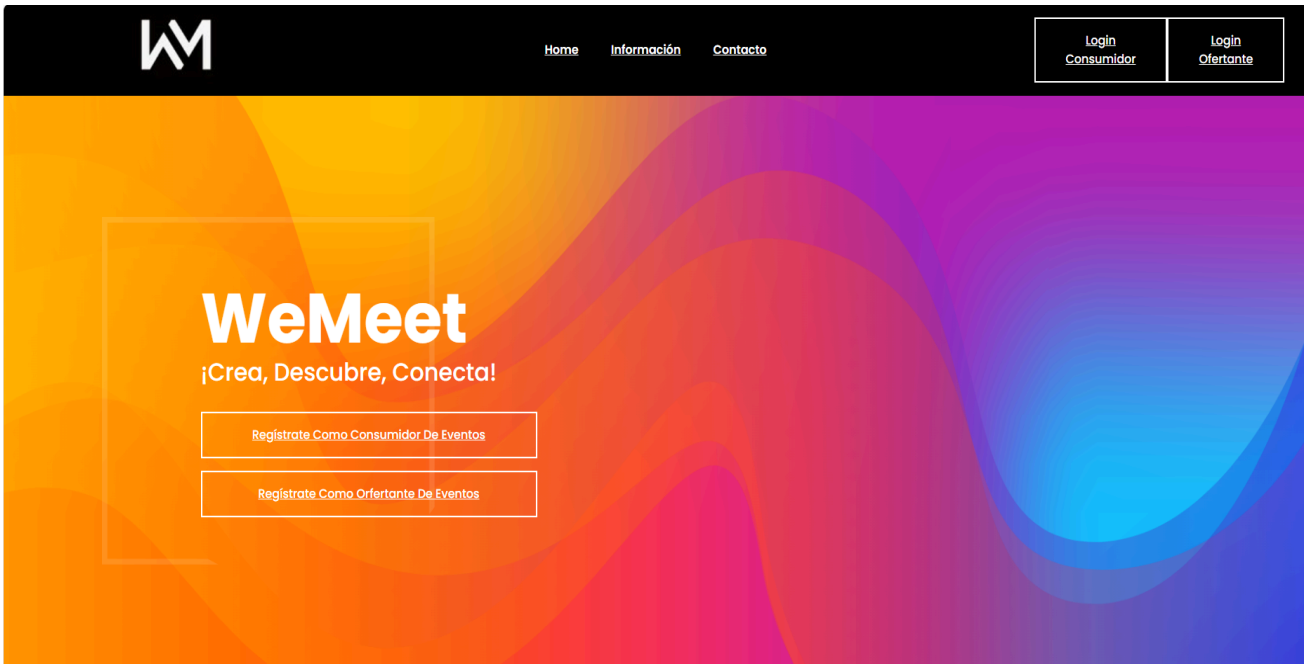
Proyecto Integrado DAW (2024)
I.E.S Velázquez



Autor: *Marco A. Blanco Lebrón*

Curso: *2023/2024*

IES Velázquez
Sevilla

Índice de contenidos

1. Resumen del Proyecto

- **Descripción:**

We Meet es una plataforma web innovadora que facilita la creación, gestión y participación en eventos sociales, conectando ofertantes y consumidores de actividades, similar al modelo de economía colaborativa de otras apps para viajes compartidos. La app permite a los usuarios ofrecer actividades diversas, desde clases de cocina hasta excursiones de fin de semana, y a otros usuarios encontrar y participar en estas actividades. WeMeet está diseñada tanto para aquellos que desean compartir sus habilidades y pasiones a través de actividades organizadas, como para los que simplemente buscan experiencias enriquecedoras y nuevas oportunidades de socialización.

- **Tecnologías Utilizadas:**

◦ Frontend:
▪ HTML5
▪ CSS3
▪ Angular
▪ Bootstrap para el diseño responsivo

◦ Backend:
▪ Node.js
▪ Base de datos MariaDB
▪ Spring Boot (Maven)

◦ Herramientas de desarrollo:

- Git y GitHub para el control de versiones
- Intellij IDEA como IDE
- Visual Studio Code como editor de código.

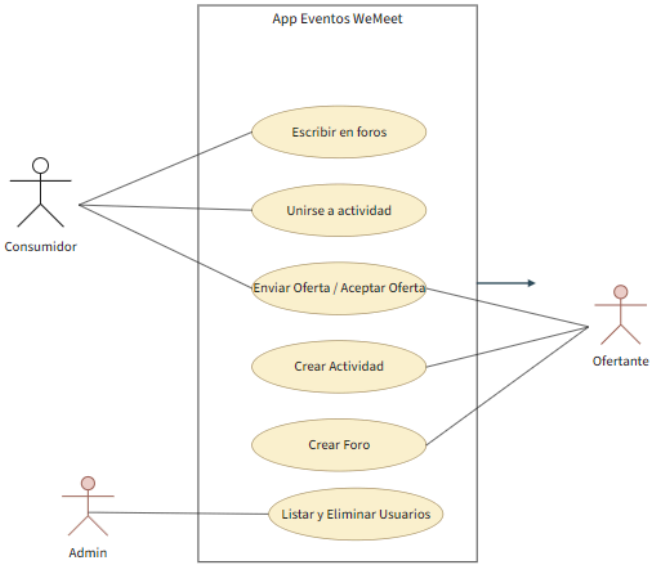
Para la creación de esta app de eventos sociales que pone en contacto a consumidores y ofertantes se ha utilizado como tecnologías de BackEnd el framework Spring con Hibernate para la creación de la base de datos. Con Spring, podemos implementar servicios RESTful para exponer funcionalidades esenciales, como la creación y gestión de eventos, la autenticación de usuarios y la recuperación de datos relevantes. Además, Spring Security nos permite garantizar la protección de los recursos y la autenticación de usuarios con el uso extra de Json Web Tokens, proporcionando un entorno seguro y confiable para los usuarios.

Para el caso de la interfaz de usuario se ha utilizado el framework de FrontEnd Angular y clases de Bootstrap en el html de los componentes para el diseño de css predefinido. Esta tecnología permite crear una interfaz de usuario interactiva e intuitiva de Single Page-Application que permite a los usuarios explorar, registrarse y participar en eventos sociales de manera eficiente.

