO

Fase I: Idea del Proyecto

Esta es la primera fase y el origen de todo proyecto. Se podría decir que es la parte más básica, dado que sin ella no sería posible construir nada en concreto.

Tras barajar otras ideas iniciales los alumnos se ponen de acuerdo, toman la decisión definitiva y pasan a la siguiente fase.

Fase II: Usuarios y roles, definición de requisitos

Factores considerados:

Para poder definir los usuarios y los roles se identifica primero el público objetivo del sector al que va enfocada la aplicación. Identificar el grupo de audiencia adecuado en gran medida decide el éxito o el fracaso de la aplicación.

1. Definición de usuarios y roles.

Se definen los tipos de usuarios y las acciones que pueden llevar a cabo en la aplicación web.

1.1 **Cliente:** cualquier persona que puede acceder al sitio web y pueda buscar oficinas sin registrarse. También podría solicitar información a través de un formulario de contacto.

1.2 **Comercial:** usuario registrado en la aplicación que se pone en contacto con los clientes interesados en alquilar un espacio de trabajo y les propone realizar una visita conjunta.

1.3 **Administrador:** usuario registrado que tiene todos los permisos: puede obtener un listado de todas las oficinas o comerciales, dar de alta, modificar, o borrar tanto comerciales, como oficinas. Podría ser un comercial o no.

2. Se definen los requisitos funcionales y no funcionales.

2.1 Requisitos funcionales.

Se refieren expresamente al funcionamiento de la aplicación. Se definen los siguientes:

- La aplicación debe permitir la búsqueda de oficina de alquiler o coworking por alguno de los siguientes campos: nombre, dirección, provincia, ciudad, código postal o por metros cuadrados.
- La aplicación debe tener la funcionalidad de proporcionar un formulario de contacto al cliente interesado.
- Iniciar sesión en el sistema por usuarios registrados (solamente administrador y/o comerciales).
- Dar de alta los comerciales en el sistema (solamente por el administrador).
- Eliminar comerciales del sistema (solamente por el administrador)
- Modificar datos de los comerciales en el sistema (solamente por el administrador).
- Dar de alta las oficinas en el sistema (administrador y comercial).
- Subir archivos de fotografías en el sistema (administrador y comercial).
- Eliminar oficinas del sistema (administrador y comercial).
- Modificar los datos de las oficinas (administrador y comercial).

2.2 Requisitos no funcionales.

Se refieren a las características y restricciones del sistema, es decir aquellos que hay que tener en cuenta para llevar a cabo el proyecto, cubriendo los objetivos.

Algunos de ellos son disponibilidad, confidencialidad, estabilidad, mantenibilidad, usabilidad y rendimiento.

1. **Disponibilidad:** se garantiza el acceso a la página, así como la información deseada en un momento dado.
2. **Confidencialidad:** La información está protegida de accesos no autorizada gracias al uso de tokens de autenticación.
3. **Estabilidad:** El sistema debe tener el menor número posible de fallos.
4. **Mantenibilidad:** La aplicación está construida de tal manera que se requiere el mínimo esfuerzo para solucionar posibles errores e incorporar nuevas funcionalidades.

5. **Usabilidad:** La aplicación está pensada para ser fácil de manejar en cualquier contexto.
6. **Rendimiento:** El sistema al ser una aplicación Angular SPA, permite tiempos de carga muy cortos.

Fase III: Creación del entorno NodeJS. Incorporación de Express

Las herramientas que se utilizan tal y como se menciona en el capítulo de Herramientas/Lenguajes son NodeJS + Express.

1. Creación del entorno:

Una vez que se descarga y se instala Node.js en el equipo se puede utilizar el comando “npm”.

A través del IDE Visual Studio Code se crea la carpeta *app* y una vez abierto el terminal del editor de código se ejecuta el comando *npm init --yes* que solicita datos del proyecto y crea el archivo *package.json* con dichos datos y las dependencias de la aplicación.

Teniendo acceso al archivo *package.json* se pueden empezar a instalar los módulos externos necesarios para el proyecto como Express.

2. Instalación y configuración de Express.

Se instala Express con el comando *npm i express --save*. Se añade el parámetro *--save* porque esta va a ser una dependencia que siempre se va necesitar en el proyecto.

Una vez instalado, se crea la carpeta *node_modules*. Conforme se van instalando más módulos se van agregando dentro de esa carpeta.

2.1 Archivo server.js

Dentro de la carpeta app se crea el archivo *server.js*. En este archivo se incorpora el uso de express, el puerto de escucha del servidor y todas las demás configuraciones como enrutamientos que usará la app, el tipo de formato de las peticiones, etc.

A continuación, se presentan los detalles de la incorporación a express al proyecto y de la configuración del puerto de escucha dentro del archivo *server.js*. No obstante, tal y como se ha mencionado antes, no son las únicas configuraciones que contiene este archivo, ya que según se van añadiendo más elementos se van añadiendo más configuraciones.

Express se incorpora con la sentencia “require” y después se crea una instancia de el en la constante “app” Para que el servidor se lance al iniciar el proyecto se utiliza el método “listen ()”.

```
const express = require("express");
const app = express ();

const PORT = process.env.PORT || 8082;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

Configuración Express

Para incorporar otros módulos externos se utiliza el método “use ()”.

```
var corsOptions = {
  origin: "http://localhost:8082"
};

app.use(cors());
app.use(express.json())

app.use(bodyParser.json({limit: '10mb', extended: true}))
app.use(bodyParser.urlencoded({limit: '10mb', extended: true}))
```

Incorporación de otros módulos

- Body-parser: Permite acceder a los datos recibidos en el mensaje desde “request.body”
- urlencoded: permite acceder a los datos enviados como pares clave/valor en el cuerpo.
- Cors: permite peticiones de dominio cruzado ignorando la política del mismo origen. Por ejemplo, en fase de desarrollo el servidor del cliente Angular (4200) y la aplicación Express (8082) están separados.
- json(): Permite acceder a los datos enviados como json.

Fase IV: Diseño de la Base de Datos

Una vez creado el entorno e incorporado el framework Express se pasa a diseñar y crear la base de datos.

Se ha decidido utilizar la base de datos NoSQL MongoDB por sus principales características de flexibilidad, escalabilidad y rapidez del manejo de datos. Otros aspectos que se han tenido en cuenta son la rapidez de la etapa de desarrollo y por otro lado la integración de este tipo de BBDD con las nuevas tecnologías de desarrollo.

En vez de utilizar bases y registros como las bases de datos relacionales, se utilizan documentos y colecciones.

Las colecciones contienen una serie de documentos, los cuales se traducen en una unión de multitud de objetos con valores asociados a diferentes claves. Cada documento puede ser totalmente diferente a otro, aunque pertenezcan a la misma colección.

Una de las ventajas que destacan es que los objetos que se guardan dentro de la base de datos creada con MongoDB están más alineados con los propios objetos con los que se va a trabajar dentro del código desarrollado. Es decir, no se debe realizar ningún tipo de transformación sobre la información al extraerla de la base de datos.

Los documentos de MongoDB no deben tener un esquema predefinido con anterioridad, aunque en muchas ocasiones es recomendable y es por ello que en este proyecto se ha tratado con librerías como **Mongoose**. Para poder

manipular los datos en la API es necesario incorporar este módulo externo que se instala desde el gestor de paquetes NPM, “*npm install mongoose --save*”. Una vez instalado, en la carpeta “app” se crea otra subcarpeta “config” y dentro se crea el archivo *db.config.js*. Dentro de ese archivo se especifica la url que permite a mongoose conectarse a la base de datos y el nombre de esta. Según el tipo de datos que se tiene, se procede a un breve análisis de cuáles son los elementos que aportan información para poder definir la estructura de datos más adecuada.

Dentro de la carpeta models que es una subcarpeta de app, se deciden crear los siguientes modelos o esquemas de colecciones de datos. Cada uno de ellos está asociado a una colección de MongoDB y define el formato de los objetos dentro de esa colección:

- **Oficinas**, se compone de un id, título, descripción, disponibilidad, destacada, nombre de la calle, código postal, modalidad, latitud, longitud, planta, aparcamiento público, aparcamiento privado, línea de metro, línea de autobús, nombre del comercial asignado, metros cuadrados.



```

EdixBackend > app > models > JS oficinas.model.js > <unknown>
1  module.exports = mongoose => {
2  var schema = mongoose.Schema(
3  {
4    title: String,
5    description: String,
6    metros: {type: Number, default: 0},
7    is_destacada: {type: Boolean, default: false},
8    road_name: String,
9    zip_code: String,
10   province: String,
11   city: String,
12   modality: String,
13   latitude: String,
14   longitude: String,
15   floor: Number,
16   parking_public: {type: Boolean, default: false},
17   parking_private: {type: Boolean, default: false},
18   underground: String,
19   train: String,
20   bus: String,
21   comercial: String,
22   extPath: String
23 },
24 { timestamps: true }
25 );
  
```

Esquema de la colección oficinas

- **Perfiles**, se compone de un id, nombre, imagen principal, imagePath (ruta del archivo de la imagen), extPath (tipo de extensión del archivo de la imagen).

```
EdixBackend > app > models > JS profile.model.js > profileSchema > extPath
1  const mongoose = require('mongoose');
2  const { integer, number } = require('sharp/lib/is');
3
4  const profileSchema = mongoose.Schema({
5    oficinaId: { type: String, required: true },
6    name: { type: String, required: true },
7    imagePrincipal: { type: String, required: true },
8    imagePath: { type: String, required: true },
9    extPath: { type: String, required: true },
10 });
11
12 module.exports = mongoose.model('Profiles', profileSchema);
13
```

Esquema de la colección perfiles

- **Comercial**, se compone de nombre, usuario, contraseña, e-mail y rol.

