Eligiendo un futuro

MEMORIA DESCRIPTIVA
Desarrollo de aplicaciones web
Mayo 2024

DAVID CEREZO HERNÁNDEZ

Memoria descriptiva

Contenido

1. Introducción

1.1. Objeto del proyecto.

Eligiendo un futuro es una aplicación web para las familias que se encuentran ante uno de los momentos más importantes para el futuro de las nuevas generaciones, como es la decisión de qué colegio elegir para nuestros peques.

También es una aplicación que da la oportunidad a las escuelas, colegios e institutos de darse a conocer y mostrar su oferta educativa.

En esta aplicación podremos registrarnos como usuarios o como colegios. Podremos ver la oferta educativa de los colegios que formen parte de nuestra web además de poder hacer comentarios y valoraciones.
Los colegios podrán responder a estos comentarios y así se fomentara una comunicación fluida entre usuarios y colegios.

Podremos filtrar los colegios basándonos en diferentes filtros o criterios de búsqueda para así encontrar los que mejor se adapten a las necesidades de las familias.

La justificación de este proyecto se basa en que la elección de colegio es una decisión crucial para cualquier familia y debido a la cantidad de opciones que se presentan el proceso puede ser abrumador.

Se necesita información confiable y accesible y en ese sentido nuestra web proporciona una plataforma que ofrece esas cualidades debido a las valoraciones de otros usuarios, así como a las informaciones de los propios centros.

Y el sentido fundamental es darles a las familias una herramienta que les facilite esta importante decisión y elección, centralizando las ofertas educativas en una sola web.

1.2. Lenguajes empleados:

- Lenguajes de Programación:
    - TypeScript: Empleado en el frontend con Angular para proporcionar tipado estático y características avanzadas de desarrollo.

    - JavaScript: Utilizado en el frontend y en el backend para la ejecución de código en el navegador y en el servidor.
    - Java: Utilizado en el backend con Spring Boot para la lógica del servidor y el acceso a la base de datos.
- Frameworks:
    - Angular: Framework utilizado en el frontend para la construcción de la interfaz de usuario.
    - Spring Boot: Framework utilizado en el backend para el desarrollo de la api.
    - Express.js: Framework utilizado en el backend con Node.js para la autentificación.
    - PrimeNg: Framework para el diseño utilizado.
    - Angular Material: Framework para el diseño utilizado.
    - Bootstrap: Framework para el diseño utilizado.
- Base de Datos:
    - MySQL: Sistema de gestión de bases de datos relacional utilizado para almacenar y administrar datos estructurados.
- Herramientas de Gestión de Bases de Datos:
    - DBeaver: Herramienta de gestión de bases de datos utilizada para interactuar con la base de datos MySQL y ejecutar consultas SQL.
- Entornos de Desarrollo Integrados (IDE):
    - Visual Studio Code (VS Code): IDE utilizado para el desarrollo frontend y backend (parte Node), con soporte para TypeScript, JavaScript y Java, así como extensiones necesarias.
    - IntelliJ IDEA: IDE utilizado específicamente para el desarrollo en Java, con características avanzadas para el desarrollo con Spring Boot.
- Despliegue con Docker Compose:
    - Despliegue de la base de datos.
- JSON:
    - Archivos extraídos del portal de datos abiertos de la Junta de Castilla y Leon.

1.3. Distribución

El despliegue de la aplicación se realizará localmente desde la máquina de desarrollo.

Método de Despliegue:

Maven y Tomcat**:** Se utiliza Maven para gestionar dependencias y construir el proyecto Java. La aplicación Java será desplegada en un servidor web Tomcat en la máquina local utilizando JDK 17.

Angular y Node.js**:** Se utiliza Node.js para ejecutar el servidor de desarrollo de Angular localmente. Angular CLI se encargará de compilar y construir el frontend de la aplicación Angular.

Características del Hosting necesario: Se utilizará un entorno de hosting local en la máquina de desarrollo. No se requiere acceso a un servidor externo, ya que la aplicación será desplegada y ejecutada en la misma máquina donde se realiza el desarrollo del software.

Sistema Operativo: La máquina local tiene un sistema Windows, se a probado tanto en Windows 10 como en el 11

Recursos del Sistema: Sera necesario tener un almacenamiento en disco suficiente para poder instalar la aplicación aproximadamente un Gigabyte sería suficiente. Además, una memoria RAM de 16 Gb y un procesador i5 de séptima generación al menos

Configuración de Red: Acceso a internet de alta velocidad para la optima carga de la página

1.4. Requisitos de los clientes

- Navegador Web:

Los clientes deberán tener acceso a un navegador web como Google Chrome, Mozilla Firefox, Microsoft Edge, Opera o Safari.

- Conexión a internet:

Se requerirá una conexión a Internet estable y de banda ancha para acceder a la herramienta y cargar el contenido de manera eficiente.

- Dispositivos compatibles:

Los clientes podrán acceder a la herramienta desde una variedad de dispositivos, incluyendo equipos de escritorio, portátiles, tabletas y teléfonos.

1.5. Licencia

Se distribuirá bajo la Licencia Pública General de GNU (GPL), o cualquier versión de esa licencia que se considere apropiada para el proyecto. Esta elección asegurará que la herramienta sea considerada software libre y esté alineada con los principios del movimiento del software libre.

2. Recursos

2.1. Hardware

El desarrollo, implementación y distribución requerirá el siguiente hardware:

- Desarrollo:

Se utilizará un equipo con capacidades suficientes de procesamiento y memoria RAM para ejecutar los entornos de desarrollo y las herramientas necesarias, incluyendo IDE, compiladores y servidores de aplicaciones por lo que necesitaremos al menos unos 16GB de RAM y un procesador adecuado, como un procesador Intel Core i5 o equivalente de AMD, para garantizar un rendimiento óptimo.

Se necesitará una conexión a Internet estable para acceder a recursos en línea, descargar dependencias y por ejemplo el acceso a repositorios remotos.

- Implementación y Distribución:

Servidor de implementación:

Se utilizará el equipo de desarrollo, para implementar la aplicación de manera local pudiendo ser visibles por otros equipos que se encuentren dentro de la misma red local o desde fuera mediante conexión remota.

- Dispositivos de Usuario Final:

Los usuarios finales accederán a través de diversos dispositivos, como equipos de escritorio, portátiles, tabletas o teléfonos inteligentes.

2.2. Software

El desarrollo, implementación y distribución de la herramienta requerirá el siguiente software:

Desarrollo:

Entorno de Desarrollo Integrado (IDE):

Se utilizarán como IDES. IntelliJ IDEA para el backend con Java y Visual Studio Code para el Frontend con angular y para la parte del backend con Express. Ambos para escribir, editar y depurar el código de la aplicación.

Gestión de Dependencias:

Se empleará Maven como herramienta de gestión de dependencias para el proyecto Java.

Y Node con npm para las dependencias necesarias en Express y Angular.

Framework de Desarrollo Frontend:

Se utilizará Angular como framework de desarrollo frontend para la construcción de la interfaz de usuario.

Entorno de Ejecución de JavaScript:

Node.js se utilizara para ejecutar el servidor de desarrollo de Angular y para instalar las dependencias de Node.js necesarias para el frontend además de usarlo para Express para el login, y la comunicación con el servidor de correo.

Implementación y Distribución:

Servidor de Aplicaciones Java:

Se utilizará Apache Tomcat como servidor de aplicaciones Java para implementar la parte del backend de la aplicación.

Gestión de Versiones:

Se empleará Git como sistema de control de versiones para gestionar el código de la aplicación.

Herramientas de Despliegue:

Se utilizará Docker y Docker Compose para el empaquetado y despliegue del servidor de base de datos MySQL.

Sistema Operativo:

Se utilizará el sistema operativo Windows en las versiones 10 y 11

2.3. Humanos

Análisis, diseño y Estudio:12 horas aprox

Desarrollo: 180 horas

Documentación: 40 horas

Total: 232 horas

2.4. Previsión económica

Análisis, diseño y Estudio: 12h *20€= 240€
Desarrollo: 180h*20€ =3600€
Documentación: 40h * 20€= 800€
Total: 232 horas * 20€ = 4640€

El precio por hora es el que consultando con la empresa me dijo que es a lo que se facturaría la hora de un junior sin experiencia.

3. Descripción de la aplicación

3.1. Funcionamiento general

La aplicación tiene como objetivo el acceso a la información de los centros educativos de la junta de Castilla y León.
En esta aplicación tenemos tres roles diferenciados que son el de administrados, el de colegios y el de usuarios. Todos ellos tendrán ciertas partes comunes de la aplicación y otras opciones y comportamientos diferenciados por el rol de estos.
Nada más acceder a la página se podrá ver una página de inicio que nos mostrará la información de esta además de las imágenes subidas por los centros.

Desde aquí podremos acceder al login para acceder a la parte privada de la aplicación o registrarnos en esta.

O podrán registrarse como usuarios o como colegios (esto tendrá que verificarse por el administrador).

Una vez dentro el usuario tendrá acceso a las zonas comunes que serán el acceso al listado de colegios, su buscador y desde esta parte se podrá acceder al detalle de cada colegio con toda la información de estos.

En función del tipo de rol que tenga el usuario podrá interactuar de una manera u otra con estás partes de la web, en las zonas comunes los que tengan el rol usuario (o el administrador de la página) podrán valorar o comentar estos colegios.
Además, contarán con el acceso a las zonas privadas de los distintos roles que detallamos a continuación.

En las zonas privadas tanto lo que se verá como el cómo se interactuara con la web es muy diferente dependiendo del rol.

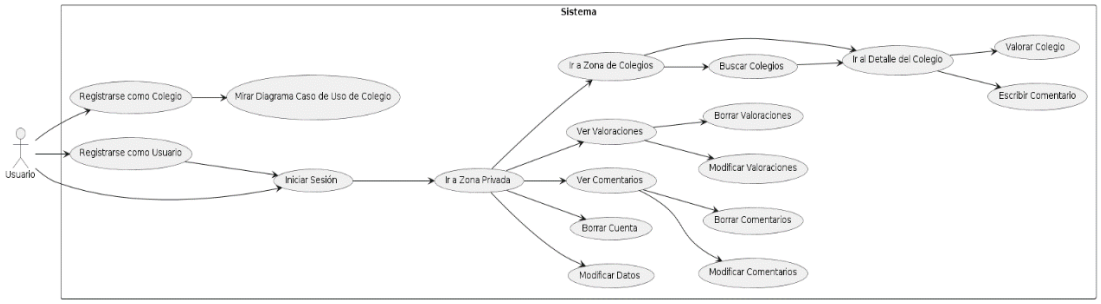
Los usuarios con el rol usuario podrán modificar sus datos, también podrán ver sus comentarios, cambiarlos o borrarlos y las respuestas de estos si es que las tuviesen y por último eliminar la cuenta (el borrado es lógico).

Los colegios podrán modificar sus datos de acceso además de la descripción (el resto de los campos viene marcado por los datos abiertos de la junta), podrán subir imágenes y borrarlas, ver los comentarios que se les han dejado además de poder responder a estos y cambiar sus respuestas o borrarlas, añadir actividades de la lista de actividades o crear nuevas además de poder modificar o eliminar las que ya tienen y borrar su usuario (el borrado es lógico).