2. Arquitectura del Sistema

Diagrama UML del proyecto y Flujo

UML Casos de Uso



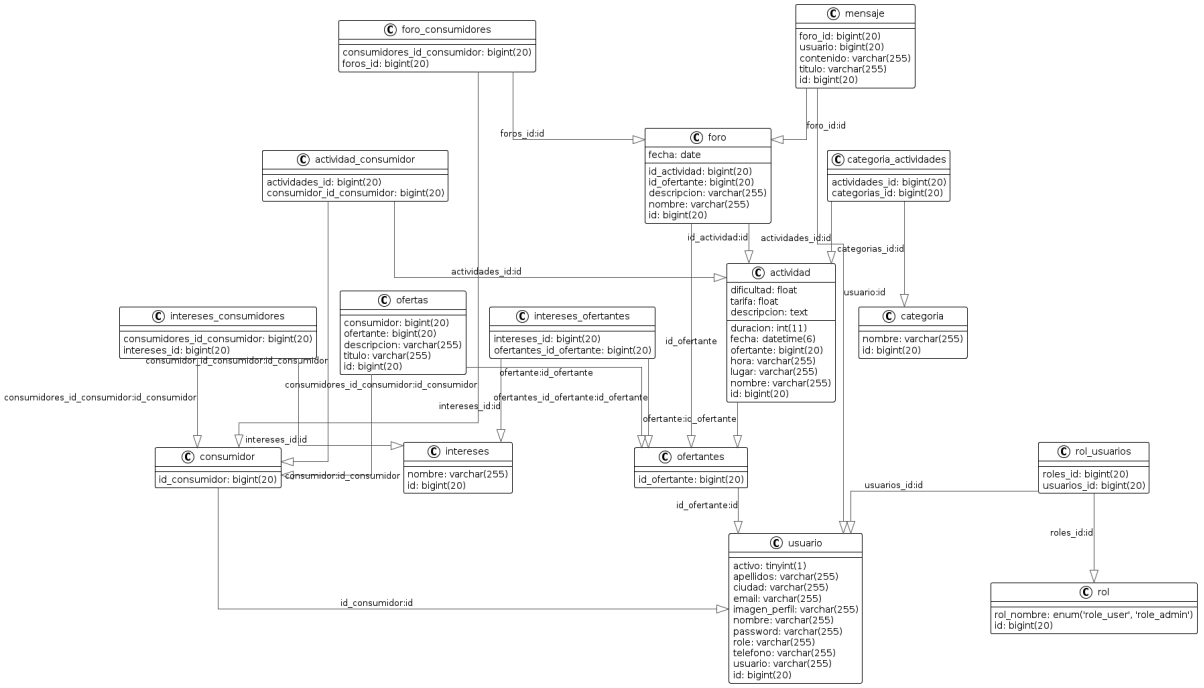
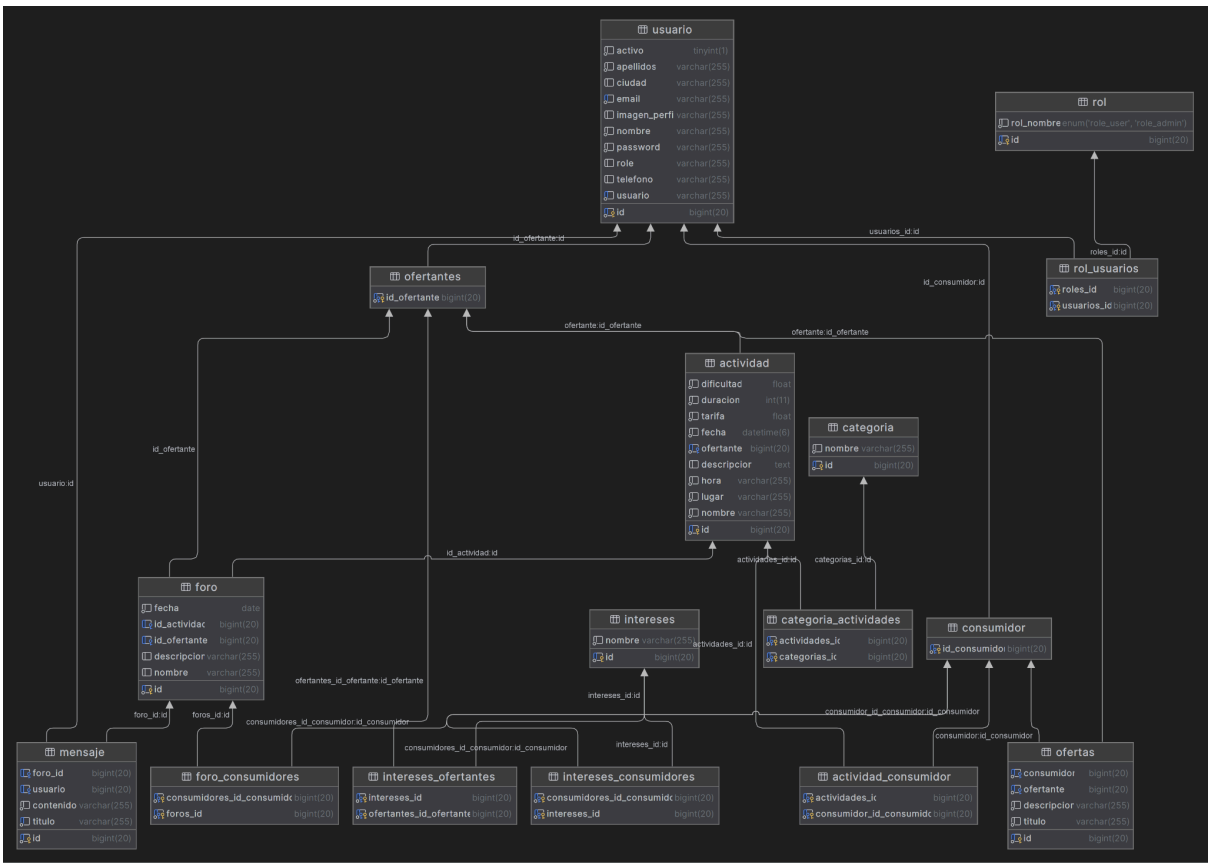