```
package.json JS oficinas.model.js JS comercial.model.js X
EdixBackend > app > models > JS comercial.model.js > <unknown>
1  module.exports = mongoose => {
2    var schema = mongoose.Schema(
3      {
4        nombre: String,
5        usuario: String,
6        pwd: String,
7        email: String,
8        role: String
9      },
10     { timestamps: true }
11   );
```

Esquema de la colección comercial

Nota*

La opción *timestamps* le indica a Mongoose que administre automáticamente la propiedad *createdAt*: fecha y hora de creación del documento y la propiedad *updatedAt*: fecha y hora cuando se actualiza.

- **Contacto e-mail**, se compone de nombre, mensaje, e-mail, teléfono, empresa, provincia, modalidad, puestos, comercial, oficina.

```
EdixBackend > app > models > JS emailcontact.model.js > <unknown>
1  module.exports = mongoose => {
2    var schema = mongoose.Schema(
3      {
4        nombre: String,
5        message: String,
6        email: String,
7        telefono: String,
8        empresa: String,
9        provincia: String,
10       modalidad: String,
11       puestos: String,
12       comercial: String,
13       oficina: String
14     },
15     { timestamps: true }
16   );
```

Esquema de la colección contacto e-mail

Una de las características interesantes a mencionar es que Mongoose permite llevar a cabo desde la propia API comprobaciones antes de que los datos sean guardados. Estas comprobaciones normalmente se hacen desde el cliente, pero si se da el caso y el usuario consigue eludir las funciones JavaScript que lleven a cabo esas comprobaciones, la base de datos queda expuesta a recibir datos de todo tipo que pueden ser incompatibles con la API incorporada. Estas comprobaciones se hacen a través de validadores y destacan los siguientes:

- Validador “*required*”, comprueba que la clave sea proporcionada en el tratamiento de los datos.
- Validadores “*min*” y “*max*”, los contienen los SchemaType de tipo Number y comprueban que la clave a tratar contenga un valor mínimo o máximo establecido
- Validadores “*minlength*” y “*maxlength*”, los contienen los SchemaType de tipo String y comprueban que la clave a tratar contenga un número de caracteres mínimo o máximo establecido.

Fase IV: Back-End. Creación de la API-REST

Una vez diseñada la base de datos para poder acceder, recuperar y manipular los datos almacenados de la aplicación a través de los diferentes métodos de solicitud HTTP (GET, POST, PUT y DELETE) se procede a crear la API-REST.

Los datos almacenados se envían y se reciben en formato JSON.

Antes de entrar en materia y explicar la API, es conveniente aclarar dos conceptos que tienen un papel importante en la aplicación y además están relacionados uno con otro en el presente proyecto, estos son el Middleware y la implementación de la autenticación JWT.

Middleware

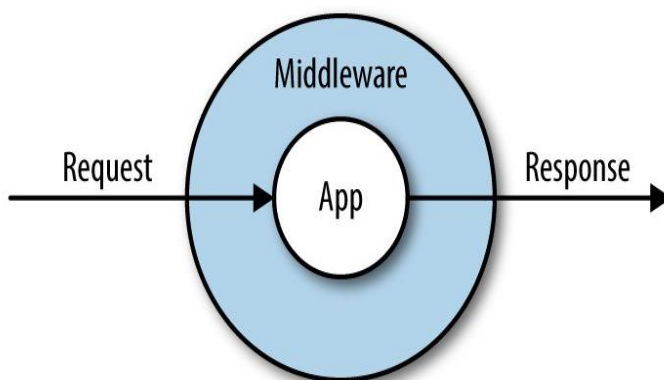
Un middleware es un bloque de código que se ejecuta entre la petición que hace el usuario (request) hasta que la petición llega al servidor.

Es una función que se puede ejecutar antes o después de una ruta. Esta función tiene acceso a los objetos Request, Response y la función next (). Antes de que una petición llegue a un controlador pasa por los middlewares que tenga en el camino.

Las funciones middleware suelen ser utilizadas como mecanismo para verificar niveles de acceso antes de entrar en una ruta, manejo de errores, validación de datos, etc.

Aunque los controladores son prácticamente middlewares y que serán explicados más adelante, se hace especial mención a la implementación de un middleware de autenticación que se crea en la carpeta “*helpers*” junto con otros archivos *js*.

El middleware comprueba que el token de usuario no haya expirado. Recibe en la cabecera de la ruta el token creado al iniciar la sesión del usuario.



Representación gráfica de un middleware

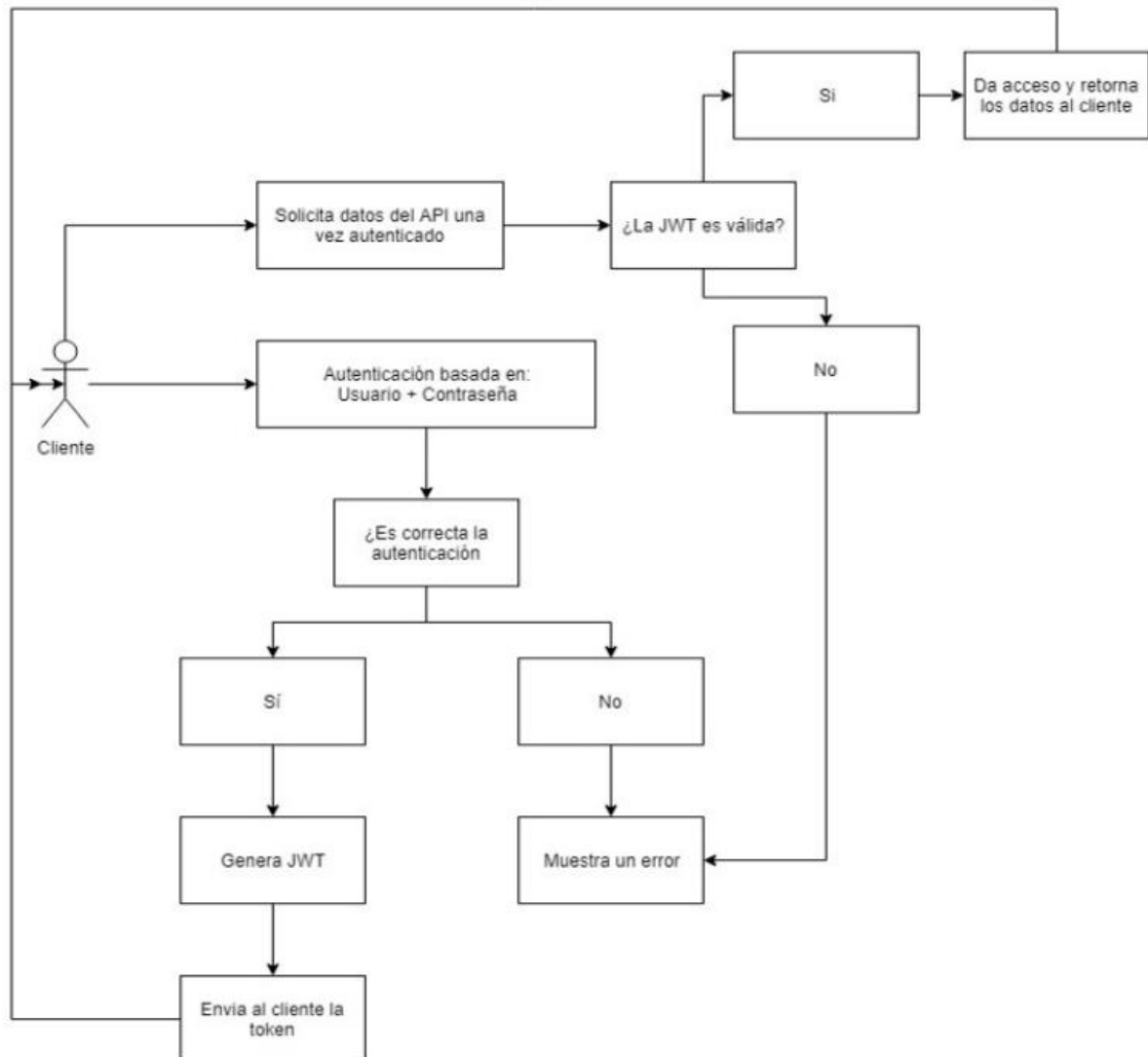
Autenticación JWT (Jason Web Tokens)

En la aplicación se implementa la autenticación JWT para restringir los endpoints a los usuarios autenticados.

JSON Web Token (JWT) es un estándar abierto (RFC-7519) basado en JSON para crear un token que sirva para enviar datos entre aplicaciones o servicios y garantizar que sean válidos y seguros.

El funcionamiento de JWT se basa en los siguientes pasos:

1. Autentica usando credenciales regulares (usuario-contraseña normalmente).
2. Una vez que autentica en el servidor, genera una cadena de caracteres que contiene el token de JWT integrado.
3. La ficha se envía al cliente.
4. La ficha se almacena al lado del cliente.
5. El token se manda al lado del servidor en cada petición que se realiza.
6. El servidor valida el token y otorga (o no) acceso al recurso que el cliente solicita.



Representación de funcionamiento de JWT

Controladores

Son los encargados de unir el modelo con la vista, y también poseen la lógica para transformar los datos para que los entienda tanto el modelo como la vista. En el proyecto se crea la carpeta “*controllers*” donde se crean cuatro archivos .js para los controladores:

- **comercial.controller.js**
- **emailcontact.controller.js**
- **oficina.controller.js**
- **profiles.controllers.js**

En cada uno de estos archivos se definen las funciones manejadoras que se ejecutan al llamar desde el cliente.

Definición de rutas o enrutamiento

El enrutamiento se refiere a cómo los endpoints de la aplicación responden a las peticiones HTTP generadas desde el cliente y que transferirán el flujo de control a los controladores. Cada ruta puede tener una o más funciones manejadoras, que se ejecutan cuando la ruta coincide.

Para crear manejadores de rutas se utiliza la clase “*express. Router*”

En el presente proyecto se crea la carpeta “*routes*” donde se crean los archivos *js* correspondientes a cada uno de los modelos a tratar.

- **comercial.routes.js**
- **emailcontact.routes.js**
- **oficinas.routes.js**
- **profile.routes.js**

La definición de una ruta tiene la siguiente estructura: `router.METHOD(PATH, HANDLER)`

Donde:

- router es la instancia de “*express. Router()*”
- METHOD es el método de solicitud HTTP (GET, POST, PUT o DELETE)
- PATH es la ruta de acceso.
- HANDLER es la función que se ejecuta al llamar a la ruta desde el cliente.

Ejemplo de la definición del enrutamiento para el login de un comercial:

```
router.post (“/login”, comerciales.login);
```

En el presente proyecto las funciones manejadoras que se ejecutan al llamar desde el cliente se definen en los controladores.

Carpeta helpers

Dentro de esta carpeta se crean los archivos, tal y como indica el nombre que sirven como ayudantes de apoyo para llevar a cabo las tareas de diferentes módulos. Los archivos creados son los siguientes:

-email.js: en este archivo se establece la lógica del envío del e-mail por parte del cliente. Se utiliza la librería nodemailer.

- middlewares.js: en este archivo se implementa la lógica del middleware de autenticación.

-storage.js: define las tareas utilizando la biblioteca multer que proporciona el disk storage motor para cargar imágenes en disco.

Pruebas con Postman

Como último paso del proceso de la creación de la API se realizan las pruebas de las peticiones con el programa Postman.

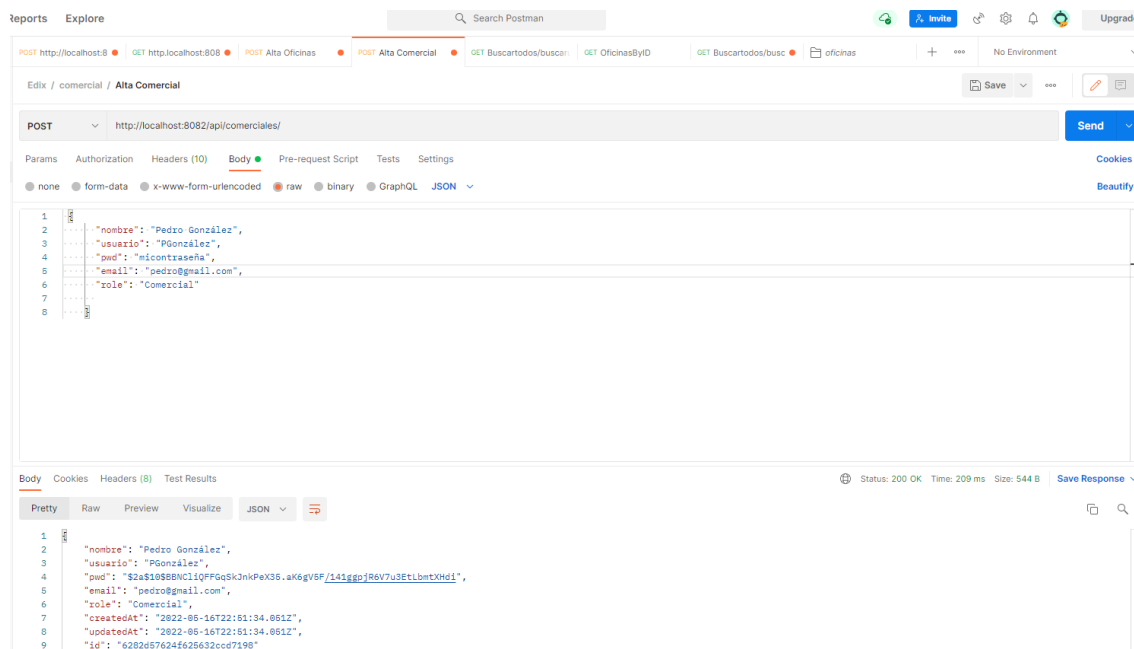
Gracias a esta herramienta **se pueden probar las peticiones sobre la API** de una forma muy sencilla. Es un cliente HTTP que ofrece la posibilidad de testear “HTTP request” a través de una interfaz gráfica de usuario, por medio de la cual se obtienen diferentes tipos de respuesta que posteriormente deberán ser validadas.

Ejemplo de prueba para dar de alta un comercial.

Para dar de alta un comercial se siguen los siguientes pasos:

1. Se selecciona el método POST
2. En el body se introducen los datos del comercial en formato JSON (datos de entrada).

3. Se introduce la url definida.
4. Se envía la petición.
5. Si todo es correcto el servidor devuelve el estado 200 OK y el formato JSON del nuevo documento creado (datos de salida).



Ejemplo prueba alta comercial con Postman

Se realizan las pruebas de los cuatro métodos HTTP (GET, POST, PUT Y DELETE) y se comprueba que funciona todo correctamente.

Una vez realizadas las pruebas de la API-REST se puede decir que concluye la parte del Back-End.

Fase V: Front-End de la aplicación

En esta fase se construye la capa en la que el usuario puede interactuar con la aplicación a través de las vistas y las diferentes funcionalidades creadas.

Esta capa se ha realizado con el framework Angular, por lo cual juega un papel importante el lenguaje de programación TypeScript cuyas características se mencionaron en el capítulo de Herramientas/Lenguajes utilizados.

Elementos sobre los que se basa la construcción de la aplicación.

Antes de proceder a la descripción del desarrollo de la parte front se hará una breve explicación de algunos elementos del propio framework Angular y sobre los que se basa la creación de la aplicación:

1. **Componente:** es un elemento que está compuesto por los siguientes archivos:
 - 1.1 archivo html que es la plantilla que se visualiza en el interfaz de usuario.
 - 1.2 archivo de lógica .ts que contiene una clase con las propiedades que se van a usar en la vista HTML y los métodos que son las acciones que se ejecutarán en la vista.
 - 1.3 Un archivo CSS donde se incluyen los estilos que se aplicarán en la vista HTML.
2. **Servicio:** básicamente es un proveedor de datos, que mantiene la lógica de acceso a ellos y la operativa relacionada con el negocio y las cosas que se hacen con los datos dentro de la aplicación. Los servicios se consumen por los componentes, que delegan en ellos la responsabilidad de acceder a la información y la realización de operaciones con los datos.
3. **Navegación o comunicación entre componentes:** es el mecanismo que permite a la aplicación navegar entre diferentes componentes enviando datos a la misma si los necesita.

4. Angular-CLI

Para generar la estructura inicial de la aplicación se utiliza la herramienta oficial del framework Angular-CLI. Se instala usando el gestor de paquetes npm (npm install -g @angular/cli). Gracias a los comandos que ofrece se pueden realizar las siguientes acciones:

- Generar el proyecto inicial.
- Generar partes posteriormente.
- Ejecutar el proyecto en modo desarrollo con compilado automático de TypeScript y actualización automática en el navegador.
- Construir el proyecto para distribución.

5. NgModule

Las aplicaciones Angular son modulares y Angular tiene su propio sistema de modularidad llamado **NgModules**.

Los NgModules son contenedores para un bloque cohesivo de código dedicado a un dominio de aplicación, un flujo de trabajo o un conjunto de capacidades estrechamente relacionado. Pueden contener componentes, proveedores de servicios y otros archivos de código cuyo alcance está definido por el NgModule contenedor. Pueden importar la funcionalidad que se exporta desde otros NgModules y exportar la funcionalidad seleccionada para que la usen otros NgModules.

Cada aplicación Angular tiene al menos una clase NgModule, el módulo raíz, que tiene un nombre convencional AppModule y reside en un archivo llamado app.module.ts. Inicia su aplicación arrancando el *NgModule* raíz.

La etiqueta @NgModule identifica al archivo “AppModule” como un módulo de Angular y contiene la información para que Angular pueda compilar y ejecutar la aplicación correctamente. La información se divide en tres partes:

- **declarations:** propiedad que incluye los componentes que forman parte de la aplicación.
- **imports:** en esta propiedad se incluyen los módulos que se van a utilizar en la aplicación.
- **providers:** propiedad que incluye los servicios de la aplicación.

Toda aplicación Angular tiene un módulo raíz que es el que lanza la aplicación y suele ser llamado “App Module”, pero puede contener más módulos si el proyecto lo requiere.

Descripción de las distintas partes que conforman el Front-End de la aplicación

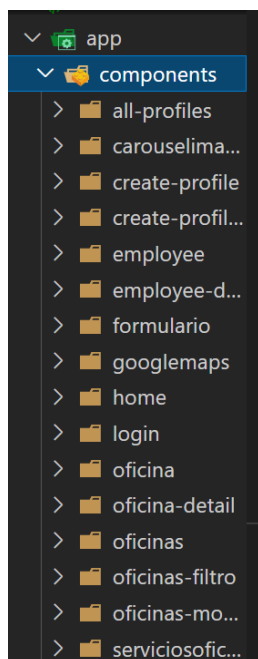
Componentes

Con la creación del proyecto con la ayuda de Angular- CLI se crea el módulo raíz que reside en el archivo `app.module.ts` y el componente `App` que se crea por defecto y que contiene 5 archivos:

- **app.component.html**: archivo para el HTML.
- **app.component.ts**: archivo para la lógica del componente.
- **app.component.css**: archivo para los estilos que se van a aplicar en el HTML.
- **app.component.spec.ts**: archivo para las pruebas unitarias.

El resto de componentes que se necesitan se van creando después (`ng generate component <name> [options]`) en la carpeta “components”.

Los componentes que componen la aplicación son los siguientes:

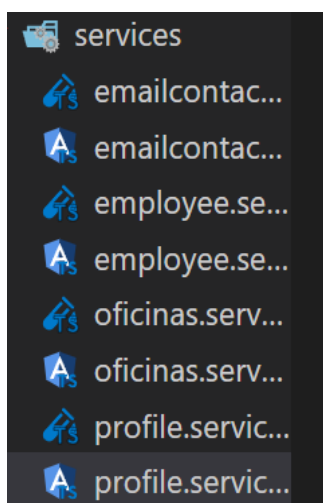


- **all-profiles**: componente para visualizar las imágenes cargadas de cualquier perfil (imagen principal y/o secundaria).
- **carouselimagenes**: componente que maneja el carrusel de las imágenes subidas.
- **create-profile**: componente que maneja la creación del perfil y la selección de la imagen principal.
- **create-profile-secondary**: componente para crear el perfil de imagen secundaria.
- **employee**: este componente se utiliza para manejar los diferentes métodos asociados al empleado/comercial.
- **employee-detail**: se utiliza para visualizar los detalles de los comerciales y manejar los diferentes métodos asociados.

- **formulario:** componente para el formulario de contacto.
- **google-maps:** componente para la visualización de la ubicación de las oficinas sobre el mapa de google-maps.
- **home:** este componente es para la pantalla principal de la aplicación.
- **login:** componente para el inicio de sesión de los usuarios registrados.
- **oficina:** una vez que un usuario registrado entra en la opción de “mantenimiento oficinas” se le redirige a este componente.
- **oficina-detail:** una vez que un usuario registrado está dentro de la opción “mantenimiento oficinas” y quiere añadir una oficina cuyos detalles tiene que rellenar se le redirige a este componente.
- **oficinas:** una vez que un cliente selecciona una oficina de las que salen en la pantalla principal se le redirige a este componente.
- **oficinas-filtro:** componente para filtrar la búsqueda de oficinas.
- **oficinas-modality:** una vez que un cliente selecciona una modalidad de oficina “Alquiler” o “Coworking” desde la pantalla principal se le redirige a este componente.
- **serviciosoficinas:** una vez que un cliente elige la opción “Servicios para Oficinas” desde el menú principal se le redirige a este componente.

Servicios

A la par que se crean los componentes se van creando los servicios para que puedan servir los datos a los componentes que los necesitan.



De nuevo se recurre a Angular CLI utilizando el siguiente comando: `ng generate service <nombre del servicio>`. Se crean 2 archivos por cada servicio, uno para las pruebas unitarias y otro para la lógica de las llamadas.

Para la aplicación se han creado los siguientes servicios en la carpeta “services”:

- **emailcontact:** servicio encargado de la llamada relacionada con el envío del e-mail.

- **employee:** servicio encargado de las llamadas relacionadas con empleado.
- **oficinas:** servicio encargado de las llamadas relacionadas con oficinas.
- **profile:** servicio encargado de las llamadas relacionadas con los perfiles de imágenes.

Cada vez que se crea un componente es necesario importarlo en el archivo `app.module.ts`, se añade debajo de `@NgModule` dentro de la propiedad **declarations**. Los módulos se importan en `app.module.ts` y se añaden en **imports**.