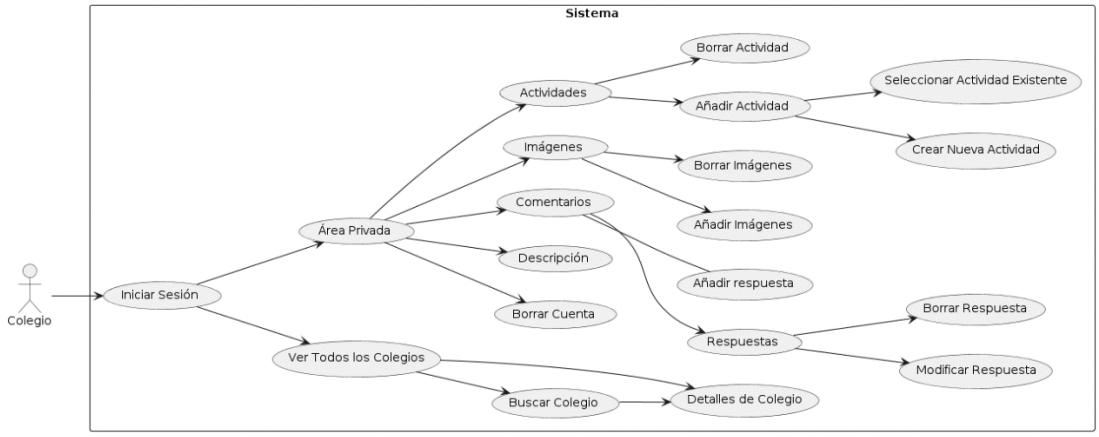
El administrador por ultimo tendrá acceso a toda la información relevante de la página, listado de usuarios, colegios, comentarios, e imágenes y podrá hacer el borrado (borrado físico) o la desactivación (borrado lógico) o activación de estos. además podrá ver las nuevas solicitudes de usuarios que se registraron como colegios tendrá que registrarlos en el caso de que estos se encuentren en los datos oficiales de la junta.
Los usuarios podrán solicitar un email para recuperar la contraseña, ya que cada vez es más difícil recordar todas.

Diagramas de casos de uso:

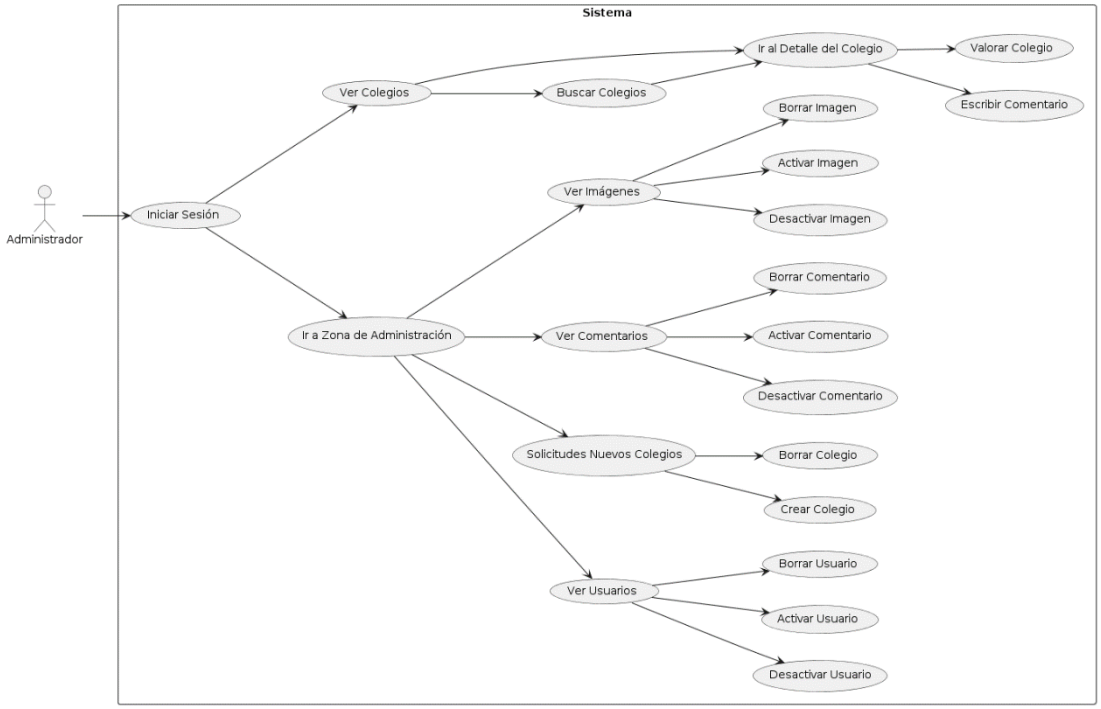
- **Usuario:**



- **Colegio:**

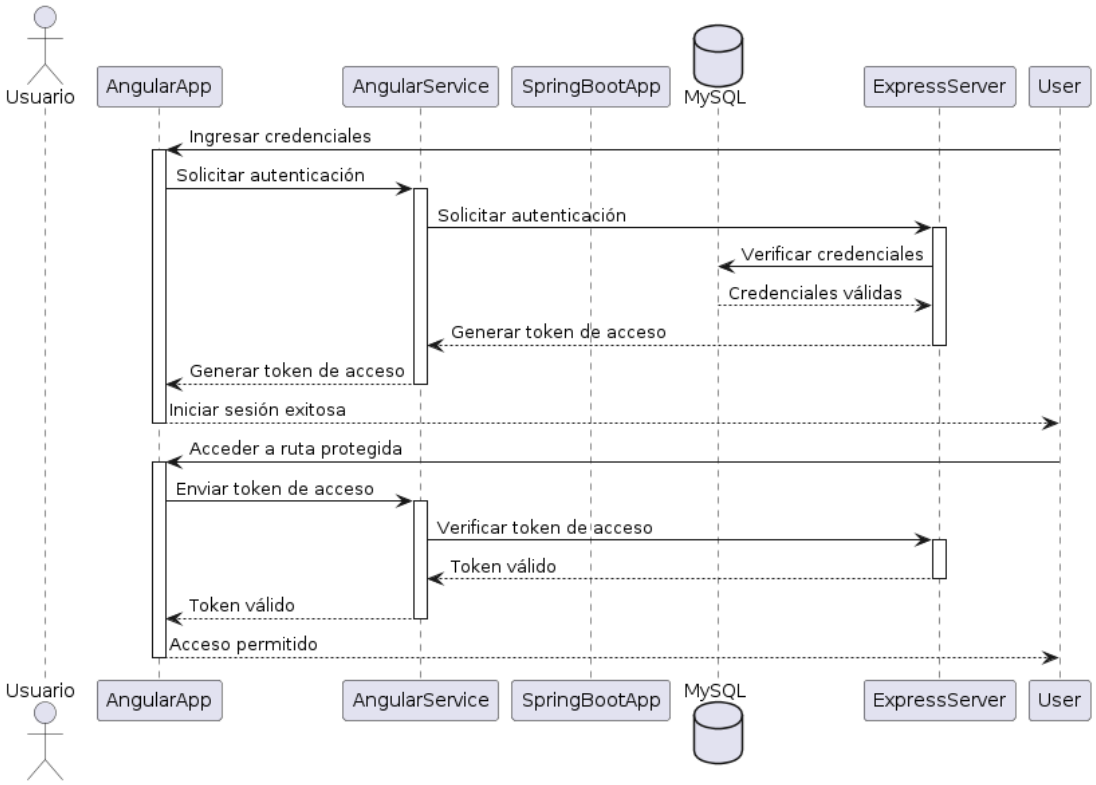


- **Administrador:**

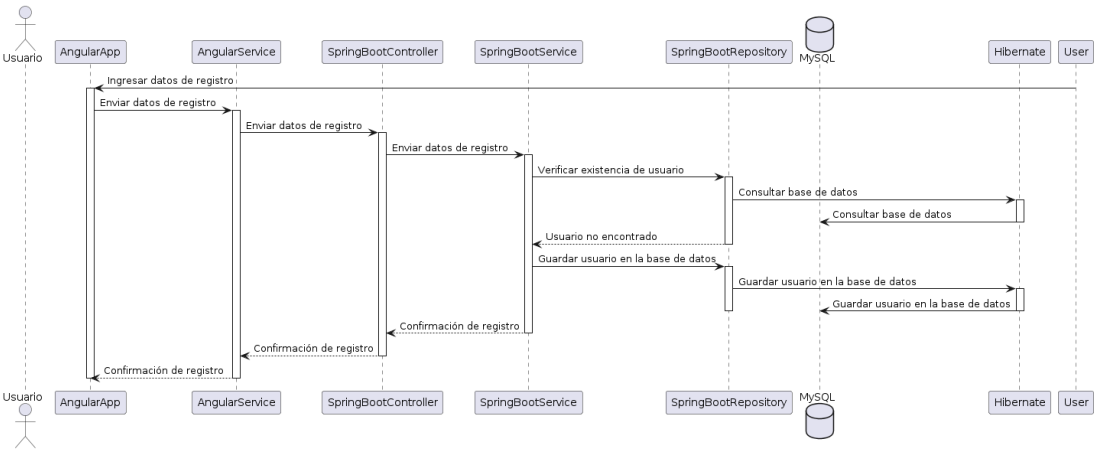


Diagramas secuencia:

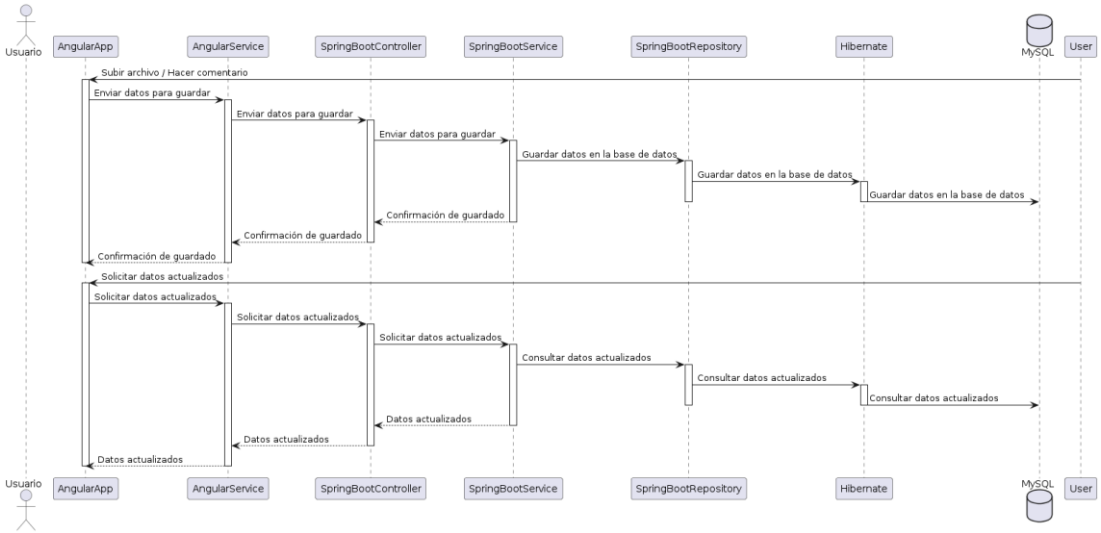
- **Login:**



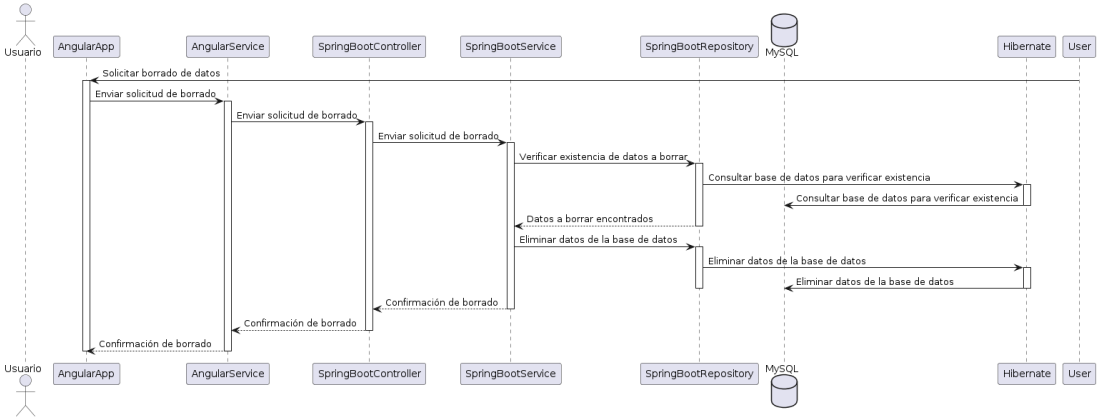
- **Registro:**



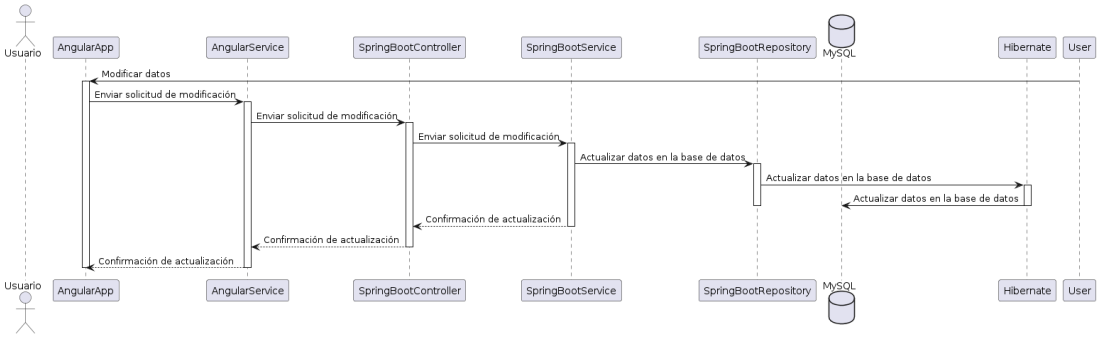
- **Crear**



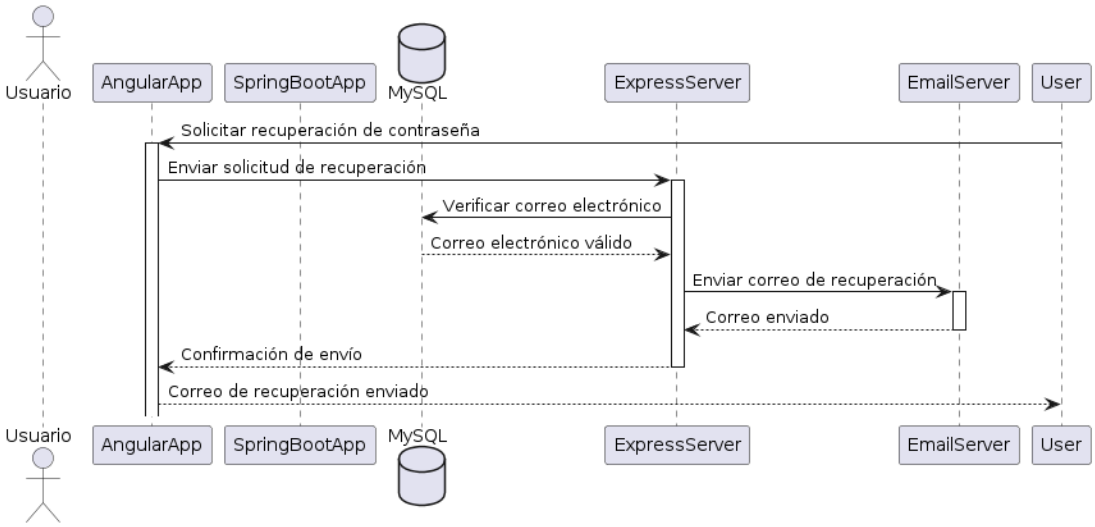
- **Borrar:**



- **Modificar:**



- **Recuperación contraseña:**



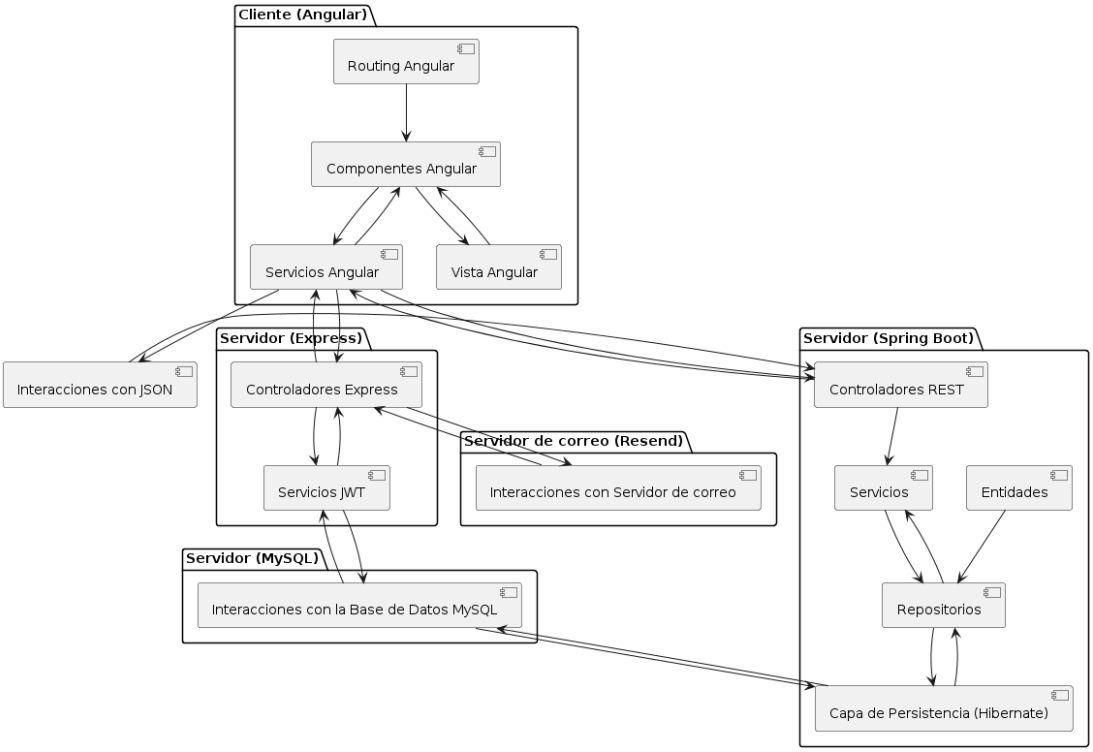
3.2. Arquitectura

En la arquitectura del sistema se usa el patron Modelo-Vista-Controlador.
Se emplean diversas tecnologías y herramientas para garantizar un desarrollo eficiente y robusto tanto en el frontend como en el backend. Para el frontend, he optado por Angular 16. Se han implementado servicios, componentes, guards, interfaces y pipes para aprovechar este proyecto para el aprendizaje personal. Además, se usan distintos submódulos para una organización eficiente del código y aprovechar la carga perezosa.

En cuanto al backend, he decidido emplear Node.js junto con Express.js para la autenticación de usuarios. Para el resto de la lógica de negocio y el manejo de datos, opté por Spring Boot de Java. Usando la arquitectura de tres capas se han desarrollado controladores, entidades, servicios y repositorios utilizando Spring Boot Hibernate se ha integrado con Spring Boot para la gestión de la capa de acceso a datos, facilitando el mapeo objeto-relacional y simplificando las operaciones de base de datos.

Además, se hace uso de archivos JSON para el almacenamiento y la manipulación de ciertos datos dentro de la aplicación (ya que al no estar desplegada en un servidor la api de la que se iba acceder nos da problemas con las CORS) no he tenido tiempo de implantarlo pero la idea es usar MongoDb para gestionar esos archivos JSON.

Por último, se ha usado un servidor de correo electrónico para la funcionalidad de envío de correos en su versión gratuita para pruebas en el desarrollo, aunque la lógica esta implementada y lista para su versión de pago cuando la aplicación este desplegada.
Diagrama de arquitectura:

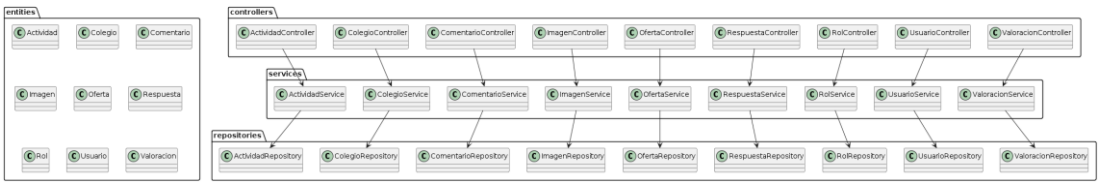


Diagramas de paquetes:

Angular:

Todos las carpetas tanto de shared como de components incluyen los archivos .ts .htm y .scss

Spring Boot:



3.2.1. Diseño de las bases de datos

Tras la idea del proyecto tuve que estudiar el diseño y la estructura de base de datos en función de mi proyecto, tras una primera versión en la que no afrontaba el borrado lógico me surgió la idea de implementarlo, pero con el proyecto ya en una fase bastante avanzada y con poco tiempo para ello decidí modificarlo para poder llevar a cabo esa idea, para ello añadí los campos activo y desactivaciones para

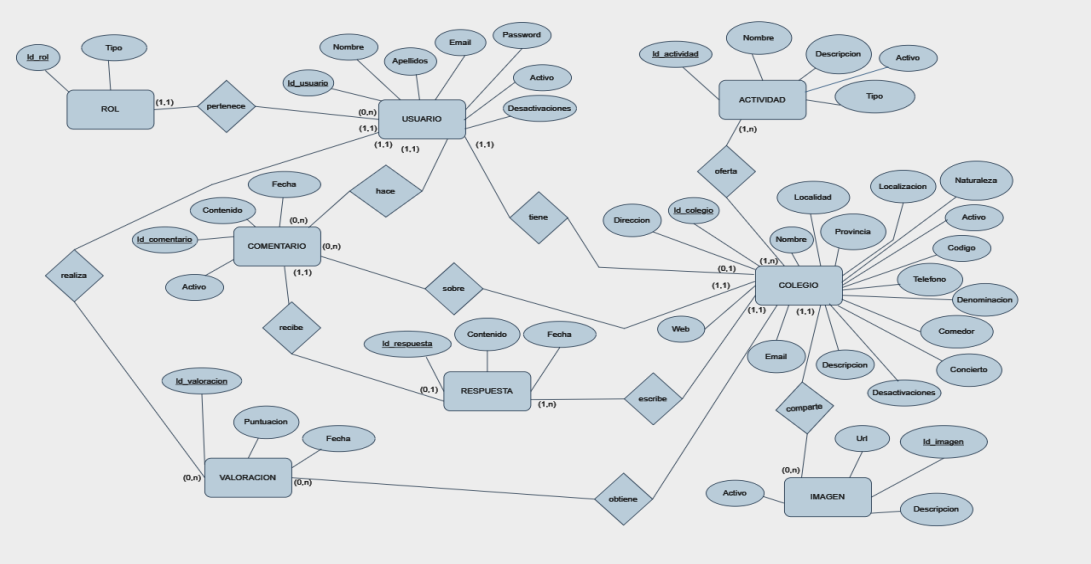
junto con el email forman una clave única y por tanto al borrarse un usuario de forma lógica se pone el campo activo en falso y se suma uno en desactivaciones luego cada vez que se borra un usuario o colegio recorrerá los demás y usuarios con el campo activo en false y sumara uno en el campo desactivaciones.