Diagrama Entidad-Relación



Base de Datos

La estructura interna de la base de datos, sus tablas y relaciones están dadas por el siguiente script en sql (MariaDB):

```
CREATE TABLE actividad (
    dificultad FLOAT NOT NULL,
    duracion INT NOT NULL,
    tarifa FLOAT NOT NULL,
    fecha DATETIME NOT NULL,
    id BIGINT NOT NULL AUTO_INCREMENT,
    ofertante BIGINT NOT NULL,
    descripcion TEXT,
    hora VARCHAR(255) NOT NULL,
    lugar VARCHAR(255) NOT NULL,
    nombre VARCHAR(255) NOT NULL,
    PRIMARY KEY (id)
);
```

```
CREATE TABLE actividad_consumidor (
    actividades_id BIGINT NOT NULL,
    consumidor_id_consumidor BIGINT NOT NULL,
    PRIMARY KEY (actividades_id, consumidor_id_consumidor)
);

CREATE TABLE categoria (
    id BIGINT NOT NULL AUTO_INCREMENT,
    nombre VARCHAR(255) NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE categoria_actividades (
    actividades_id BIGINT NOT NULL,
    categorias_id BIGINT NOT NULL,
    PRIMARY KEY (actividades_id, categorias_id)
);

CREATE TABLE consumidor (
    id_consumidor BIGINT NOT NULL,
    PRIMARY KEY (id_consumidor)
);

CREATE TABLE foro (
    fecha DATE,
    id BIGINT NOT NULL AUTO_INCREMENT,
    id_actividad BIGINT,
    id_ofertante BIGINT,
    descripcion VARCHAR(255),
    nombre VARCHAR(255),
    PRIMARY KEY (id),
    UNIQUE KEY UK_2hpp2ok4haoa33i0wx4hp2gxt (id_actividad),
    UNIQUE KEY UK_pu0q4wqnp6oa4cccbu9u45xhu (id_ofertante)
);

CREATE TABLE foro_consumidores (
    consumidores_id_consumidor BIGINT NOT NULL,
    foros_id BIGINT NOT NULL,
    PRIMARY KEY (consumidores_id_consumidor, foros_id)
);

CREATE TABLE intereses (
    id BIGINT NOT NULL AUTO_INCREMENT,
    nombre VARCHAR(255) NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE intereses_consumidores (
```

```
    consumidores_id_consumidor BIGINT NOT NULL,
    intereses_id BIGINT NOT NULL,
    PRIMARY KEY (consumidores_id_consumidor, intereses_id)
);

CREATE TABLE intereses_ofertantes (
    intereses_id BIGINT NOT NULL,
    ofertantes_id_ofertante BIGINT NOT NULL,
    PRIMARY KEY (intereses_id, ofertantes_id_ofertante)
);

CREATE TABLE mensaje (
    foro_id BIGINT,
    id BIGINT NOT NULL AUTO_INCREMENT,
    usuario BIGINT,
    contenido VARCHAR(255) NOT NULL,
    titulo VARCHAR(255),
    PRIMARY KEY (id),
    FOREIGN KEY (foro_id) REFERENCES foro (id),
    FOREIGN KEY (usuario) REFERENCES usuario (id)
);

CREATE TABLE ofertantes (
    id_ofertante BIGINT NOT NULL,
    PRIMARY KEY (id_ofertante)
);

CREATE TABLE ofertas (
    consumidor BIGINT NOT NULL,
    id BIGINT NOT NULL AUTO_INCREMENT,
    ofertante BIGINT NOT NULL,
    descripcion VARCHAR(255) NOT NULL,
    titulo VARCHAR(255) NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (consumidor) REFERENCES consumidor (id_consumidor),
    FOREIGN KEY (ofertante) REFERENCES ofertantes (id_ofertante)
);

CREATE TABLE rol (
    id BIGINT NOT NULL AUTO_INCREMENT,
    rol_nombre ENUM('ROLE_USER', 'ROLE_ADMIN') NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE rol_usuarios (
    roles_id BIGINT NOT NULL,
    usuarios_id BIGINT NOT NULL,
    PRIMARY KEY (roles_id, usuarios_id),
```

```
    FOREIGN KEY (usuarios_id) REFERENCES usuario (id),
    FOREIGN KEY (roles_id) REFERENCES rol (id)
);

CREATE TABLE usuario (
    activo BOOLEAN NOT NULL,
    id BIGINT NOT NULL AUTO_INCREMENT,
    apellidos VARCHAR(255) NOT NULL,
    ciudad VARCHAR(255),
    email VARCHAR(255) NOT NULL,
    imagen_perfil VARCHAR(255),
    nombre VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    role VARCHAR(255),
    telefono VARCHAR(255),
    usuario VARCHAR(255) NOT NULL,
    PRIMARY KEY (id),
    UNIQUE KEY UK_5171l57faosmj8myawaucatdw (email),
    UNIQUE KEY UK_i02kr8ui5pqddyd7pkm3v4jbt (usuario)
);

ALTER TABLE foro ADD CONSTRAINT FK_2hpp2ok4haoa33i0wx4hp2gxt FOREIGN KEY
(id_actividad) REFERENCES actividad (id);
ALTER TABLE foro ADD CONSTRAINT FK_pu0q4wqnp6oa4cccbu9u45xhu FOREIGN KEY
(id_ofertante) REFERENCES ofertantes (id_ofertante);
ALTER TABLE usuario ADD CONSTRAINT FK_5171l57faosmj8myawaucatdw FOREIGN
KEY (email) REFERENCES usuario (id);
ALTER TABLE usuario ADD CONSTRAINT FK_i02kr8ui5pqddyd7pkm3v4jbt FOREIGN KEY
(usuario) REFERENCES usuario (id);
ALTER TABLE actividad ADD CONSTRAINT FK_8a3rvfoq1o026pm79y93vne16 FOREIGN
KEY (ofertante) REFERENCES ofertantes (id_ofertante);
ALTER TABLE actividad_consumidor ADD CONSTRAINT
FK_qhb31y3w9j2mdmdu0sigkw9xq FOREIGN KEY (consumidor_id_consumidor)
REFERENCES consumidor (id_consumidor);
ALTER TABLE actividad_consumidor ADD CONSTRAINT FK_gwqx83dht1pathruj72gc1u1v
FOREIGN KEY (actividades_id) REFERENCES actividad (id);
ALTER TABLE categoria_actividades ADD CONSTRAINT FK_g71dnl8oxxjfaqaho5j9esbss
FOREIGN KEY (actividades_id) REFERENCES actividad (id);
ALTER TABLE categoria_actividades ADD CONSTRAINT
FK_4f1y5ymwebjwoqvawc6omicm3 FOREIGN KEY (categorias_id) REFERENCES
categoria (id);
ALTER TABLE consumidor ADD CONSTRAINT FK_aomcp22eap9toykxctvaan2fo
FOREIGN KEY (id_consumidor) REFERENCES usuario (id);
ALTER TABLE foro ADD CONSTRAINT FK_las1y392h3bh6xehwp9ysran FOREIGN KEY
(id_actividad) REFERENCES actividad (id);
ALTER TABLE foro ADD CONSTRAINT FK_hnw5fmk3tyntxxv4cml119l8h FOREIGN KEY
(id_ofertante) REFERENCES ofertantes (id_ofertante);
```

```
ALTER TABLE foro_consumidores ADD CONSTRAINT FK_ethymg70yh064fb4whf5c9o85
FOREIGN KEY (consumidores_id_consumidor) REFERENCES consumidor
(id_consumidor);
ALTER TABLE foro_consumidores ADD CONSTRAINT FK_qoh1hc2cltwiy0omdmbl3v7mg
FOREIGN KEY (foros_id) REFERENCES foro (id);
ALTER TABLE intereses_consumidores ADD CONSTRAINT
FK_sy77wjsldkynks5vqqy7uy4u2 FOREIGN KEY (consumidores_id_consumidor)
REFERENCES consumidor (id_consumidor);
ALTER TABLE intereses_consumidores ADD CONSTRAINT
FK_28hx79ral1c0mry4rr486cgy0 FOREIGN KEY (intereses_id) REFERENCES intereses
(id);
ALTER TABLE intereses_ofertantes ADD CONSTRAINT FK_5wmsa6d0qq7e2aeyihpf57kyq
FOREIGN KEY (ofertantes_id_ofertante) REFERENCES ofertantes (id_ofertante);
ALTER TABLE intereses_ofertantes ADD CONSTRAINT FK_q9sebckroyjsdqv9cfh8owgla
FOREIGN KEY (intereses_id) REFERENCES intereses (id);
ALTER TABLE mensaje ADD CONSTRAINT FK_npn2p2asveqjadyec0r0bp0ym FOREIGN
KEY (foro_id) REFERENCES foro (id);
ALTER TABLE mensaje ADD CONSTRAINT FK_16b741pl8vmflu3s99ih92etr FOREIGN
KEY (usuario) REFERENCES usuario (id);
ALTER TABLE ofertantes ADD CONSTRAINT FK_1jt5hi7e32so5dqnar373av7l FOREIGN
KEY (id_ofertante) REFERENCES usuario (id);
ALTER TABLE ofertas ADD CONSTRAINT FK_o1sy76y9an5cau896l6wxv8f6 FOREIGN
KEY (consumidor) REFERENCES consumidor (id_consumidor);
ALTER TABLE ofertas ADD CONSTRAINT FK_jvrh9jpv7hjxbm2uahl2boird FOREIGN KEY
(ofertante) REFERENCES ofertantes (id_ofertante);
ALTER TABLE rol_usuarios ADD CONSTRAINT FK_v87dibmskom8m81onec3u3hv
FOREIGN KEY (usuarios_id) REFERENCES usuario (id);
ALTER TABLE rol_usuarios ADD CONSTRAINT FK_7md72xqhxlgdsvphe4eqo7utt
FOREIGN KEY (roles_id) REFERENCES rol (id);
```

```
INSERT INTO actividad (dificultad, duracion, tarifa, fecha, ofertante, descripcion, hora, lugar,
nombre)
VALUES (3.5, 120, 25.50, '2024-05-06 08:00:00', 1, 'Actividad de senderismo en las
montañas', '08:00', 'Montañas de ejemplo', 'Senderismo en las Montañas');

INSERT INTO actividad (dificultad, duracion, tarifa, fecha, ofertante, descripcion, hora, lugar,
nombre)
VALUES (2.0, 90, 15.75, '2024-05-07 10:30:00', 2, 'Clase de cocina italiana', '10:30', 'Calle
Principal, Ciudad', 'Clase de Cocina Italiana');

INSERT INTO actividad (dificultad, duracion, tarifa, fecha, ofertante, descripcion, hora, lugar,
nombre)
VALUES (4.0, 180, 35.00, '2024-05-08 18:00:00', 3, 'Concierto de jazz en vivo', '18:00',
'Teatro Municipal', 'Concierto de Jazz en Vivo');
```

```
INSERT INTO categoria (nombre) VALUES ('Deportes');
INSERT INTO categoria (nombre) VALUES ('Cocina');
INSERT INTO categoria (nombre) VALUES ('Música');
INSERT INTO categoria (nombre) VALUES ('Deportes Acuáticos');
INSERT INTO categoria (nombre) VALUES ('Postres');
INSERT INTO categoria (nombre) VALUES ('Clásica');
INSERT INTO categoria (nombre) VALUES ('Pintura');
INSERT INTO categoria (nombre) VALUES ('Aventura');
INSERT INTO categoria (nombre) VALUES ('Danza');
```

```
INSERT INTO intereses (nombre) VALUES ('Senderismo');
INSERT INTO intereses (nombre) VALUES ('Cocina Italiana');
INSERT INTO intereses (nombre) VALUES ('Jazz');
INSERT INTO intereses (nombre) VALUES ('Fútbol');
INSERT INTO intereses (nombre) VALUES ('Gastronomía Asiática');
INSERT INTO intereses (nombre) VALUES ('Rock');
INSERT INTO intereses (nombre) VALUES ('Arte');
INSERT INTO intereses (nombre) VALUES ('Viajes');
INSERT INTO intereses (nombre) VALUES ('Baile');
```