```
EdixFrontend13 > src > app > app.module.ts > AppModule
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3 import { FormsModule, ReactiveFormsModule } from '@angular/forms';
4 import { HttpClientModule } from '@angular/common/http';
5
6 import { AppRoutingModule } from './app-routing.module';
7 import { AppComponent } from './app.component';
8 import { HomeComponent } from './components/home/home.component';
9 import { EmployeeComponent } from './components/employee/employee.component';
10 import { LoginComponent } from './components/login/login.component';
11 import { OficinasComponent } from './components/oficinas/oficinas.component';
12 import { OficinaComponent } from './components/oficina/oficina.component';
13 import { OficinasModalityComponent } from './components/oficinas-modality/oficinas-modality.component';
14 import { OficinaDetailComponent } from './components/oficina-detail/oficina-detail.component';
15 import { EmployeeDetailComponent } from './components/employee-detail/employee-detail.component';
16 import { OficinasFiltroComponent } from './components/oficinas-filtro/oficinas-filtro.component';
17 import { AllProfilesComponent } from './components/all-profiles/all-profiles.component';
18 import { CreateProfileComponent } from './components/create-profile/create-profile.component';
19 import { CreateProfileSecondaryComponent } from './components/create-profile-secondary/create-profile-secondary.component';
20 import { GoogleMapsModule } from '@angular/google-maps';
21 import { ServiciosOficinasComponent } from './components/serviciosoficinas/serviciosoficinas.component';
22 import { CarouselImagenesComponent } from './components/carouselimagenes/carouselimagenes.component';
23 import { NgbModule } from '@ng-bootstrap/ng-bootstrap';
24 import { FormularioComponent } from './components/formulario/formulario.component';
25 import { AgmCoreModule } from '@agm/core';
26 import { GooglemapsComponent } from './components/googlemaps/googlemaps.component';
```

Importación de los módulos y de los componentes

```
app.module.ts ×
EdixFrontend13 > src > app > app.module.ts > AppModule
28
29
30 @NgModule({
31   declarations: [
32     AppComponent,
33     HomeComponent,
34     EmployeeComponent,
35     LoginComponent,
36     OficinasComponent,
37     OficinaComponent,
38     OficinasModalityComponent,
39     OficinaDetailComponent,
40     EmployeeDetailComponent,
41     OficinasFiltroComponent,
42     AllProfilesComponent,
43     CreateProfileComponent,
44     CreateProfileSecondaryComponent,
45     ServiciosOficinasComponent,
46     CarouselImagenesComponent,
47     FormularioComponent,
48     GooglemapsComponent
49   ],
50   imports: [
51     BrowserModule,
52     AppRoutingModule,
53     FormsModule,
54     ReactiveFormsModule,
55     HttpClientModule,
56     GoogleMapsModule,
57     NgbModule,
58     AgmCoreModule.forRoot({
59       apiKey: 'AIzaSyB1d_zDf-iMYKvBjM1fYt2N1z7nIzeRWI'
60     })
61   ],
62 })
```

Propiedades declarations e imports de @NgModule

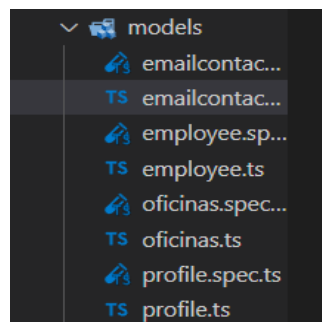
Clases e interfaces

Como en la aplicación hay que trabajar con las entidades en la BD se han definido tres clases y una interfaz que especifican los datos que contienen esas estructuras.

Por ejemplo, como en la aplicación se va a usar contacto e-mail, comercial, perfil y oficina es conveniente definirlos. Una vez definidas las clases se podrán instanciar y crear objetos en caso de necesitarlos.

Dentro de la carpeta “models” se crean las clases: emailcontact, employee y oficinas.

La interfaz profile se crea en la misma carpeta. La razón por la cual se crea esa interfaz es porque solamente se necesita verificar el tipo para esa estructura, por lo cual no se necesita de ningún método. Sin embargo, las clases necesitan del método toString () para pasar el formato JSON a String.



Definición de rutas

Las rutas le indican al enrutador qué vista mostrar cuando un usuario hace clic en un enlace o pega una URL en la barra de direcciones del navegador. Se definen en el archivo app-routing.module.ts.

AppRoutingModule importa RouterModule y Routes para que la aplicación pueda tener funcionalidad de enrutamiento.

Las rutas tienen dos propiedades:

- **path:** una cadena que coincide con la URL en la barra de direcciones del navegador.
- **component:** el componente que el enrutador debe crear al navegar a esta ruta.

Los metadatos @NgModule inicializan el enrutador y le hacen escuchar los cambios de ubicación del navegador.

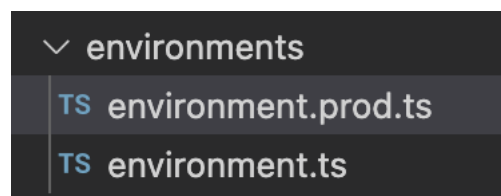
Ejemplo de la ruta definida para el login de la aplicación:

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { LoginComponent } from '../components/login/login.component';

const routes: Routes = [
  {
    path: 'login',
    component: LoginComponent
  }
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Variables de entorno

Las variables de entorno sirven para saber el entorno en el que estamos (producción o desarrollo) y también para acceder a otros datos que cambian dependiendo del entorno como se puede observar en la imagen adjunta, en función de cada entorno podemos tener unos valores distintos para esas variables.



Entornos

```
export const environment = {
  production: true,
  token: "Barear
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vyljpw7InVzdWFyaW8iOiJkdWVhY3RpdmUiLCJwd2QiOiJBMjM0NWNicmUifSwiaWF0IjoxNjQ5MzI3NjIzQgP9wJ9hJV8UPw9DwZ1Fjedp1jHAS96-dmE9BzfnUusg",
  pathRoot: "http://localhost:8082/api"
};
```

Producción

```
export const environment = {
  production: false,
  token: "Barear
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vyljpw7InVzdWFyaW8iOiJkdWVhY3RpdmUiLCJwd2QiOiJBMjM0NWNicmUifSwiaWF0IjoxNjQ5MzI3NjIzQgP9wJ9hJV8UPw9DwZ1Fjedp1jHAS96-dmE9BzfnUusg",
  pathRoot: "http://localhost:8082/api"
};
```

Desarrollo

En ese caso, la variable token es el token que usaremos para validar las peticiones realizadas por los clientes.

Navegación entre componentes

Como último paso de la presentación de la estructura del front-end, se muestra un ejemplo de navegación entre componentes:

```
<div class="form-group col-md-10 ">
  <a [routerLink]="[ '/employee' ]" id="btnEmployee" class="btn btn-secondary">Comerciales
</a>
  <a [routerLink]="[ '/oficina' ]" id="btnOficina" class="btn btn-secondary btn-warning">Oficinas
</a>
```

Ejemplo de navegación entre componentes

Para navegar entre los distintos componentes se utiliza la directiva “routerLink =ruta” dentro de los archivos html de los distintos componentes. Como se muestra en la imagen, si el usuario hace clic sobre el botón “Comerciales” irá al componente definido para la ruta “/employee” y si hace clic sobre el botón “Oficinas” irá al componente definido para la ruta “/oficina” y visualizará las oficinas.

Fase VI: Documentar el proyecto

En esta fase los alumnos realizan la documentación del código de la aplicación.

Para este fin se han utilizado 2 herramientas: Swagger para el Back-End y Compodoc para el Front-End.

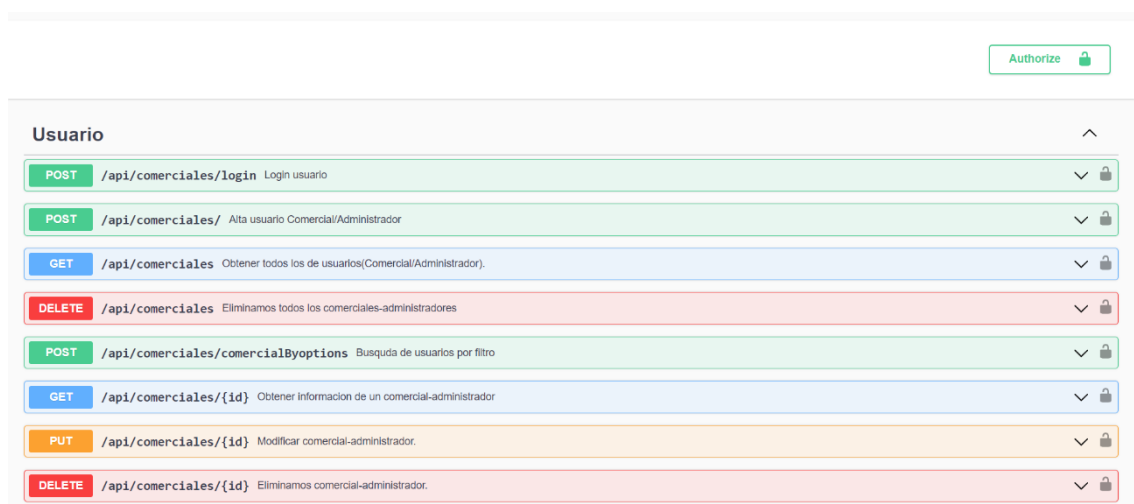
- **Documentación back-end con Swagger:**

Swagger es una especificación abierta para definir APIs-REST. Tiene incorporadas una serie de reglas, especificaciones y herramientas que nos ayudan a documentar las APIs.

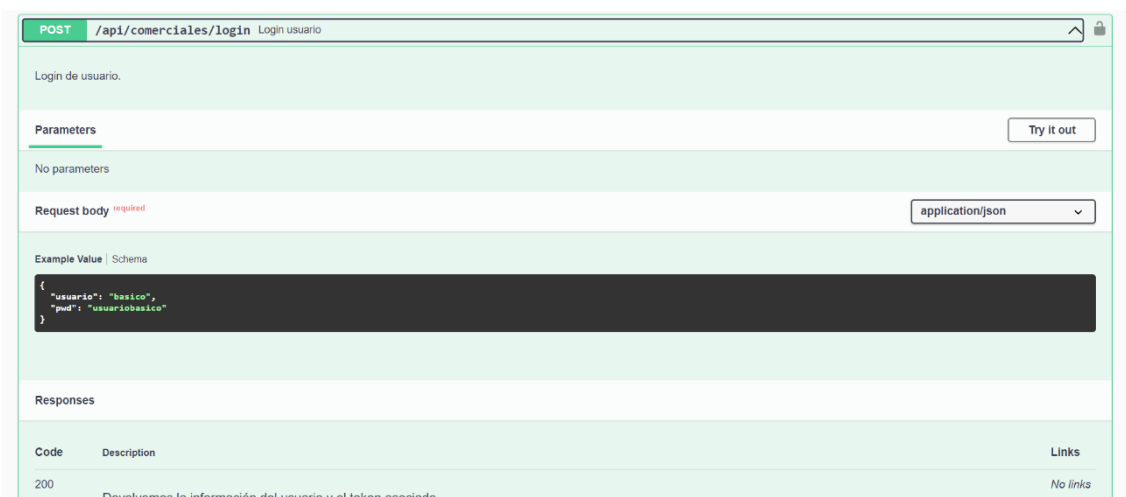
Cabe destacar que utilizando la UI de Swagger ha ayudado a ahorrar mucho tiempo en documentar el proyecto, dado que ofrece la posibilidad de organizar los métodos e incluso poner ejemplos.

Swagger UI utiliza un documento JSON y lo hace interactivo. Crea una plataforma que ordena cada uno de los métodos (get, put, post, delete) y categoriza las operaciones. Cada uno de los métodos es expandible, y en ellos se puede encontrar un listado completo de los parámetros con sus respectivos ejemplos. Incluso se pueden probar valores de llamada.

Se puede ver desde el back-end: <https://localhost/8082/api-docs/#/>



UI de Swagger con la organización de los métodos



Ejemplo de la documentación del método “POST” del login usuario

- **Documentación front-end con Compodoc:**

Es la herramienta de código abierto que ha ayudado en la generación de la documentación de Angular. La herramienta ofrece un diseño de documentación limpio y receptivo que incluye funcionalidad de búsqueda y varios temas a elegir. También permite la generación de la documentación en varios idiomas.

Su instalación es relativamente simple ya que basta en agregar el paquete @compodoc/compodoc usando el administrador de paquetes npm.


Después de instalar el paquete se realizan las configuraciones necesarias en el archivo package.json (se indica el puerto, tema elegido, etc.).

Una vez hechas las configuraciones, se ejecuta con el comando “*npm run compodoc*” y la documentación se genera automáticamente en la carpeta “documentation” y a la vez se abre como página web en <https://localhost:9000>

LÉAME

- dependencias
- Propiedades
- MÓDULOS
- CLASES
- INYECTABLES
- INTERFACES
- MISCELÁNEAS
- RUTAS
- COBERTURA DE DOCUMENTACIÓN

Documentación generada usando



Interfaz de Edix13

Este proyecto fue generado con [Angular CLI](#) versión 13.3.3.

servidor de desarrollo

Ejecutar `ng serve` para un servidor dev. Navegar a `http://localhost:4200/`. La aplicación se recargará automáticamente si cambia cualquiera de los archivos de origen.

Andamio de código

Ejecutar `ng generate component component-name` para generar un nuevo componente. También puedes usar `ng generate directive|pipe|service|class|guard|interface|enum|module`.

Construir

Ejecutar `ng build` para compilar el proyecto. Los artefactos de compilación se almacenarán en el `dist/` directorio.

Ejecución de pruebas unitarias

Ejecutar `ng test` para ejecutar las pruebas unitarias a través de [Karma](#).