Además, en las otras tablas que tienen el campo activo no es necesario poner desactivaciones ya que pueden haber múltiples resultados, como es el caso por ejemplo de comentarios ya que la clave es el id y un colegio puede tener muchos comentarios de una misma persona

Otra de las cosas a tener en cuenta es que en mi estructura cada comentario solo puede tener una única respuesta.

También e decidido que las actividades de los colegios puedan tener borrado físico y las valoraciones no se puedan modificar una vez que se vote.

A continuación, muestro tanto el modelo entidad relación como el paso a relacional y los ddl correspondientes:



Paso a Relacional

PASO 1

ROL (**Id_rol**, Tipo)

USUARIO (**Id_usuario**, Nombre, Apellidos, Email, Password, Activo, Desactivaciones)

ACTIVIDAD (**Id_actividad**, Nombre, Descripción, Tipo, Activo)

COMENTARIO (**Id_comentario**, Contenido, Fecha, Activo)

VALORACION (**Id_valoracion**, Puntuación, Fecha)

RESPUESTA (**Id_respuesta ,** Contenido, Fecha)

COLEGIO (**Id_colegio**, Nombre, Provincia, Localidad, Localización, Naturaleza, Telefono, Denominación, Comedor, Concierto, Descripción. Email, Web, Direccion, Codigo, Activo, Desactivaciones)

IMAGEN (**Id_imagen**, Url, Descripcion)

pertenece (Id_rol, Id_usuario)

tiene (Id_colegio, Id_usuario)

oferta (Id_actividad, Id_colegio)

obtiene (id_colegio, Id_valoracion)

escribe (Id_colegio, Id_respuesta)

sobre (Id_colegio, Id_comentario)

recibe (Id_respuesta, Id_comentario)

realiza (Id_valoracion, Id_usuario)

comparte (Id_colegio, Id_imagen)

hace (Id_usuario, Id_comentario)

PASO 2

ROL (**Id_rol**, Tipo)

USUARIO (**Id_usuario**, Nombre, Apellidos, Email, Password, Activo, Desactivaciones, Rol(Id_rol)

ACTIVIDAD (**Id_actividad**, Nombre, Descripción, Tipo, Activo)

COMENTARIO (**Id_comentario**, Contenido, Fecha, Activo, Usuario(Id_usuario), Colegio(Id_colegio))

VALORACION (**Id_valoracion**, Puntuación, Fecha, Usuario(Id_usuario), Colegio(Id_colegio))

RESPUESTA (**Id_respuesta ,** Contenido, Fecha, Comentario(Id_comentario), Colegio(Id_colegio))

COLEGIO (**Id_colegio**, Nombre, Provincia, Localidad, Localización, Naturaleza, Telefono, Denominación, Comedor, Concierto, Descripción. Email, Web, Direccion, Codigo, Activo, Desactivaciones, Usuario(Id_usuario))

IMAGEN (**Id_imagen**, Url, Descripcion Activo, Colegio(Id_colegio))

~~**pertenece (Id_rol, Id_usuario)**~~

~~**tiene (Id_colegio, Id_usuario)**~~

oferta (Id_oferta, Id_actividad, Id_colegio**)** ~~**obtiene (id_colegio, Id_valoracion)**~~

~~**escribe (Id_colegio, Id_respuesta)**~~

~~sobre (Id_colegio, Id_comentario)~~

~~recibe (Id_respuesta, Id_comentario)~~

~~realiza (Id_valoracion, Id_usuario)~~

~~comparte (Id_colegio, Id_imagen)~~

~~hace (Id_usuario, Id_comentario)~~

DDl:

Imagen:

```
CREATE TABLE `imagen` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `colegio_id` int DEFAULT NULL,
  `url` mediumblob,
  `descripcion` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci DEFAULT NULL,
  `activo` tinyint(1) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `imagen_UN` (`colegio_id`,`descripcion`),
  KEY `IDX_8319D2B37FDC9E6F` (`colegio_id`),
  CONSTRAINT `FK_8319D2B37FDC9E6F` FOREIGN KEY (`colegio_id`)
REFERENCES `colegio` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=48 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

Colegio:

```
CREATE TABLE `colegio` (
  `id` int NOT NULL AUTO_INCREMENT,
  `usuario_id` int NOT NULL,
  `nombre` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
NOT NULL,
  `provincia` varchar(30) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
NOT NULL,
  `localidad` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
NOT NULL,
  `denominacion` varchar(100) CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci NOT NULL,
  `naturaleza` varchar(20) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
NOT NULL,
  `comedor` tinyint(1) NOT NULL,
  `concierto` tinyint(1) NOT NULL,
```

```
  `email` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT
NULL,
  `web` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
DEFAULT NULL,
  `descripcion` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci DEFAULT NULL,
  `direccion` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
NOT NULL,
  `telefono` varchar(18) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
NOT NULL,
  `localizacion` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci DEFAULT NULL,
  `codigo` varchar(100) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
NOT NULL,
  `activo` tinyint(1) NOT NULL,
  `desactivaciones` bigint DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `colegio_UN` (`email`,`activo`,`desactivaciones`),
  KEY `IDX_AA00D876DB38439E` (`usuario_id`),
  CONSTRAINT `FK_AA00D876DB38439E` FOREIGN KEY (`usuario_id`)
REFERENCES `usuario` (`id`) ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=17 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

Usuario:

```
CREATE TABLE `usuario` (
  `id` int NOT NULL AUTO_INCREMENT,
  `rol_id` int NOT NULL,
  `nombre` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
NOT NULL,
  `apellidos` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
DEFAULT NULL,
  `email` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT
NULL,
  `password` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
NOT NULL,
  `activo` tinyint(1) NOT NULL,
  `desactivaciones` bigint DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `usuario_UN` (`activo`,`desactivaciones`,`email`),
  KEY `IDX_2265B05D4BAB96C` (`rol_id`),
  CONSTRAINT `FK_2265B05D4BAB96C` FOREIGN KEY (`rol_id`) REFERENCES
`rol` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=61 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

Actividad:

```
CREATE TABLE `actividad` (
  `id` int NOT NULL AUTO_INCREMENT,
  `nombre` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
NOT NULL,
  `tipo` varchar(20) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT
NULL,
  `activo` tinyint(1) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `actividad_UN` (`tipo`,`nombre`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

Rol

```
CREATE TABLE `rol` (
  `id` int NOT NULL AUTO_INCREMENT,
  `tipo` varchar(20) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT
NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

Oferta:

```
CREATE TABLE `oferta` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `actividad_id` int NOT NULL,
  `colegio_id` int NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `oferta_UN` (`actividad_id`,`colegio_id`),
  KEY `IDX_7479C8F26014FACA` (`actividad_id`),
  KEY `IDX_7479C8F27FDC9E6F` (`colegio_id`),
  CONSTRAINT `FK_7479C8F26014FACA` FOREIGN KEY (`actividad_id`)
REFERENCES `actividad` (`id`),
  CONSTRAINT `FK_7479C8F27FDC9E6F` FOREIGN KEY (`colegio_id`)
REFERENCES `colegio` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=19 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

Valoración:

```
CREATE TABLE `valoracion` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `usuario_id` int DEFAULT NULL,
  `colegio_id` int DEFAULT NULL,
  `fecha` datetime(6) DEFAULT NULL,
  `puntuacion` int NOT NULL,
  PRIMARY KEY (`id`),
  KEY `IDX_6D3DE0F4DB38439E` (`usuario_id`),
  KEY `IDX_6D3DE0F47FDC9E6F` (`colegio_id`),
  CONSTRAINT `FK_6D3DE0F47FDC9E6F` FOREIGN KEY (`colegio_id`)
REFERENCES `colegio` (`id`),
  CONSTRAINT `FK_6D3DE0F4DB38439E` FOREIGN KEY (`usuario_id`)
REFERENCES `usuario` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=28 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

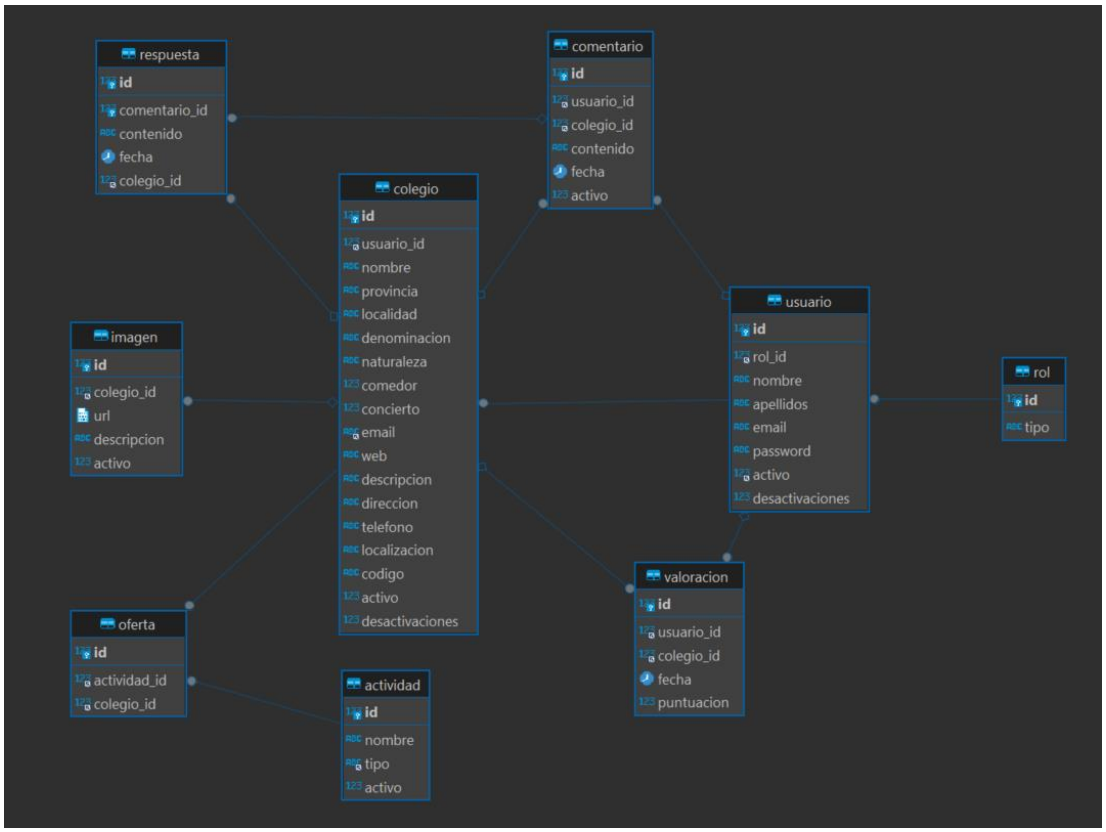
Comentario:

```
CREATE TABLE `comentario` (
  `id` int NOT NULL AUTO_INCREMENT,
  `usuario_id` int DEFAULT NULL,
  `colegio_id` int DEFAULT NULL,
  `contenido` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
NOT NULL,
  `fecha` datetime DEFAULT NULL,
  `activo` tinyint(1) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `IDX_4B91E702DB38439E` (`usuario_id`),
  KEY `IDX_4B91E7027FDC9E6F` (`colegio_id`),
```

```
  CONSTRAINT `FK_4B91E7027FDC9E6F` FOREIGN KEY (`colegio_id`)
REFERENCES `colegio` (`id`),
  CONSTRAINT `FK_4B91E702DB38439E` FOREIGN KEY (`usuario_id`)
REFERENCES `usuario` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=21 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

Respuesta:

```
CREATE TABLE `respuesta` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `comentario_id` int DEFAULT NULL,
  `contenido` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
NOT NULL,
  `fecha` date NOT NULL,
  `colegio_id` int DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `UNIQ_6C6EC5EEF3F2D7EC` (`comentario_id`),
  KEY `IDX_6C6EC5EE7FDC9E6F` (`colegio_id`),
  CONSTRAINT `FK_6C6EC5EE7FDC9E6F` FOREIGN KEY (`colegio_id`)
REFERENCES `colegio` (`id`) ON DELETE RESTRICT ON UPDATE CASCADE,
  CONSTRAINT `FK_6C6EC5EEF3F2D7EC` FOREIGN KEY (`comentario_id`)
REFERENCES `comentario` (`id`) ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=24 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```



3.2.2. Arquitectura del sistema

Empezamos con el diagrama de clases de las clases más importantes de mi aplicación como son en este caso las entidades en dentro de mi api rest :

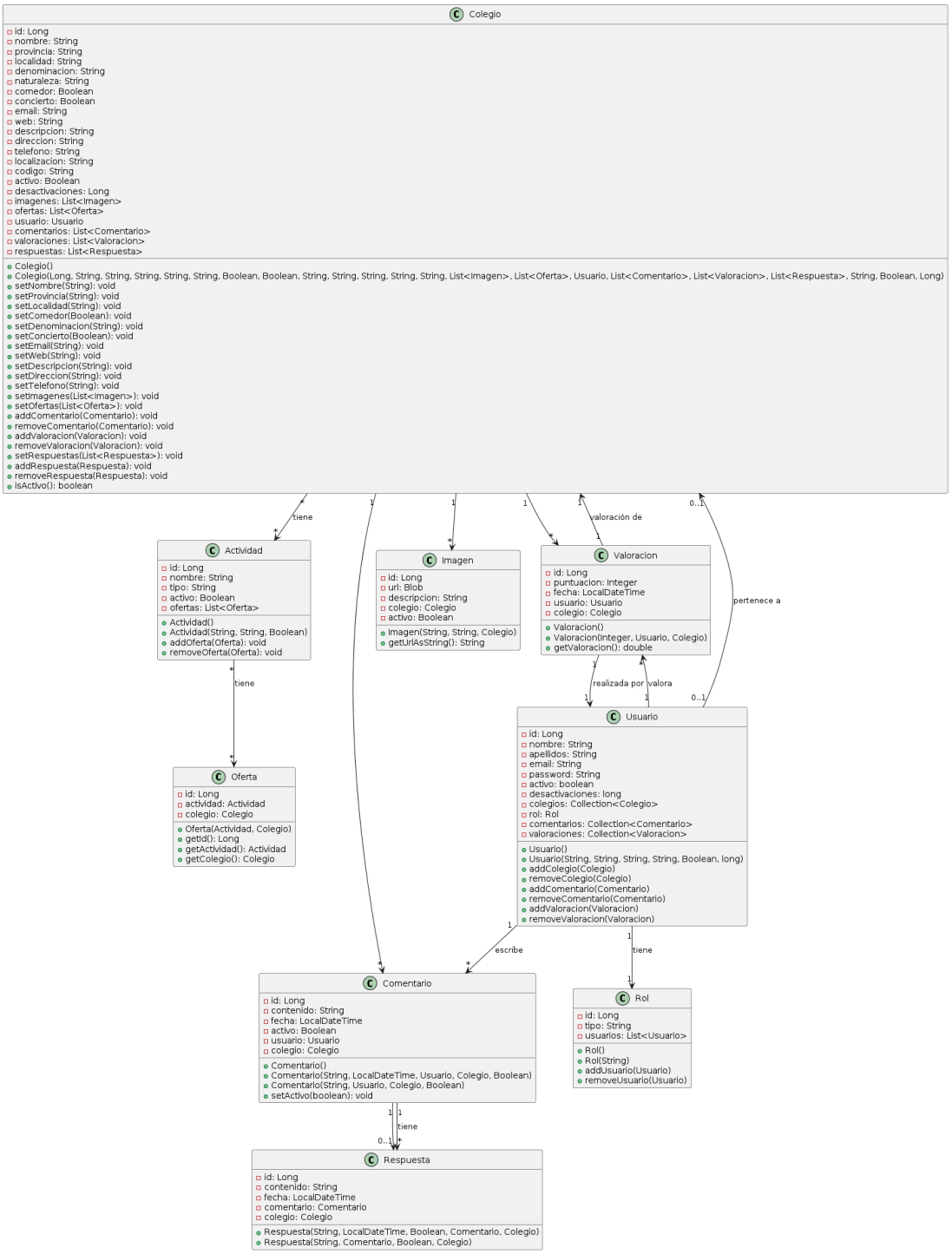
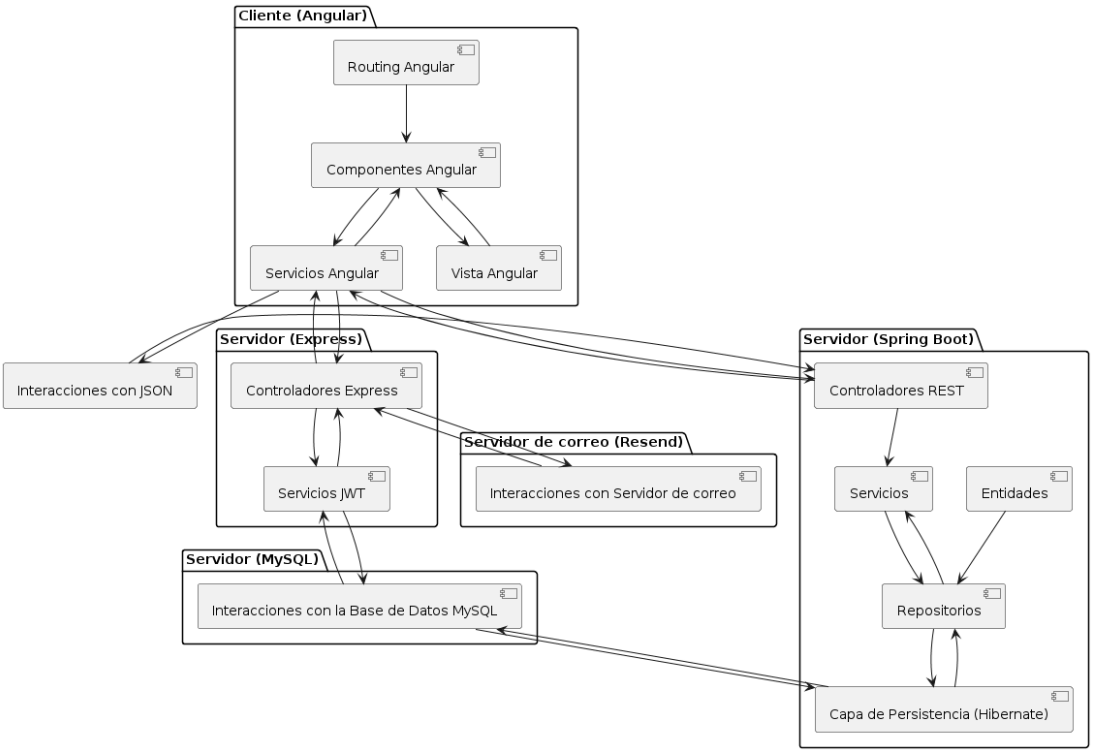


Diagrama de arquitectura:



Cliente (Angular):

Este es el lado del frontend, construido con Angular. Incluye componentes, servicios, routing y vistas.

Los componentes son las diferentes partes de la aplicación con sus archivos:

- Component.html para la vista.
- Component.scss(en mi caso) para el estilo.
- Component.ts para la lógica

Los servicios son funciones reutilizables compartidas entre los componentes y las partes encargadas de conectarse con la api..

El routing maneja la navegación entre las diferentes vistas de la aplicación.

Las vistas son las páginas que los usuarios ven y con las que interactúan.

Servidor (Express):

Este es el servidor backend construido con Express, de Node.js.

Incluye controladores y servicios relacionados con la autenticación JWT (JSON Web Tokens).

Los controladores manejan las solicitudes HTTP y las rutas de la aplicación.

Los servicios JWT se encargan de la autenticación y autorización utilizando tokens.

Hay interacciones con la base de datos MySQL para la persistencia de datos.

Servidor (Spring Boot):

Este es otro servidor backend construido con Spring Boot, un framework de Java.

Incluye controladores REST, entidades, servicios y repositorios.

Los controladores REST manejan las solicitudes HTTP y las rutas de la API.

Las entidades representan los objetos de datos en la aplicación.

Los servicios contienen la lógica de la aplicación.