Descripción de las Tablas y sus Relaciones

Tabla: `actividad`

- **Descripción:** Representa las actividades ofrecidas por los ofertantes.
- **Columnas:**
  - `dificultad`: Nivel de dificultad de la actividad (FLOAT).
  - `duracion`: Duración de la actividad en minutos (INT).
  - `tarifa`: Costo de la actividad (FLOAT).
  - `fecha`: Fecha y hora de la actividad (DATETIME).
  - `id`: Identificador único de la actividad (BIGINT, AUTO_INCREMENT).
  - `ofertante`: Identificador del ofertante que organiza la actividad (BIGINT).
  - `descripcion`: Descripción de la actividad (TEXT).

    - `hora`: Hora de inicio de la actividad (VARCHAR(255)).
    - `lugar`: Lugar donde se realizará la actividad (VARCHAR(255)).
    - `nombre`: Nombre de la actividad (VARCHAR(255)).
- **Llaves Primarias y Foráneas:**
    - `PRIMARY KEY (id)`
    - `FOREIGN KEY (ofertante) REFERENCES ofertantes (id_ofertante)`

Tabla: `actividad_consumidor`

- **Descripción:** Representa la relación entre las actividades y los consumidores que participan en ellas.
- **Columnas:**
    - `actividades_id`: Identificador de la actividad (BIGINT).
    - `consumidor_id_consumidor`: Identificador del consumidor (BIGINT).
- **Llaves Primarias y Foráneas:**
    - `PRIMARY KEY (actividades_id, consumidor_id_consumidor)`
    - `FOREIGN KEY (consumidor_id_consumidor) REFERENCES consumidor (id_consumidor)`
    - `FOREIGN KEY (actividades_id) REFERENCES actividad (id)`

Tabla: `categoria`

- **Descripción:** Representa las categorías de las actividades.
- **Columnas:**
    - `id`: Identificador único de la categoría (BIGINT, AUTO_INCREMENT).
    - `nombre`: Nombre de la categoría (VARCHAR(255)).
- **Llaves Primarias:**
    - `PRIMARY KEY (id)`

Tabla: `categoria_actividades`

- **Descripción:** Representa la relación entre categorías y actividades.
- **Columnas:**
    - `actividades_id`: Identificador de la actividad (BIGINT).
    - `categorias_id`: Identificador de la categoría (BIGINT).
- **Llaves Primarias y Foráneas:**
    - `PRIMARY KEY (actividades_id, categorias_id)`

    - `FOREIGN KEY (actividades_id) REFERENCES actividad (id)`
    - `FOREIGN KEY (categorias_id) REFERENCES categoria (id)`

Tabla: `consumidor`

- **Descripción:** Representa los consumidores que participan en las actividades.
- **Columnas:**
    - `id_consumidor`: Identificador único del consumidor (BIGINT).
- **Llaves Primarias y Foráneas:**
    - `PRIMARY KEY (id_consumidor)`
    - `FOREIGN KEY (id_consumidor) REFERENCES usuario (id)`

Tabla: `foro`

- **Descripción:** Representa los foros asociados a actividades y ofertantes.
- **Columnas:**
    - `fecha`: Fecha de creación del foro (DATE).
    - `id`: Identificador único del foro (BIGINT, AUTO_INCREMENT).
    - `id_actividad`: Identificador de la actividad asociada (BIGINT).
    - `id_ofertante`: Identificador del ofertante asociado (BIGINT).
    - `descripcion`: Descripción del foro (VARCHAR(255)).
    - `nombre`: Nombre del foro (VARCHAR(255)).
- **Llaves Primarias y Foráneas:**
    - `PRIMARY KEY (id)`
    - `UNIQUE KEY (id_actividad)`
    - `UNIQUE KEY (id_ofertante)`
    - `FOREIGN KEY (id_actividad) REFERENCES actividad (id)`
    - `FOREIGN KEY (id_ofertante) REFERENCES ofertantes (id_ofertante)`

Tabla: `foro_consumidores`

- **Descripción:** Representa la relación entre foros y consumidores.
- **Columnas:**
    - `consumidores_id_consumidor`: Identificador del consumidor (BIGINT).
    - `foros_id`: Identificador del foro (BIGINT).
- **Llaves Primarias y Foráneas:**
    - `PRIMARY KEY (consumidores_id_consumidor, foros_id)`