Ejecución de pruebas de extremo a extremo

Ejecutar `ng e2e` para ejecutar las pruebas de extremo a extremo a través de una plataforma de su elección. Para usar este comando, primero debe agregar un paquete que implemente capacidades de prueba de un extremo a otro.

Pantalla de inicio al ejecutar compodoc

OficinasComponente

- OficinasFiltroComponente
- OficinasModalidadComponente
- ServiciosoficinasComponente
- Módulo de enrutamiento de aplicacion...
- CLASES
- INYECTABLES
- INTERFACES
- MISCELÁNEAS
- RUTAS
- COBERTURA DE DOCUMENTACIÓN

Documentación generada usando



Componentes / OficinasComponente

- Información
- Fuente
- Modelo
- Estilos
- Árbol DOM

Archivo

`src/app/components/oficinas/oficinas.component.ts`

Descripción

OficinasComponente

Implementos

`OnInit`

Metadatos

proveedores	<code>OficinasService</code>
selector	<code>app-oficinas</code>
URL de estilo	<code>./oficinas.component.css</code>

Ejemplo de la documentación del componente “Oficinas”

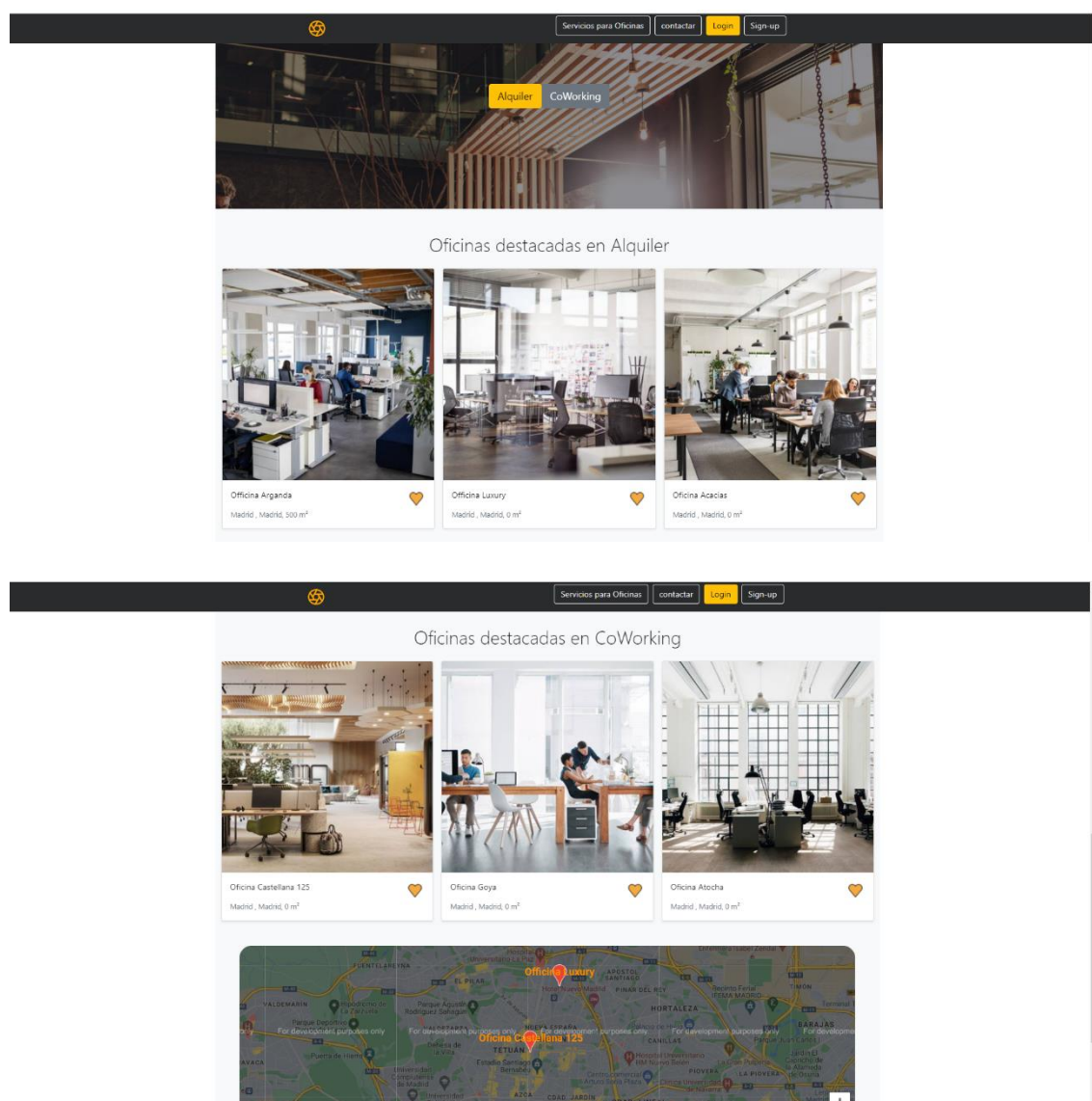
FUNCIONAMIENTO DE LA APLICACIÓN

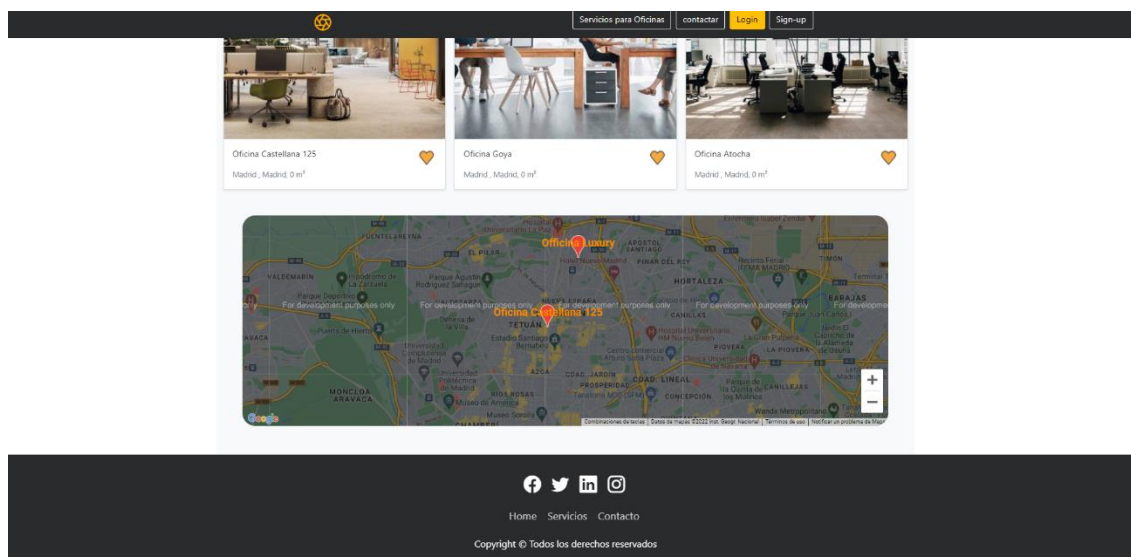
En esta sección de la memoria se muestra el aspecto visual de las diferentes partes, así como las posibles funciones a realizar por los usuarios.

Pantalla principal o de inicio

A esta pantalla tienen acceso todos los usuarios (cliente, comercial y administrador).

Se muestra el menú de navegación y las oficinas destacadas de las dos modalidades y el mapa de google-maps con la ubicación de las mismas.





Pantalla opción Alquiler

Si el usuario selecciona la opción de “Alquiler” desde la pantalla principal se mostrarán todas las oficinas destacadas de alquiler y aparte tendrá la opción de realizar una búsqueda de oficinas de esta modalidad por los campos que se muestran en la pantalla.



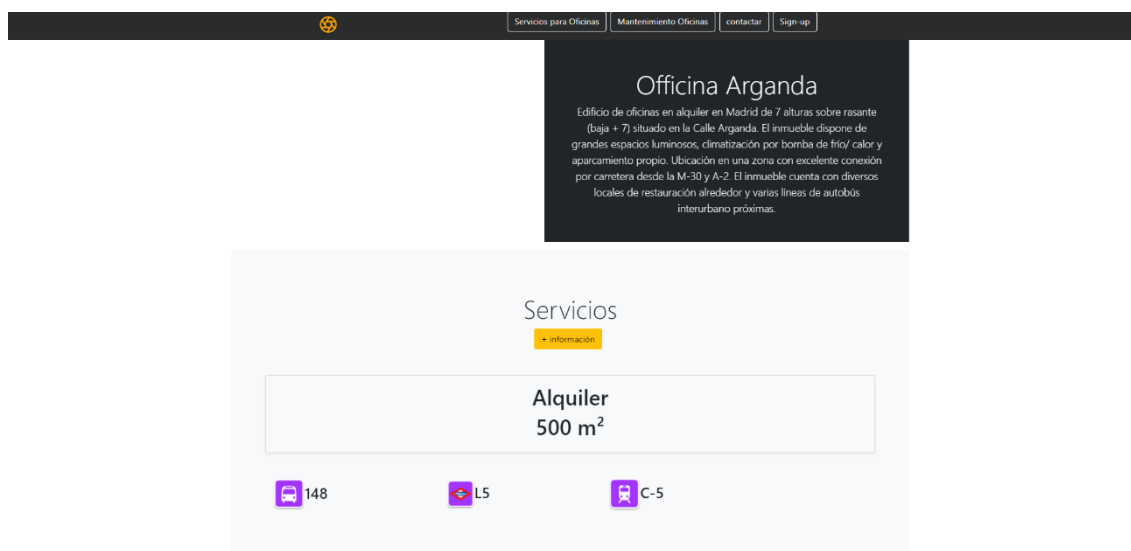
Pantalla opción Coworking

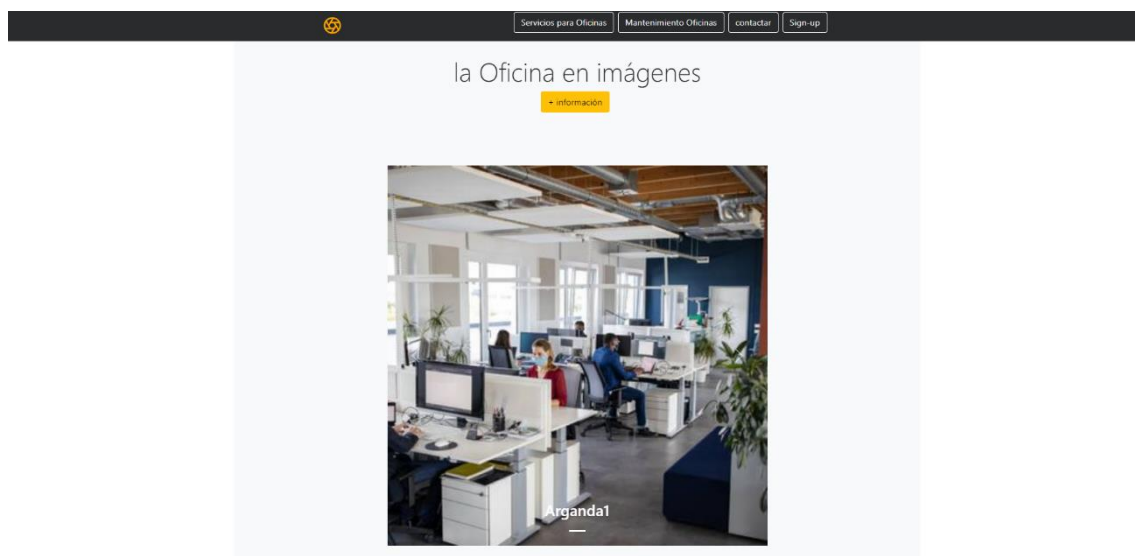
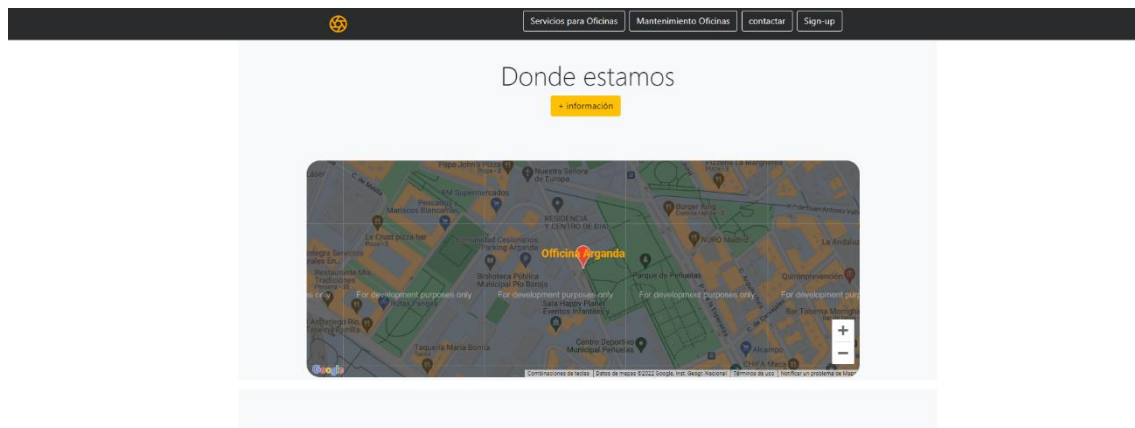
Lo mismo ocurre con la opción “Coworking, se muestran las oficinas destacadas de coworking y el buscador de esta modalidad de oficinas.



Pantalla detalles oficina

Si el usuario selecciona alguna de las oficinas destacadas, se muestra la vista con los detalles de la oficina seleccionada: descripción, servicios, imágenes y ubicación.





Haciendo clic sobre el botón “+ información” se despliega un formulario para solicitar información sobre esa oficina. Una vez rellenado ese formulario se envía al correo del comercial asignado que se pondrá en contacto con el cliente.

Formulario de contacto

Solicitud de información Déjanos tus datos y te contestaremos lo antes posible. Muchas gracias.

Nombre*
Introduce tu nombre

Email*
Introduce tu email

Teléfono
Introduce tu teléfono

Empresa
Introduce el nombre de tu empresa

Modalidad*
Alquiler

Cantidad de puestos
Cantidad de puestos

Mensaje*
Escribe tu mensaje

☐ He leído y acepto el [Aviso Legal](#) y la [Política de Privacidad](#)

¿Preferes llamarnos? [911 01 46 56](tel:911014656)

Enviar

Oficinas&Coworking, S.A. le informa que los datos de carácter personal proporcionados quedarán incorporados en sus bases de datos, y con

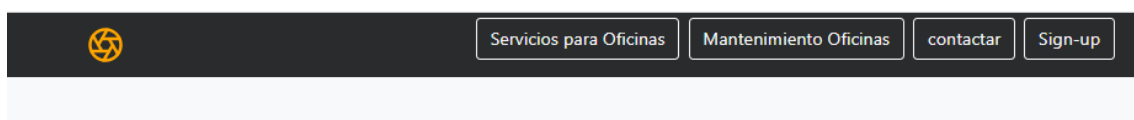
Botones del menú de navegación

El menú de navegación dispone de 5 botones. No obstante, dependiendo del rol de usuario se muestran unos u otros:

Usuarios con rol cliente: visualizan los botones “Inicio”, “Servicios para oficinas”, “contactar”, “Login” y “Sign-up”.



Usuarios con rol administrador o comercial: visualizan los mismos, pero una vez que inicien sesión con su nombre de usuario y contraseña visualizan uno más: “Mantenimiento oficinas”.