Los repositorios manejan la interacción con la base de datos a través de Hibernate, una herramienta de mapeo objeto-relacional.

Servidor (MySQL):

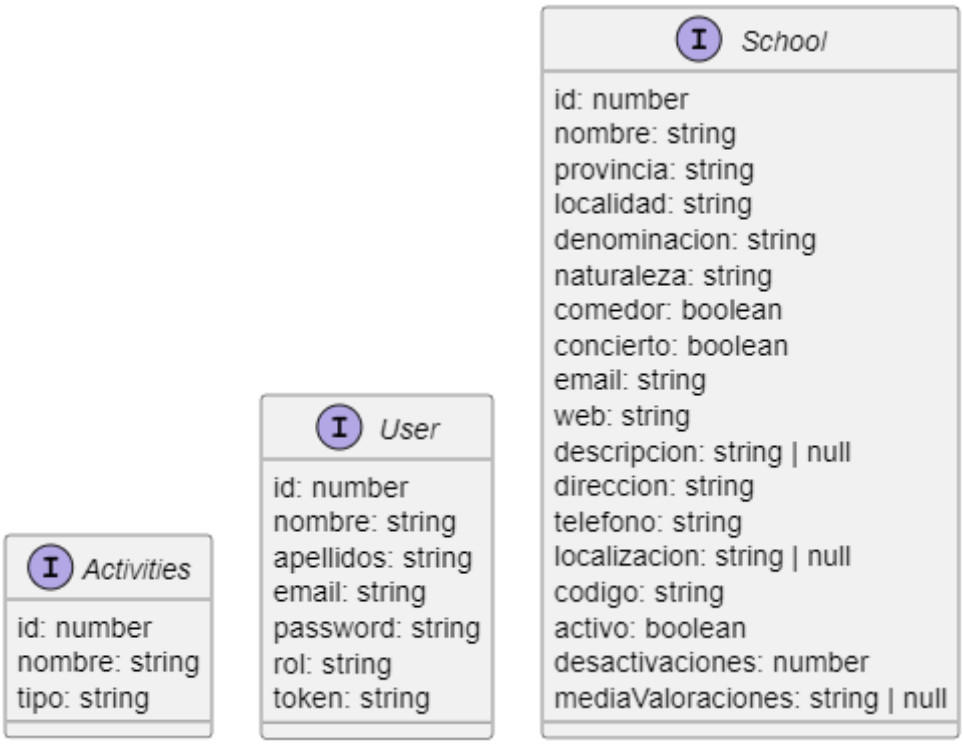
Este componente representa la base de datos MySQL utilizada por el servidor backend para almacenar datos.

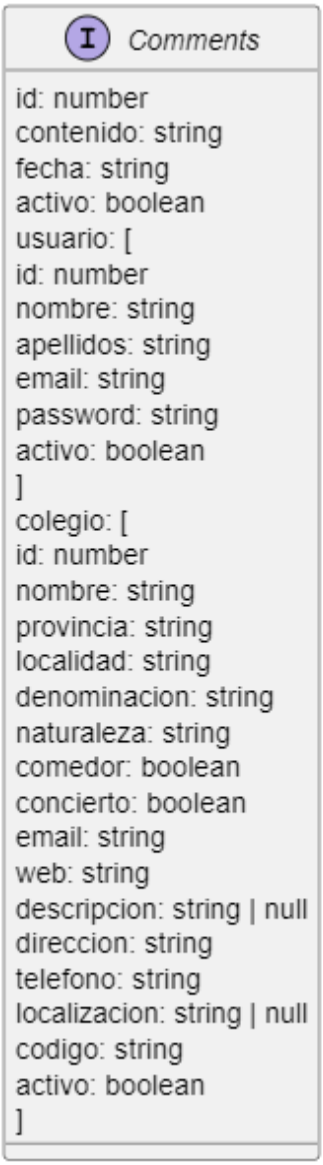
Servidor de correo (Resend):

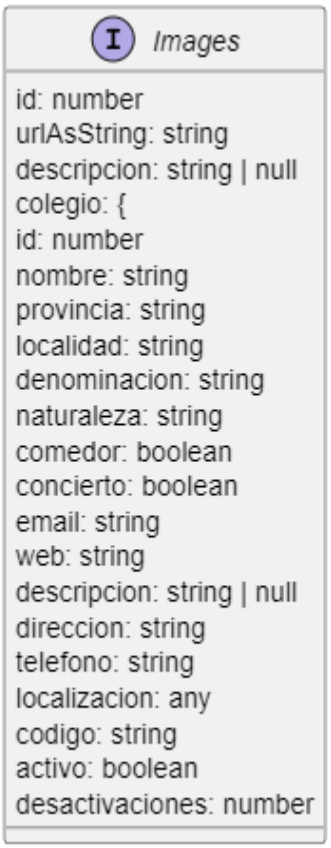
Este componente maneja las interacciones con un servidor de correo para el envío y recepción de correos electrónicos.

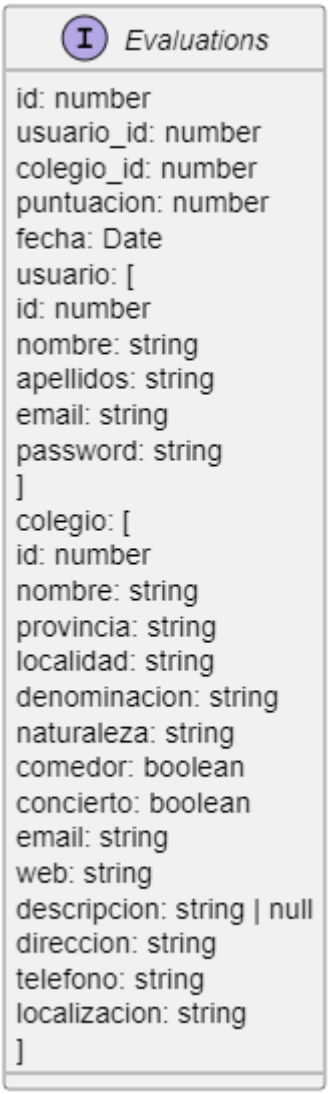
Está integrado con los controladores Express para manejar el envío de correos desde el servidor.

Diagramas de objetos:




Comments
id: number
contenido: string
fecha: string
activo: boolean
usuario: [
id: number
nombre: string
apellidos: string
email: string
password: string
activo: boolean
]
colegio: [
id: number
nombre: string
provincia: string
localidad: string
denominacion: string
naturaleza: string
comedor: boolean
concierto: boolean
email: string
web: string
descripcion: string | null
direccion: string
telefono: string
localizacion: string | null
codigo: string
activo: boolean
]


Images
id: number
urlAsString: string
descripcion: string | null
colegio: {
id: number
nombre: string
provincia: string
localidad: string
denominacion: string
naturaleza: string
comedor: boolean
concierto: boolean
email: string
web: string
descripcion: string | null
direccion: string
telefono: string
localizacion: any
codigo: string
activo: boolean
desactivaciones: number

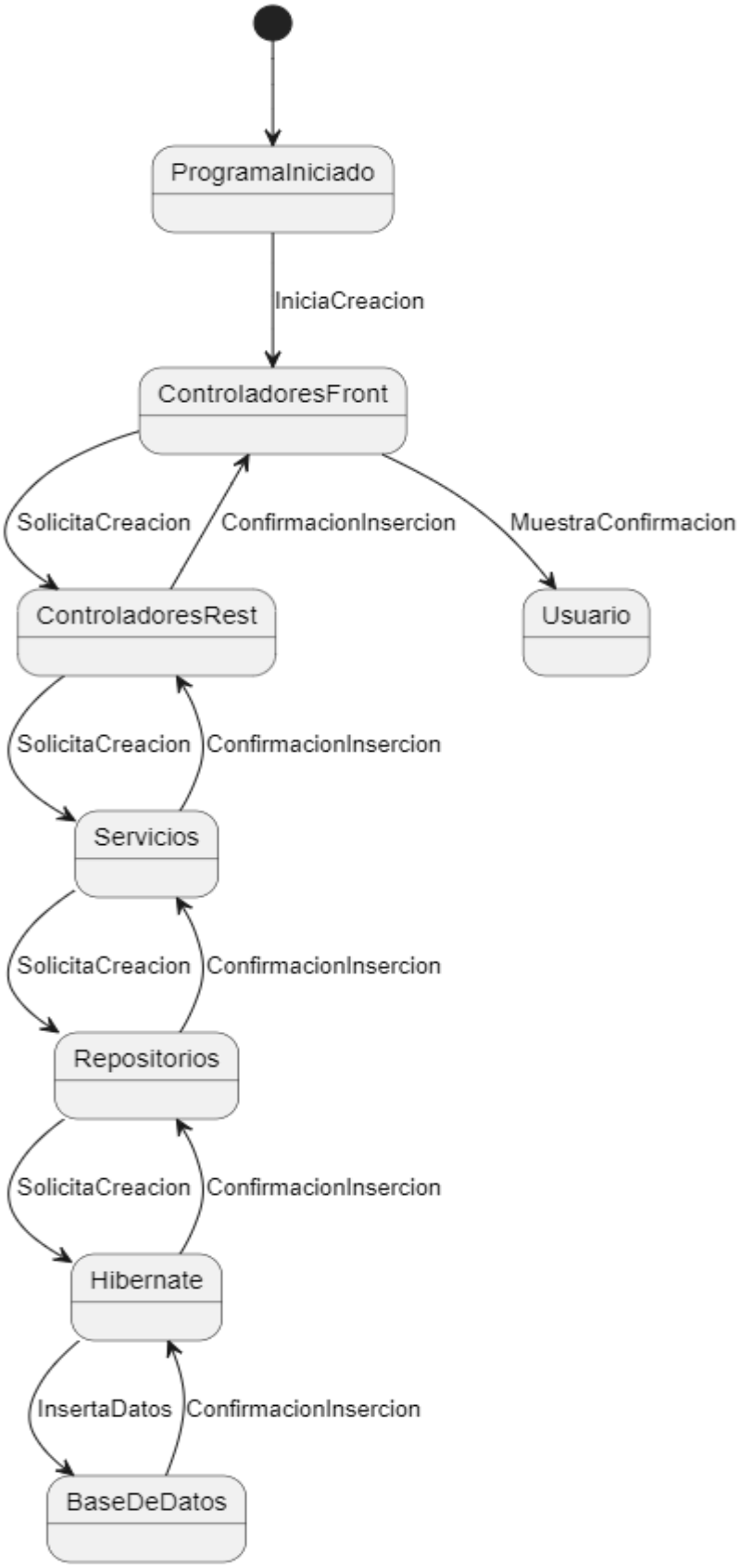


Estos son los objetos principales dentro de la parte de Angular usando el tipado que nos proporciona Typescript. En una aplicación Angular, estos tipos de datos definidos con TypeScript son fundamentales para representar la estructura y la forma de los datos que se manejan en la aplicación. Por ejemplo, el objeto User representa los datos de un usuario, como su nombre, apellidos, correo electrónico y rol. Estos objetos se utilizan en toda la aplicación Angular para mantener la coherencia de los datos y proporcionar una forma segura de trabajar con ellos. Además, estos tipos de datos son útiles para validar los datos recibidos del servidor y garantizar que cumplan con las expectativas de la aplicación

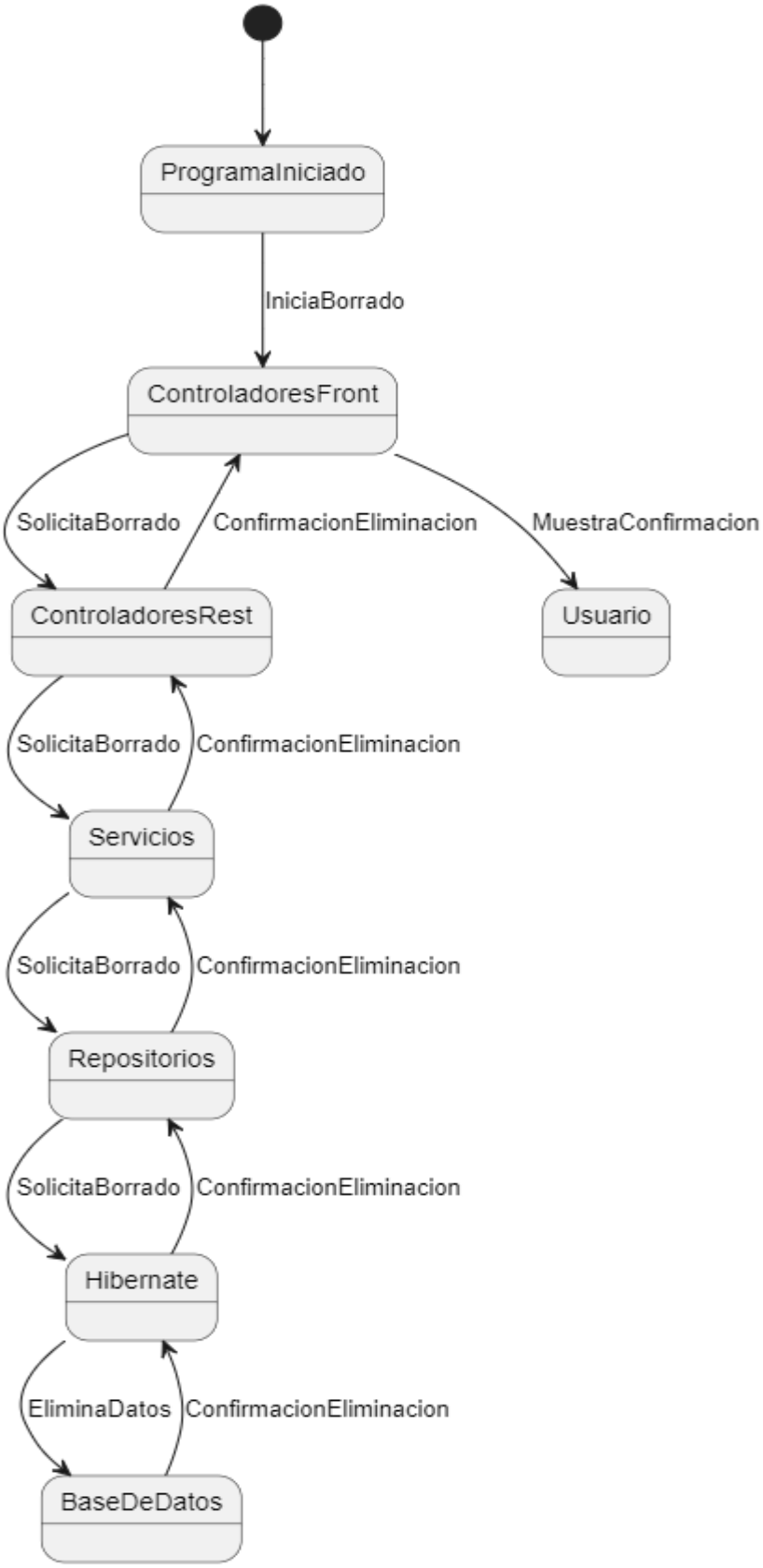
Diagramas de estados:

- **Inicio aplicación:**

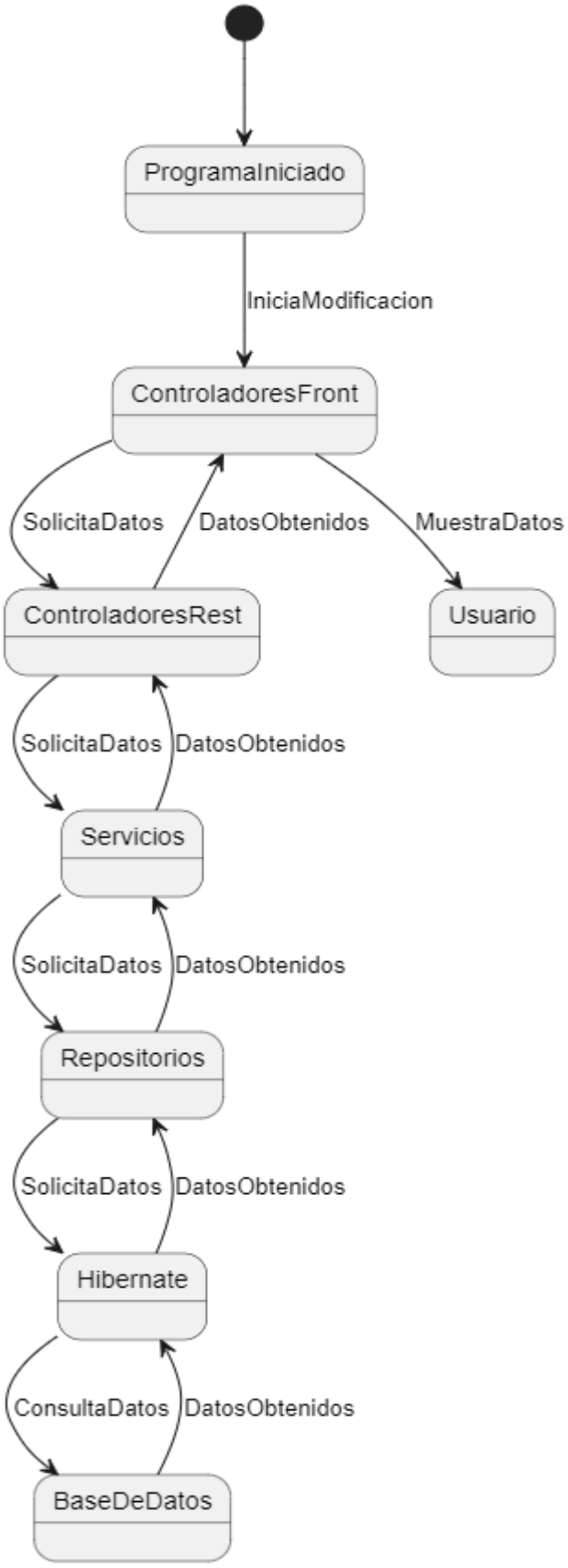
- **Crear:**


ProgramaIniciado
IniciaCreacion
ControladoresFront
SolicitaCreacion
ConfirmacionInsercion
MuestraConfirmacion
ControladoresRest
Usuario
SolicitaCreacion
ConfirmacionInsercion
Servicios
SolicitaCreacion
ConfirmacionInsercion
Repositorios
SolicitaCreacion
ConfirmacionInsercion
Hibernate
InsertaDatos
ConfirmacionInsercion
BaseDeDatos

- **Borrar**

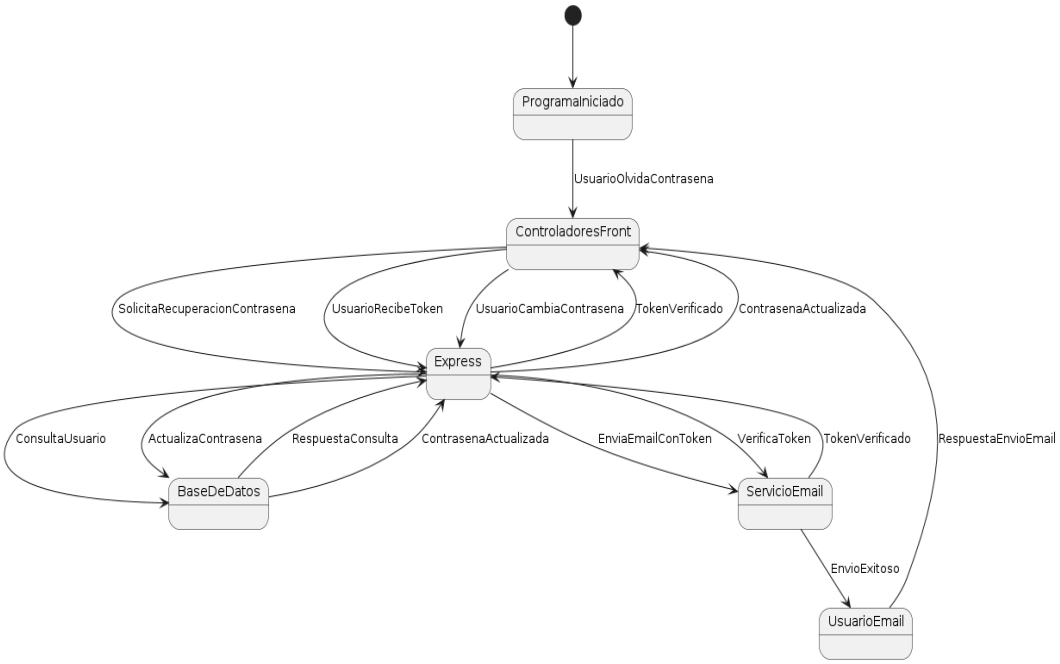

ProgramaIniciado
IniciaBorrado
ControladoresFront
SolicitaBorrado
ConfirmacionEliminacion
MuestraConfirmacion
ControladoresRest
Usuario
SolicitaBorrado
ConfirmacionEliminacion
Servicios
SolicitaBorrado
ConfirmacionEliminacion
Repositorios
SolicitaBorrado
ConfirmacionEliminacion
Hibernate
EliminaDatos
ConfirmacionEliminacion
BaseDeDatos