    - `FOREIGN KEY (consumidores_id_consumidor) REFERENCES consumidor (id_consumidor)`
    - `FOREIGN KEY (foros_id) REFERENCES foro (id)`

Tabla: `intereses`

- **Descripción:** Representa los intereses de los usuarios.
- **Columnas:**
    - `id`: Identificador único del interés (BIGINT, AUTO_INCREMENT).
    - `nombre`: Nombre del interés (VARCHAR(255)).
- **Llaves Primarias:**
    - `PRIMARY KEY (id)`

Tabla: `intereses_consumidores`

- **Descripción:** Representa la relación entre intereses y consumidores.
- **Columnas:**
    - `consumidores_id_consumidor`: Identificador del consumidor (BIGINT).
    - `intereses_id`: Identificador del interés (BIGINT).
- **Llaves Primarias y Foráneas:**
    - `PRIMARY KEY (consumidores_id_consumidor, intereses_id)`
    - `FOREIGN KEY (consumidores_id_consumidor) REFERENCES consumidor (id_consumidor)`
    - `FOREIGN KEY (intereses_id) REFERENCES intereses (id)`

Tabla: `intereses_ofertantes`

- **Descripción:** Representa la relación entre intereses y ofertantes.
- **Columnas:**
    - `intereses_id`: Identificador del interés (BIGINT).
    - `ofertantes_id_ofertante`: Identificador del ofertante (BIGINT).
- **Llaves Primarias y Foráneas:**
    - `PRIMARY KEY (intereses_id, ofertantes_id_ofertante)`
    - `FOREIGN KEY (ofertantes_id_ofertante) REFERENCES ofertantes (id_ofertante)`
    - `FOREIGN KEY (intereses_id) REFERENCES intereses (id)`

Tabla: `mensaje`

- **Descripción:** Representa los mensajes enviados por los usuarios en los foros.

- **Columnas:**
  - `foro_id`: Identificador del foro asociado (BIGINT).
  - `id`: Identificador único del mensaje (BIGINT, AUTO_INCREMENT).
  - `usuario`: Identificador del usuario que envió el mensaje (BIGINT).
  - `contenido`: Contenido del mensaje (VARCHAR(255)).
  - `titulo`: Título del mensaje (VARCHAR(255)).
- **Llaves Primarias y Foráneas:**
  - `PRIMARY KEY (id)`
  - `FOREIGN KEY (foro_id) REFERENCES foro (id)`
  - `FOREIGN KEY (usuario) REFERENCES usuario (id)`

Tabla: `ofertantes`

- **Descripción:** Representa los ofertantes que crean y ofrecen actividades.
- **Columnas:**
  - `id_ofertante`: Identificador único del ofertante (BIGINT).
- **Llaves Primarias y Foráneas:**
  - `PRIMARY KEY (id_ofertante)`
  - `FOREIGN KEY (id_ofertante) REFERENCES usuario (id)`

Tabla: `ofertas`

- **Descripción:** Representa las ofertas creadas por los ofertantes y consumidas por los consumidores.
- **Columnas:**
  - `consumidor`: Identificador del consumidor asociado (BIGINT).
  - `id`: Identificador único de la oferta (BIGINT, AUTO_INCREMENT).
  - `ofertante`: Identificador del ofertante asociado (BIGINT).
  - `descripcion`: Descripción de la oferta (VARCHAR(255)).
  - `titulo`: Título de la oferta (VARCHAR(255)).
- **Llaves Primarias y Foráneas:**
  - `PRIMARY KEY (id)`
  - `FOREIGN KEY (consumidor) REFERENCES consumidor (id_consumidor)`
  - `FOREIGN KEY (ofertante) REFERENCES ofertantes (id_ofertante)`

Tabla: `rol`

- **Descripción:** Representa los roles de los usuarios (por ejemplo, administrador, usuario).

- **Columnas:**
  - `id`: Identificador único del rol (BIGINT, AUTO_INCREMENT).
  - `rol_nombre`: Nombre del rol (ENUM con valores 'ROLE_USER' y 'ROLE_ADMIN').
- **Llaves Primarias:**
  - `PRIMARY KEY (id)`

Tabla: `rol_usuarios`

- **Descripción:** Representa la relación entre roles y usuarios.
- **Columnas:**
  - `roles_id`: Identificador del rol (BIGINT).
  - `usuarios_id`: Identificador del usuario (BIGINT).
- **Llaves Primarias y Foráneas:**
  - `PRIMARY KEY (roles_id, usuarios_id)`
  - `FOREIGN KEY (usuarios_id) REFERENCES usuario (id)`
  - `FOREIGN KEY (roles_id) REFERENCES rol (id)`

Tabla: `usuario`

- **Descripción:** Representa los usuarios de la plataforma, incluyendo tanto consumidores como ofertantes.
- **Columnas:**
  - `activo`: Estado activo del usuario (BOOLEAN).
  - `id`: Identificador único del usuario (BIGINT, AUTO_INCREMENT).
  - `apellidos`: Apellidos del usuario (VARCHAR(255)).
  - `ciudad`: Ciudad del usuario (VARCHAR(255)).
  - `email`: Correo electrónico del usuario (VARCHAR(255)).
  - `imagen_perfil`: URL de la imagen de perfil del usuario (VARCHAR(255)).
  - `nombre`: Nombre del usuario (VARCHAR(255)).
  - `password`: Contraseña del usuario (VARCHAR(255)).
  - `role`: Rol del usuario (VARCHAR(255)).
  - `telefono`: Teléfono del usuario (VARCHAR(255)).
  - `usuario`: Nombre de usuario (VARCHAR(255)).
- **Llaves Primarias y Foráneas:**
  - `PRIMARY KEY (id)`
  - `UNIQUE KEY (email)`
  - `UNIQUE KEY (usuario)`
  - `FOREIGN KEY (email) REFERENCES usuario (id)`

    - `FOREIGN KEY (usuario) REFERENCES usuario (id)`

Relaciones Entre las Tablas

- **Usuarios y Roles:** Un usuario puede tener múltiples roles (muchos a muchos).
- **Actividades y Categorías:** Una actividad puede pertenecer a múltiples categorías y una categoría puede tener múltiples actividades (muchos a muchos).
- **Actividades y Consumidores:** Un consumidor puede participar en múltiples actividades y una actividad puede tener múltiples consumidores (muchos a muchos).
- **Foros y Actividades:** Un foro está asociado a una actividad (uno a uno).
- **Foros y Ofertantes:** Un foro está asociado a un ofertante (uno a uno).
- **Foros y Consumidores:** Un foro puede tener múltiples consumidores participando en él (muchos a muchos).
- **Mensajes y Foros:** Un mensaje está asociado a un foro (muchos a uno).
- **Mensajes y Usuarios:** Un mensaje está asociado a un usuario (muchos a uno).
- **Ofertas y Consumidores:** Una oferta está asociada a un consumidor (muchos a uno).
- **Ofertas y Ofertantes:** Una oferta está asociada a un ofertante (muchos a uno).
- **Intereses y Consumidores:** Un consumidor puede tener múltiples intereses y un interés puede ser de múltiples consumidores (muchos a muchos).
- **Intereses y Ofertantes:** Un ofertante puede tener múltiples intereses y un interés puede ser de múltiples ofertantes (muchos a muchos).