El botón “inicio” tiene forma circular, está posicionado a la izquierda del todo y es como el logo de la aplicación, traslada al usuario a la pantalla de inicio.

El botón de” contacto” tiene la función de mostrar un formulario de contacto genérico que no irá destinado sobre una oficina en concreto como en la pantalla “detalles oficina”, por lo cual se envía a un buzón genérico.

The screenshot shows a modal form titled "Solicitud de información" (Information Request) overlaid on the website. The form includes a close button (X) and a message: "Déjanos tus datos y te contestaremos lo antes posible. Muchas gracias." (Leave us your data and we will answer you as soon as possible. Thank you very much.). The form fields are:

- Nombre*** (Name): Introduce tu nombre
- Email*** (Email): Introduce tu email
- Teléfono** (Phone): Introduce tu teléfono
- Empresa** (Company): Introduce el nombre de tu empresa
- Provincia*** (Province): Selecciona provincia
- Modalidad*** (Modality):

El botón “Sign-up” cierra la sesión y vuelve a mostrar la pantalla de inicio.



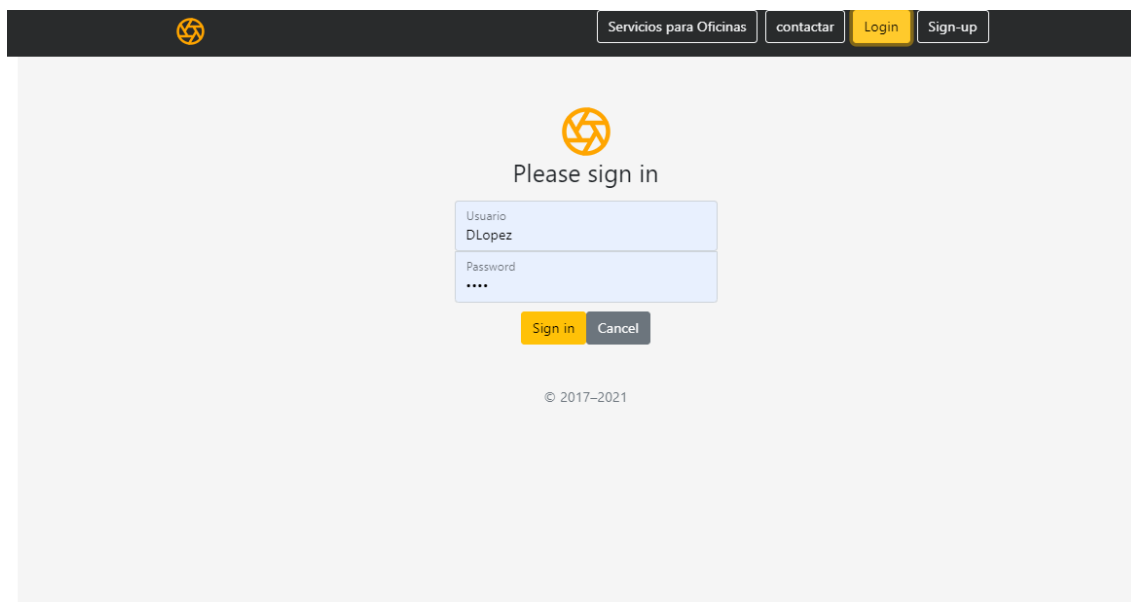
Pantalla servicios

Al hacer clic sobre el botón “Servicios para oficinas” se visualiza la pantalla con todos los servicios que se ofrecen al cliente:



Pantalla login para usuarios registrados

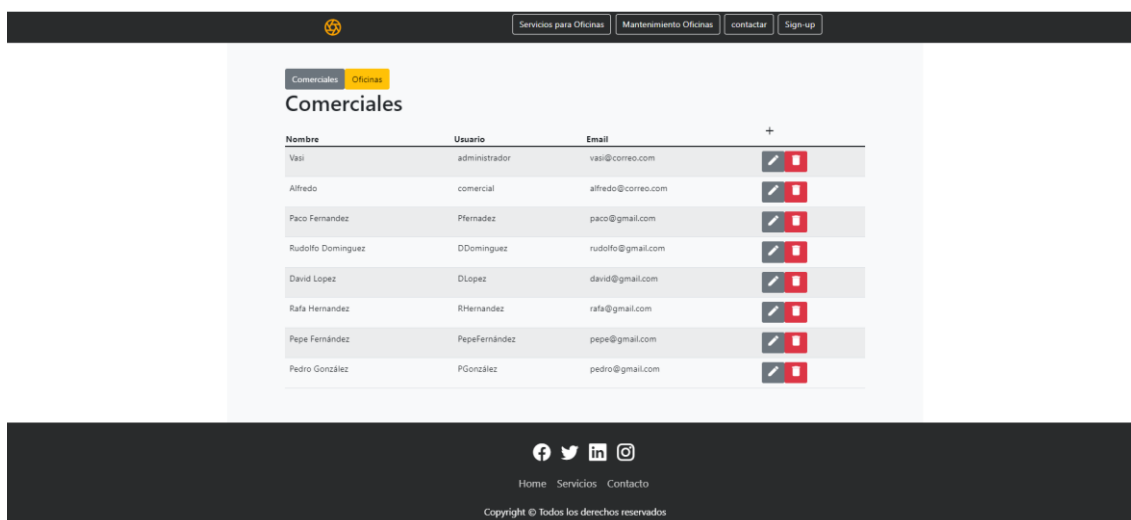
Los usuarios registrados tienen rol de administrador o de comercial.



















The login screen features a dark header with the company logo and navigation links: 'Servicios para Oficinas', 'contactar', 'Login', and 'Sign-up'. The main content area is light gray and contains the text 'Please sign in' with the company logo above it. Below this is a form with two input fields: 'Usuario' (containing 'DLopez') and 'Password' (containing four dots). At the bottom of the form are two buttons: 'Sign in' (yellow) and 'Cancel' (gray). The footer shows the copyright notice '© 2017-2021'.

Pantalla vista administrador

Una vez que el administrador inicie sesión y haga clic sobre el botón “Mantenimiento oficinas” visualiza la siguiente pantalla:

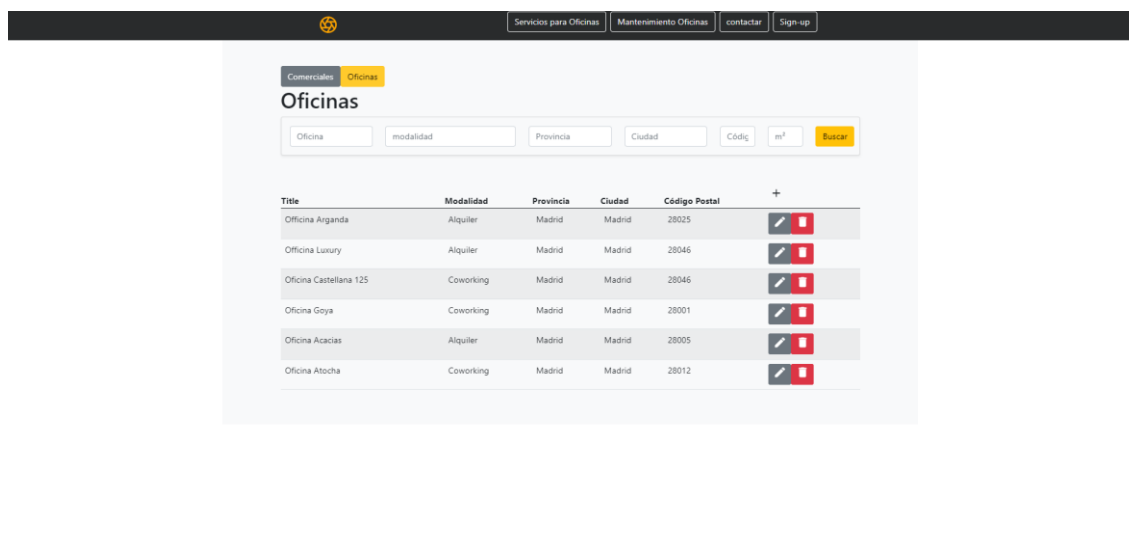


The administrator view shows a dark header with navigation links: 'Servicios para Oficinas', 'Mantenimiento Oficinas', 'contactar', and 'Sign-up'. The main content area has a sub-header 'Comerciales' with a toggle for 'Oficinas'. Below this is a table listing commercial users with columns for 'Nombre', 'Usuario', and 'Email'. Each row includes edit and delete icons. The footer contains social media icons, a navigation menu ('Home', 'Servicios', 'Contacto'), and a copyright notice.

Nombre	Usuario	Email	
Vasi	administrador	vasi@correo.com	 
Alfredo	comercial	alfredo@correo.com	 
Paco Fernandez	PFernandez	paco@gmail.com	 
Rudolfo Dominguez	DDominguez	rudolfo@gmail.com	 
David Lopez	DLopez	david@gmail.com	 
Rafa Hernandez	RHernandez	rafa@gmail.com	 
Pepe Fernández	PepeFernández	pepe@gmail.com	 
Pedro González	PGonzalez	pedro@gmail.com	 

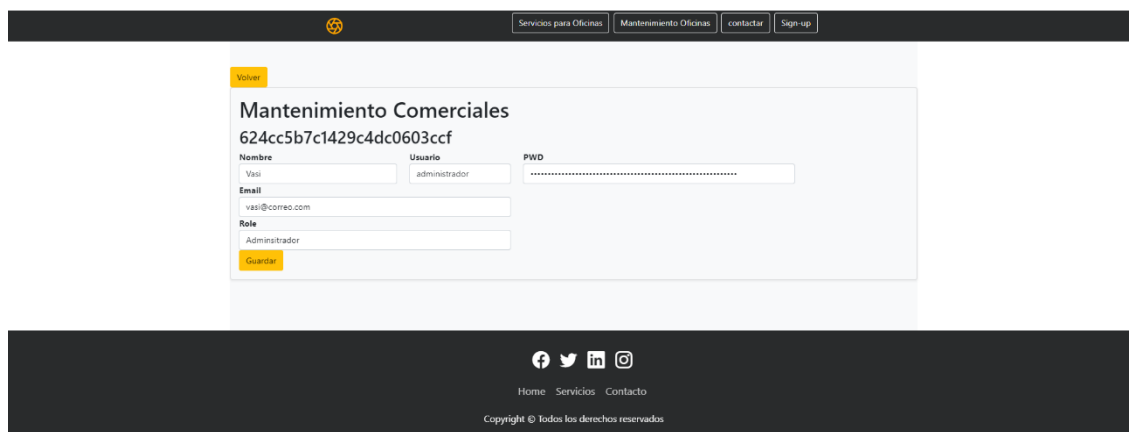
Se visualizan todos los comerciales que hay dados de alta con las opciones de editar y borrar.

También se visualiza el botón “Oficinas”. Al hacer clic en el se muestra un buscador y todas las oficinas dadas de alta con las opciones de editar y borrar.



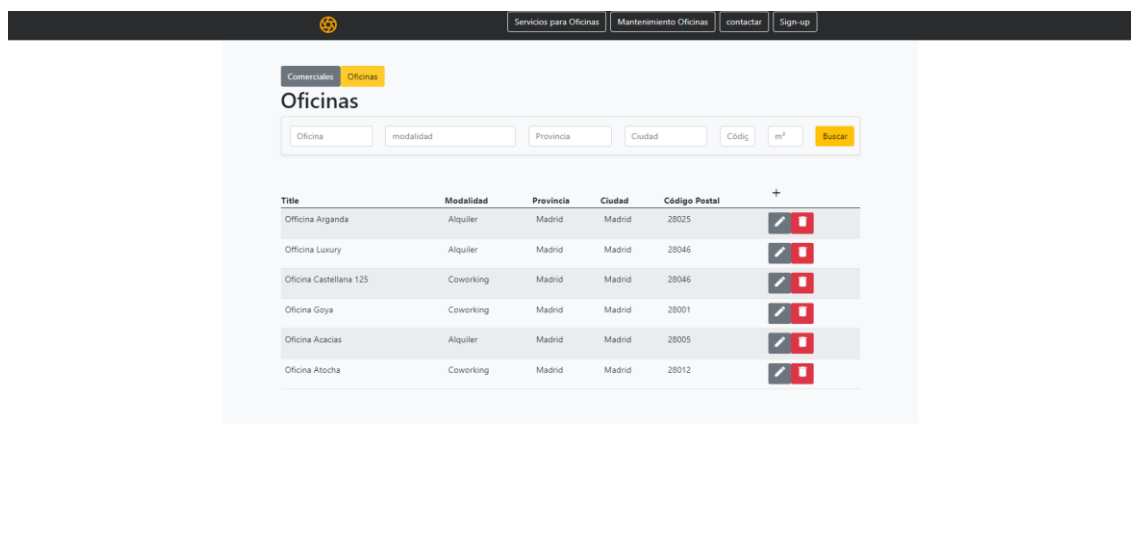
Pantalla editar comerciales

Si se selecciona la opción de “editar” haciendo clic sobre el botón, se visualiza la pantalla con todos los datos del comercial que se pueden modificar.




Pantalla editar oficinas

Si el administrador hace clic sobre el botón “Oficinas” y después elige la opción editar haciendo clic sobre el botón, se visualiza la pantalla con los datos de la oficina que se pueden modificar.