- **Modificar:**

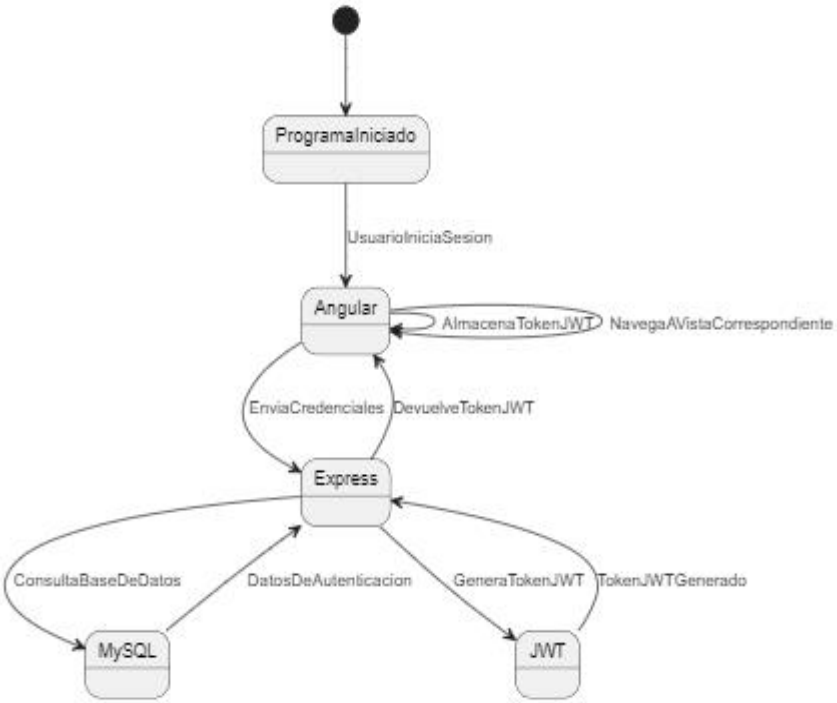

ProgramaIniciado
IniciaModificacion
ControladoresFront
SolicitaDatos
DatosObtenidos
MuestraDatos
ControladoresRest
Usuario
SolicitaDatos
DatosObtenidos
Servicios
SolicitaDatos
DatosObtenidos
Repositorios
SolicitaDatos
DatosObtenidos
Hibernate
ConsultaDatos
DatosObtenidos
BaseDeDatos

- **Crear colegio:**

- **Olvidar contraseña:**


ProgramaIniciado
UsuarioOlvidaContrasena
ControladoresFront
SolicitaRecuperacionContrasena
UsuarioRecibeToken
UsuarioCambiaContrasena
TokenVerificado
ContrasenaActualizada
Express
ConsultaUsuario
ActualizaContrasena
RespuestaConsulta
ContrasenaActualizada
EnviaEmailConToken
VerificaToken
TokenVerificado
RespuestaEnvioEmail
BaseDeDatos
ServicioEmail
EnvioExitoso
UsuarioEmail

- **Login:**


ProgramaIniciado
UsuarioIniciaSesion
Angular
AlmacenaTokenJWT
NavegaAVistaCorrespondiente
EnviaCredenciales
DevuelveTokenJWT
Express
ConsultaBaseDeDatos
DatosDeAutenticacion
GeneraTokenJWT
TokenJWTGenerado
MySQL
JWT

3.3. Interfaz

3.3.1Características generales

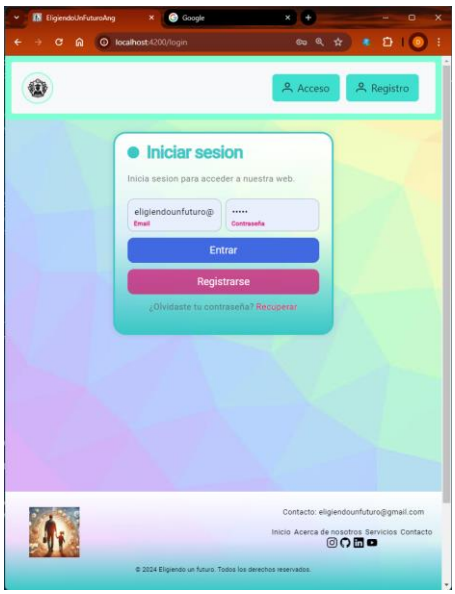
Es una aplicación con un diseño sencillo ya que lo que se busca es que no agobie que tenga armonía entre las páginas y que las personas que usen la aplicación se puedan centrar en el contenido y se sientan cómodas. Sin colores chillones ni demasiado recargada solo lo necesario para el objetivo que se busca que es conseguir mejor y más información para una de las decisiones más importantes ante las que nos encontramos los padres y las madres que es la elección del colegio o instituto que marcara el futuro de nuestros hijos en una parte muy importante.

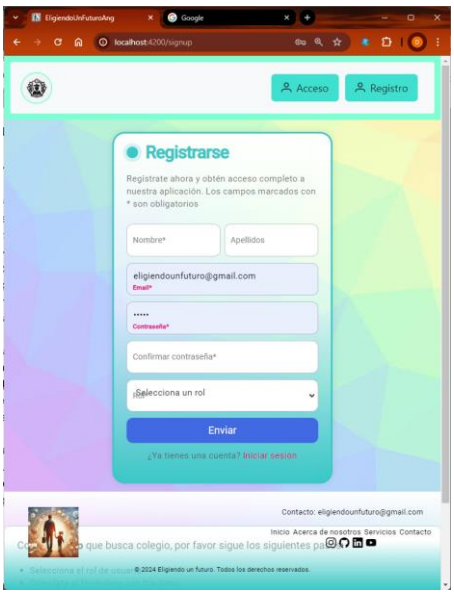
He usado la misma paleta de colores para todas las páginas como veremos a continuación:

Página principal

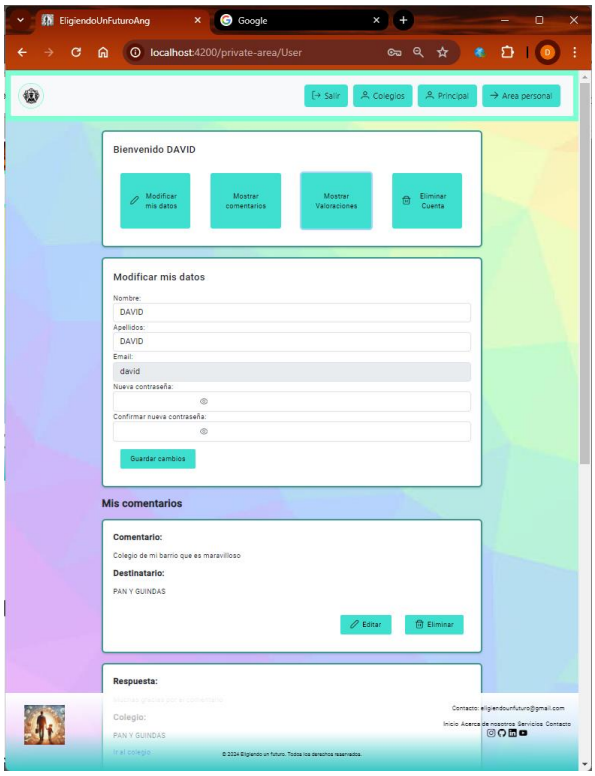
Página de colegios

Registro y login:

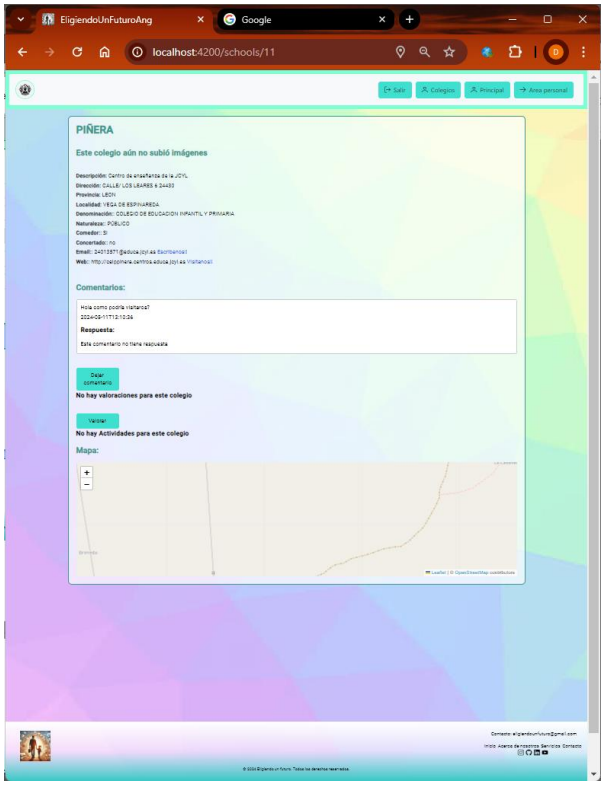




Área privada:



Detalle colegio:



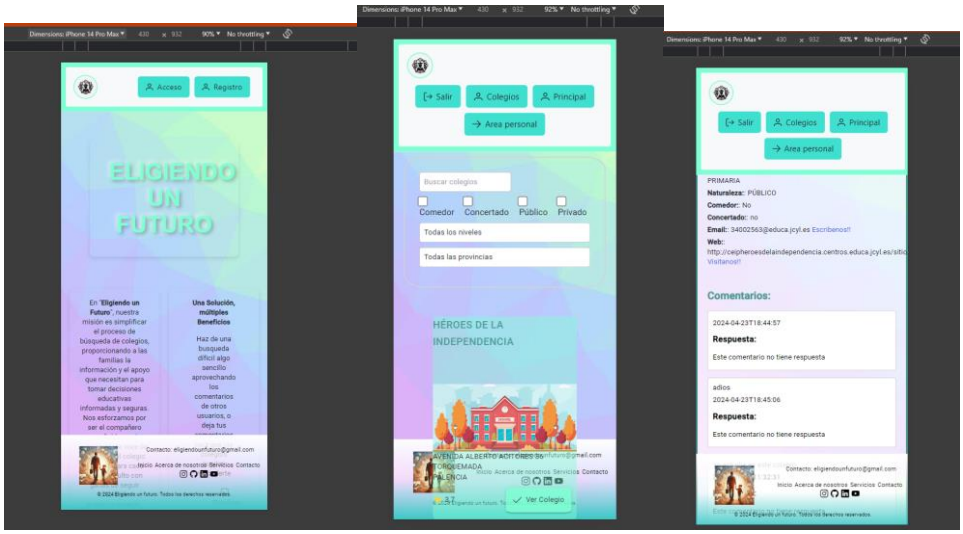
Detalle cuadros de díalogo:

3.3.2Adaptación a dispositivos móviles

El diseño es responsive para la adaptación a dispositivos móviles que en el momento en el que nos encontramos es tan frecuente su uso.

También por eso la elección de un diseño más simple y ligero ya que en los dispositivos móviles más aun la importancia de que tengamos muchísimas cosas en poco espacio.

Ejemplos en teléfono de 6.7 pulgadas:



3.3.3Usabilidad/Accesibilidad

Se a intentado hacer una aplicación fácil de usar e intuitiva para que sea fácilmente recordada por los usuarios y fácilmente adaptable.
Uno de los puntos de futuro es hacerla más accesible para diferentes tipos de usuario eso incluye usuarios con necesidades especiales.

Poder usar lectores de pantallas, descripciones para el contenido multimedia, etc.

Además, la aplicación está preparada para hacerla multi idioma.

4. Autoevaluación y conclusiones

4.1. Valoración del trabajo y dificultades encontradas

En general estoy contento con el trabajo realizado, aunque creo que si ahora mismo volviese a empezarle le haría mejor y más optimo con el código más limpio pero de esto se trata de aprender.