Ejemplos de Datos

- **Actividades:** Senderismo, Clase de Cocina Italiana, Concierto de Jazz.
- **Categorías:** Deportes, Cocina, Música.
- **Intereses:** Senderismo, Cocina Italiana, Jazz.

3. Configuración del Entorno de Desarrollo

En Linux:

Instrucciones paso a paso para configurar el entorno de desarrollo en Ubuntu

- Clonar el Repositorio: Desde tu terminal o cmd, se ejecuta el git clone del repositorio
- Instalación de Dependencias: • Desde el frontend, se hace npm install para que se se descarguen las dependencias necesarias de angular.

- Desde el backend, en la carpeta raíz, se ejecuta: mvn spring-boot:run

- Ejecución del Proyecto: • Desde el frontend, ng serve –open.
- Desde el backend, el mismo comando del punto anterior. Arrancar el proyecto en localhost: 9001.

En Windows:

- Se instala Intellij IDEA y Visual Studio junto con angular y docker.
- Se hace git clone del repositorio para obtener el back y el front
- Se abre el proyecto back en Intellij y se configura la base de datos en MariaDB. Se ejecuta el back.
- Se abre el proyecto front en Visual Studio con todas las dependencias de angular y se ejecuta ng serve –open para abrir el proyecto en el puerto asignado.

4. Desarrollo del Backend

El backend de la aplicación está desarrollado en Spring Boot y maneja la gestión de una plataforma de eventos sociales, donde ofertantes pueden crear actividades y consumidores pueden unirse a ellas. El backend incluye las siguientes entidades principales:

- **Usuario**: La clase base para todos los usuarios de la plataforma.

- **Consumidor**: Una extensión de Usuario, que representa a los usuarios que participan en actividades.
- **Ofertante**: Una extensión de Usuario, que representa a los usuarios que crean actividades.
- **Oferta**: Representa ofertas hechas por los ofertantes.
- **Actividad**: Representa las actividades creadas por los ofertantes.
- **Categoría**: Categoriza las actividades.
- **Rol**: Representa los roles de usuario (e.g., ROLE_USER, ROLE_ADMIN).
- **Mensaje**: Representa los mensajes dentro de los foros.
- **Foro**: Un foro de discusión asociado a una actividad.

Cada entidad tiene sus correspondientes repositorios, servicios, mapeadores(mappers), DTOs, controladores y una configuración de seguridad en Spring Security (con JWT).

Análisis de los Controladores del BackEnd

Controlador AuthenticationController

Descripción General

El controlador AuthenticationController maneja las solicitudes de autenticación y registro para los usuarios de la aplicación. Proporciona endpoints para registrar y autenticar tanto a consumidores como a ofertantes, así como un endpoint para subir imágenes de perfil.

Endpoints

1. Registrar un Consumidor

- URL: /api/v1/auth/register
- Método HTTP: POST
- Descripción: Registra un nuevo consumidor en la plataforma.
- Parámetros de la Solicitud:
    - request (en el cuerpo de la solicitud, tipo RegisterRequest): Objeto que contiene la información necesaria para registrar al consumidor (e.g., nombre, email, contraseña).
- Respuesta Exitosa:
    - Código: 200 OK
    - Cuerpo: AuthenticationResponse - Contiene información sobre el consumidor registrado y un token de autenticación.

2. Registrar un Ofertante

- URL: /api/v1/auth/register/ofertante
- Método HTTP: POST
- Descripción: Registra un nuevo ofertante en la plataforma.
- Parámetros de la Solicitud:
    - request (en el cuerpo de la solicitud, tipo RegisterRequest): Objeto que contiene la información necesaria para registrar al ofertante (e.g., nombre, email, contraseña).
- Respuesta Exitosa:
    - Código: 200 OK
    - Cuerpo: AuthenticationResponse - Contiene información sobre el ofertante registrado y un token de autenticación.

3. Autenticar un Consumidor

- URL: /api/v1/auth/authenticate
- Método HTTP: POST
- Descripción: Autentica a un consumidor en la plataforma.
- Parámetros de la Solicitud:
    - request (en el cuerpo de la solicitud, tipo AuthenticationRequest): Objeto que contiene las credenciales del consumidor (e.g., email, contraseña).
- Respuesta Exitosa:
    - Código: 200 OK
    - Cuerpo: AuthenticationResponse - Contiene información sobre el consumidor autenticado y un token de autenticación.
- Registro de Actividad:
    - El controlador registra un mensaje informativo sobre la autenticación del usuario.

4. Autenticar un Ofertante

- URL: /api/v1/auth/authenticate/ofertante
- Método HTTP: POST
- Descripción: Autentica a un ofertante en la plataforma.
- Parámetros de la Solicitud:
    - request (en el cuerpo de la solicitud, tipo AuthenticationRequest): Objeto que contiene las credenciales del ofertante (e.g., email, contraseña).
- Respuesta Exitosa:
    - Código: 200 OK
    - Cuerpo: AuthenticationResponse - Contiene información sobre el ofertante autenticado y un token de autenticación.
- Registro de Actividad:
    - El controlador registra un mensaje informativo sobre la autenticación del usuario.

5. Subir Imagen de Perfil

- URL: /api/v1/auth/imagen/{id}
- Método HTTP: POST

- Descripción: Sube una imagen de perfil para un usuario específico.
- Parámetros de la Solicitud:
    - file (en el formulario de datos, tipo MultipartFile): Archivo de imagen a subir.
    - id (en la URL, tipo Long): Identificador del usuario al que se le asociará la imagen de perfil.
- Respuesta Exitosa:
    - Código: 200 OK
- Respuesta de Error:
    - Código: 500 Internal Server Error
    - Cuerpo: String - Mensaje de error indicando que hubo un problema al subir la imagen.

Manejo de Errores

- Si ocurre un error durante la subida de la imagen de perfil, se devuelve un código de estado 500 Internal Server Error con un mensaje de error en el cuerpo de la respuesta.

Registro de Actividad

- El controlador utiliza el logger para registrar información sobre las solicitudes de autenticación.

Inyección de Dependencias

- Servicios Utilizados:
    - AuthenticationService: Servicio utilizado para manejar la lógica de negocio relacionada con la autenticación y el registro de usuarios.

Controlador ActividadController

El controlador ActividadController gestiona las operaciones relacionadas con las actividades en la aplicación. Permite realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre las actividades, así como listar todas las actividades y categorías disponibles. Este controlador utiliza diversos servicios y repositorios para llevar a cabo las operaciones.

Dependencias

- ActividadService
- OfertanteService
- ConsumidorService
- ForoService
- MensajeService
- CategoriaService
- Mapper<Actividad, ActividadDTO>
- ForoRepository
- MensajeRepository

Endpoints

1. Obtener una actividad por su ID

@GetMapping(path = "/api/actividad/{id}")
public ResponseEntity<ActividadDTO> getActividad(@PathVariable Long id)

- Descripción: Obtiene una actividad específica por su ID.
- Parámetros:
    - id (Path Variable): ID de la actividad a obtener.
- Respuesta:
    - 200 OK con el DTO de la actividad si se encuentra.
    - 404 Not Found si no se encuentra la actividad.