En la misma pantalla más abajo se visualiza el gestor de imágenes con las opciones de subir una imagen principal que es la que se visualiza primero y por otro lado está la opción de subir las imágenes secundarias. Todas las imágenes se guardan en la sección de “imágenes cargadas”. También se pueden borrar desde esa sección.



Servicios para OficinasMantenimiento OficinascontactarSign-up


Imagen Principal


Seleccionar archivoNinguno ...chivo selec.



Imágenes secundarias

Seleccionar archivoNinguno ...chivo selec.









Servicios para OficinasMantenimiento OficinascontactarSign-up


Imágenes cargadas

62681630eb4186662462a474







interior_2



interior_2.jpeg



interior_3



interior_3.jpeg

CONCLUSIONES

Una vez finalizado el proyecto, se puede decir que se ha cumplido con los requisitos especificados y se han alcanzado los objetivos planteados ya que, se ha construido una aplicación web que tiene las funcionalidades de buscar y gestionar oficinas de alquiler y coworking. Además, es una aplicación intuitiva, agradable a la vista y tiene un diseño responsive, dado que se adapta a los distintos dispositivos.

Con el desarrollo del proyecto, los alumnos han adquirido y han consolidado una serie de conocimientos que resultan importantes para la vida profesional de un desarrollador web.

Se pueden reafirmar muchas de las ventajas de utilizar MEAN Stack como entorno de trabajo.

Cabe destacar la facilidad para mejorar una aplicación creada en este entorno. A lo largo de este proyecto se han ido incorporando nuevas funcionalidades y mejoras dependiendo de los requisitos necesarios en cada momento. Gracias a Angular en el caso del front-end, o de Express, en el back-end, incorporar esas mejoras ha sido una tarea más sencilla comparado con otras tecnologías.

Otro de los puntos fuertes ha sido la comunidad de desarrolladores de JavaScript. Cualquier duda o problema encontrado ha sido fácilmente corregido gracias a toda la información encontrada en internet. Portales como StackOverflow son de gran ayuda para encontrar solución a muchos problemas ya encontrados antes por otros desarrolladores.

MEJORAS DEL PROYECTO

A pesar de que la aplicación, una vez finalizado su desarrollo cumple con los requisitos establecidos al inicio del proyecto, es cierto que se pueden añadir múltiples funcionalidades que la pueden convertir en una herramienta más completa para el sector inmobiliario en algunos aspectos.

Posibles mejoras:

- Una mejora a aplicar en un futuro sería la de añadir una funcionalidad para poder elegir más de un idioma.
- Añadir un apartado para noticias del sector inmobiliario.
- Aunque la idea original es la búsqueda solamente de oficinas de alquiler y coworking, se puede ampliar para viviendas, suelos, naves industriales, etc.

BIBLIOGRAFÍA

Documentación oficial de MongoDB

- <https://www.mongodb.com/docs/>

Documentación oficial de Node.js

- <https://nodejs.org/api/>

Documentación oficial de gestor de paquetes NPM

- <https://docs.npmjs.com/>

Documentación oficial de Express

- <http://expressjs.com/es/4x/api.html>

Documentación oficial de Angular

- <https://angular.io/api>

Documentación oficial de Bootstrap

- <https://getbootstrap.com/>

Tutorial de JavaScript

- <https://www.w3schools.com/js/default.asp>

Documentación oficial Compodoc

- <https://compodoc.app/>

Documentación oficial Swagger

- <https://swagger.io/>

Enlace a preguntas relacionadas con Angular en StackOverflow

- <https://stackoverflow.com/questions/tagged/angular>

Repositorios del proyecto (Front-End y Back-End)

- <https://github.com/Vasi81/TFG-Front>
- <https://github.com/Vasi81/TFG-Back-End>