La principal dificultad con la que me encontré es Spring Boot Security. Me dio muchos problemas algo no estaba configurando bien y tuve que recurrir a Node para generar el token pero no he podido pasárselo en las peticiones precisamente por esta dependencia aunque tengo toda la lógica implementada con un interceptor que se encarga de añadir en el encabezado de las aplicaciones el token.
Luego también tuve problemas para la conexión con Apis externas debido a las CORS al no estar desplegada la aplicación lo interpretan como un ataque, debido a ello probe a conectarme desde Node ya que yo quería que solo se puedan registrar como colegios los colegios que estén registrados en la web de la junta ya que son los centros oficiales, pero la consulta era muy lenta ya que son más de 1600 resultados y la consulta se volvía extremadamente lenta porque no había una petición para conseguir todos los datos de una vez pero como sabemos la consulta entre una api o un JSON usando una HttpRequest no es muy diferente. Hice una prueba sacándolo de una base de datos con MongoDB, pero por miedo a pillarme de tiempo no lo implemente así que será una mejora a futuro.

4.2. Valoración de la herramienta o aplicación desarrollada

Creo que es una aplicación que contiene lo necesario para el objetivo que me había planteado que es la búsqueda de colegios el poder interactuar con estos y el poder ver las opiniones y las valoraciones de otros usuarios.

He podido implementar el borrado lógico para poder mantener los datos necesarios en nuestra base de datos para su estudio análisis y futuras implementaciones de estadísticas de datos.

También tiene una interacción creo que fácil entre los colegios y los usuarios.

Se pueden añadir imágenes dándole un extra tanto para los colegios poder enseñarlos mejor como los usuarios poder verlos.
También dispone del servicio de reenvío de contraseñas por email, avisa a los usuarios si al crearse como colegio no están en la base de datos de la junta por lo que no podemos añadirles y aviso al administrador de la petición de nuevos colegios además de poder implementarse a futuro en más opciones.

Se ha usado una buena arquitectura para hacerla más escalable y fácil de implementar nuevas opciones además de preservar los datos.

4.3. Conclusiones

En general he podido implementar tanto los conocimientos adquiridos en el curso como nuevos conocimientos que me han servido en mis prácticas y viceversa.

Pero tengo la sensación de que al final al hacerse en periodo de prácticas no puedes dar el cien por cien.

No he podido implementar todo lo que me hubiese gustado por falta de tiempo, ni dejarlo como me hubiese gustado pero en general la sensación es buena y además es una aplicación en la que seguiré trabajando ya que se me han ocurrido varias ideas de mejoras como interactuar entre usuarios, ofrecer más datos mejorar la localización con los mapas ya que ahora solo muestra la posición del colegio y la nuestra pero podríamos añadir la distancia que entre el colegio y el usuario. También el poder agendar visitas a los colegios y que los colegios puedan subir videos.