Obtener todas las actividades

@GetMapping(path = "/api/actividad")
public ResponseEntity<Iterable<ActividadDTO>> getActividades()

- Descripción: Obtiene todas las actividades.
- Respuesta:
    - 200 OK con una lista de DTOs de todas las actividades.

@GetMapping("/api/actividad/list")
public List<Actividad> listarTodosLasActividades()

- Descripción: Lista todas las actividades.
- Respuesta:
    - 200 OK con una lista de todas las actividades.

2. Crear una actividad

@PostMapping(path = "/api/actividad")
public ResponseEntity<ActividadDTO> createActividad(@RequestBody ActividadDTO actividadDTO)

- Descripción: Crea una nueva actividad.
- Parámetros:
    - actividadDTO (Request Body): DTO de la actividad a crear.
- Respuesta:
    - 201 Created con el DTO de la actividad creada.
    - 400 Bad Request si hay un error en la creación.

3. Actualizar una actividad (parcialmente)

@PatchMapping(path = "/api/actividad/{id}")
public ResponseEntity<ActividadDTO> patchActividad(@PathVariable("id") Long id,
@RequestBody ActividadDTO actividadDTO)

- Descripción: Actualiza parcialmente una actividad.
- Parámetros:
    - id (Path Variable): ID de la actividad a actualizar.
    - actividadDTO (Request Body): DTO con los datos a actualizar.
- Respuesta:
    - 200 OK con el DTO de la actividad actualizada.
    - 404 Not Found si no se encuentra la actividad.
    - 400 Bad Request si hay un error en la actualización.
    - 500 Internal Server Error si ocurre un error del servidor.

Actualizar una actividad (completamente)

@PutMapping(path = "/api/actividad/update/{id}")
public ResponseEntity<ActividadDTO> updateActividad(@PathVariable("id") Long id,
@RequestBody ActividadDTO actividadDTO)

- Descripción: Actualiza completamente una actividad.
- Parámetros:
    - id (Path Variable): ID de la actividad a actualizar.
    - actividadDTO (Request Body): DTO con los datos a actualizar.
- Respuesta:
    - 200 OK con el DTO de la actividad actualizada.
    - 404 Not Found si no se encuentra la actividad.
    - 400 Bad Request si hay un error en la actualización.
    - 500 Internal Server Error si ocurre un error del servidor.

4. Borrar una actividad

@DeleteMapping(path = "/api/actividad/{id}")
public ResponseEntity deleteActividad(@PathVariable("id") Long id)

- Descripción: Elimina una actividad por su ID.
- Parámetros:
    - id (Path Variable): ID de la actividad a eliminar.
- Respuesta:
    - 204 No Content si la actividad es eliminada con éxito.

5. Listar todas las categorías

@GetMapping("/api/actividad/categorias")
public List<Categoria> listarTodosLasCategorias()

- Descripción: Lista todas las categorías disponibles.
- Respuesta:
    - 200 OK con una lista de todas las categorías.

Controlador ConsumidorController y OfertanteController

El controlador ConsumidorController gestiona las operaciones relacionadas con los consumidores en la aplicación. Permite realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre los consumidores. El controlador OfertanteController es funcionalmente equivalente al ConsumidorController, pero se enfoca en los ofertantes en lugar de los consumidores.

Dependencias

- ConsumidorService / OfertanteService
- Mapper<Consumidor, ConsumidorDTO> / Mapper<Ofertante, OfertanteDTO>

Endpoints

1. Obtener un consumidor por su ID

@GetMapping(path = "/api/consumidor/{id}")
public ResponseEntity<ConsumidorDTO> getConsumidor(@PathVariable Long id)

- Descripción: Obtiene un consumidor específico por su ID.
- Parámetros:
    - id (Path Variable): ID del consumidor a obtener.
- Respuesta:
    - 200 OK con el DTO del consumidor si se encuentra.

- 404 Not Found si no se encuentra el consumidor.

2. Obtener todos los consumidores

@GetMapping(path = "/api/consumidor")
public ResponseEntity<Iterable<ConsumidorDTO>> getConsumidores()

- Descripción: Obtiene todos los consumidores.
- Respuesta:
  - 200 OK con una lista de DTOs de todos los consumidores.

3. Crear un consumidor

@PostMapping(path = "/api/consumidor")
public ResponseEntity<ConsumidorDTO> createConsumidor(@RequestBody ConsumidorDTO consumidorDTO)

- Descripción: Crea un nuevo consumidor.
- Parámetros:
  - consumidorDTO (Request Body): DTO del consumidor a crear.
- Respuesta:
  - 201 Created con el DTO del consumidor creado.

Actualizar un consumidor (parcialmente)

@PatchMapping(path = "/api/consumidor/{id}")
public ResponseEntity<ConsumidorDTO> updateConsumidor(@PathVariable("id") Long id, @RequestBody ConsumidorDTO consumidorDTO)

- Descripción: Actualiza parcialmente un consumidor.
- Parámetros:
  - id (Path Variable): ID del consumidor a actualizar.
  - consumidorDTO (Request Body): DTO con los datos a actualizar.
- Respuesta:
  - 200 OK con el DTO del consumidor actualizado.
  - 404 Not Found si no se encuentra el consumidor.
  - 400 Bad Request si hay un error en la actualización.

4 Borrar un consumidor

@DeleteMapping(path = "/api/consumidor/{id}")

public ResponseEntity deleteConsumidor(@PathVariable("id") Long id)

- Descripción: Elimina un consumidor por su ID.
- Parámetros:
    - id (Path Variable): ID del consumidor a eliminar.
- Respuesta:
    - 204 No Content si el consumidor es eliminado con éxito.

Controlador ForoController

El controlador ForoController maneja las operaciones relacionadas con los foros y los mensajes dentro de la aplicación. Permite listar foros y mensajes, así como crear nuevos mensajes dentro de un foro.

Dependencias

- ActividadService
- UsuarioService
- ForoService
- MensajeService
- Mapper<Mensaje, MensajeDTO>

Endpoints

1. Listar todos los mensajes por ID del foro

@GetMapping("/api/foro/mensajes/{id}")
public List<Mensaje> listarTodosLosMensajesPorForoId(@PathVariable("id") Long id)

- Descripción: Obtiene todos los mensajes asociados a un foro específico.
- Parámetros:
    - id (Path Variable): ID del foro cuyos mensajes se quieren listar.
- Respuesta:
    - 200 OK con una lista de mensajes del foro especificado.

2. Listar foro por ID

@GetMapping("/api/foro/{id}")
public Foro listarForo(@PathVariable("id") Long id)

- Descripción: Obtiene un foro específico por su ID.

- Parámetros:
    - id (Path Variable): ID del foro a obtener.
- Respuesta:
    - 200 OK con el foro si se encuentra.
    - 404 Not Found si no se encuentra el foro.

3. Crear un mensaje en un foro

@PostMapping("/api/foro")
public ResponseEntity<MensajeDTO> createMensaje(@RequestBody MensajeDTO mensajeDTO)

- Descripción: Crea un nuevo mensaje en un foro.
- Parámetros:
    - mensajeDTO (Request Body): DTO del mensaje a crear, que incluye el ID del usuario, el ID del foro, el contenido y el título del mensaje.
- Respuesta:
    - 201 Created con el DTO del mensaje creado.
    - 400 Bad Request si hay un error en la creación del mensaje.

Controlador OfertasController

Descripción General

El controlador OfertasController maneja las operaciones relacionadas con las ofertas dentro de la aplicación. Permite listar ofertas por ID de ofertante y crear nuevas ofertas.

Dependencias

- OfertaService
- OfertanteService
- ConsumidorService
- Mapper<Ofertas, OfertaDTO>

Endpoints

1. Listar todas las ofertas por ID del ofertante

@GetMapping("/api/ofertas/{id}")
public List<Ofertas> listarTodosLasOfertasPorOfertanteId(@PathVariable("id") Long id)

- Descripción: Obtiene todas las ofertas asociadas a un ofertante específico.
- Parámetros:
    - id (Path Variable): ID del ofertante cuyas ofertas se quieren listar.
- Respuesta:
    - 200 OK con una lista de ofertas del ofertante especificado.
    - 404 Not Found si no se encuentra el ofertante.

2. Crear una nueva oferta

```
@PostMapping("/api/ofertas")
public ResponseEntity<OfertaDTO> createMensaje(@RequestBody OfertaDTO ofertaDTO)
```

- Descripción: Crea una nueva oferta.
- Parámetros:
    - ofertaDTO (Request Body): DTO de la oferta a crear, que incluye el ID del ofertante, el ID del consumidor, y otros detalles de la oferta.
- Respuesta:
    - 201 Created con el DTO de la oferta creada.
    - 400 Bad Request si hay un error en la creación de la oferta.

5. Desarrollo del Frontend

Componentes

A continuación se presenta la documentación de los componentes del FrontEnd de la aplicación en Angular. Cada componente tiene una breve descripción y sus principales características. Para consultar todas las características de cada componente se adjunta la documentación completa realizada en Compodoc en el enlace de la carpeta raíz del FrontEnd ubicado en Documentation/index.html.

AppComponent

Descripción

Componente raíz de la aplicación Angular.

ActividadesComponent

Descripción

Muestra una lista de actividades disponibles en la plataforma.

DetalleActividadComponent

Descripción

Muestra los detalles de una actividad específica.

EditarActividadComponent

Descripción

Permite editar los detalles de una actividad existente.

ForoDetalleComponent

Descripción

Muestra los detalles de un foro específico y sus mensajes.

ForosComponent

Descripción

Muestra una lista de foros disponibles en la plataforma.

IndexComponent

Descripción

Página de inicio de la aplicación.

LoginComponent

Descripción

Formulario de inicio de sesión para usuarios consumidores.

LoginOfertanteComponent

Descripción

Formulario de inicio de sesión para usuarios ofertantes.

NavComponent

Descripción

Barra de navegación principal de la aplicación.

OfertantesPanelComponent

Descripción

Panel para la gestión de usuarios ofertantes.

PerfilConsumidorComponent

Descripción

Página de perfil para los usuarios consumidores.

PerfilOfertanteComponent

Descripción

Página de perfil para los usuarios ofertantes.

RegisterConsumidorComponent

Descripción

Formulario de registro para nuevos usuarios consumidores.

RegisterOfertanteComponent

Descripción

Formulario de registro para nuevos usuarios ofertantes.

RegistroActComponent

Descripción

Formulario para registrar nuevas actividades.

UsuarioEditarComponent

Descripción

Formulario para editar la información de los usuarios.

Servicios

A continuación se presenta la documentación de los servicios de la misma aplicación Angular. Cada servicio tiene una breve descripción y sus principales características. Para consultar todos los métodos y variables de cada servicio se adjunta la documentación completa realizada en Compodoc en el enlace de la carpeta raíz del FrontEnd ubicado en Documentation/index.html.

ActividadService

Descripción

Proporciona métodos para gestionar actividades dentro de la aplicación.

Características

- Inyectable: @Injectable({ providedIn: 'root' })
- Dependencias: HttpClient para realizar solicitudes HTTP.

AuthService

Descripción

Maneja la autenticación de los usuarios en la aplicación.

Características

- Inyectable: @Injectable({ providedIn: 'root' })
- Dependencias: HttpClient para realizar solicitudes HTTP.

ConsumidorService

Descripción

Proporciona métodos para gestionar consumidores dentro de la aplicación.

Características

- Inyectable: @Injectable({ providedIn: 'root' })
- Dependencias: HttpClient para realizar solicitudes HTTP.

ForoService

Descripción

Proporciona métodos para gestionar foros dentro de la aplicación.

Características

- Inyectable: @Injectable({ providedIn: 'root' })
- Dependencias: HttpClient para realizar solicitudes HTTP.

LoginService

Descripción

Maneja los procesos de inicio de sesión para consumidores y ofertantes.

Características

- Inyectable: @Injectable({ providedIn: 'root' })
- Dependencias: HttpClient para realizar solicitudes HTTP.

NavService

Descripción

Gestiona el estado de la barra de navegación y las opciones de menú.

Características

- Inyectable: @Injectable({ providedIn: 'root' })

OfertanteService

Descripción

Proporciona métodos para gestionar ofertantes dentro de la aplicación.

Características

- Inyectable: @Injectable({ providedIn: 'root' })
- Dependencias: HttpClient para realizar solicitudes HTTP.

OfertaService

Descripción

Proporciona métodos para gestionar ofertas dentro de la aplicación.

Características

- Inyectable: @Injectable({ providedIn: 'root' })
- Dependencias: HttpClient para realizar solicitudes HTTP.

RegisterService

Descripción

Maneja los procesos de registro para nuevos consumidores y ofertantes.

Características

- Inyectable: @Injectable({ providedIn: 'root' })
- Dependencias: HttpClient para realizar solicitudes HTTP.

6. Dificultades en el desarrollo

El desarrollo de un aplicación web completa conlleva numerosas difucultades en muchos puntos, desde la generación de la base de datos, especialmente tratar que las relaciones se den correctamente para no generar conflictos en el backend, hasta el uso eficiente de observables para el correcto funcionamiento de un single page application en Angular. Los puntos que más dificultades me han generado al desarrollar una aplicación como WeMeet han recaído especialmente en:

Gestión de la subida de archivos: Implementar un mecanismo en el backend para recibir y procesar archivos de imagen enviados desde el frontend. Esto implica manejar correctamente las solicitudes HTTP multipartes y extraer el archivo de imagen del cuerpo de la solicitud.

Validación de archivos y seguridad: Es fundamental implementar medidas de seguridad para prevenir la subida de archivos maliciosos o la creación o acceso a rutas para las que el usuario incorrecto no debe tener permiso. Esto implica realizar una validación exhaustiva en Spring Security para que todos los usuarios cumplan con los requisitos establecidos y que no representen riesgos de seguridad.

Combinación de observables: Se hace necesario combinar múltiples observables para realizar operaciones más complejas, especialmente para que el navegador vaya recargando y actualizando correctamente los nuevos datos de acceso de los usuarios (El jwt se realiza mediante LocalStorage).

7. Bibliografía

1. *Walls, C. (2016). Spring Boot in Action. Manning Publications.*
2. *Angular. (s.f.). Angular Documentation.*
3. *Pivotal Software, Inc. (s.f.). Spring Boot Documentation.*
4. *Angular University. (s.f.)*
5. *Stack Overflow. (s.f.)*
6. *Reddit. (s.f.).*