Documentación proyecto integrado

David González García

Introducción

A continuación voy a centrarme en los puntos principales a la hora de la creación del proyecto. Tocaré tanto los scripts para la creación de la base de datos, como todo aquello necesario y más técnico que he incluido en cada apartado de la aplicación.

El proyecto se basa principalmente en la demanda y oferta de eventos y cómo los usuarios pueden escoger y consumir la actividad o evento que deseen. Por una parte tenemos así los ofertantes y por otro lado los consumidores (los cuales comparten características con los consumidores). Estos últimos podrán (al igual que los consumidores) consumir eventos de otros usuarios de la plataforma, ya que estos solamente tienen como característica única el poder crear actividades.

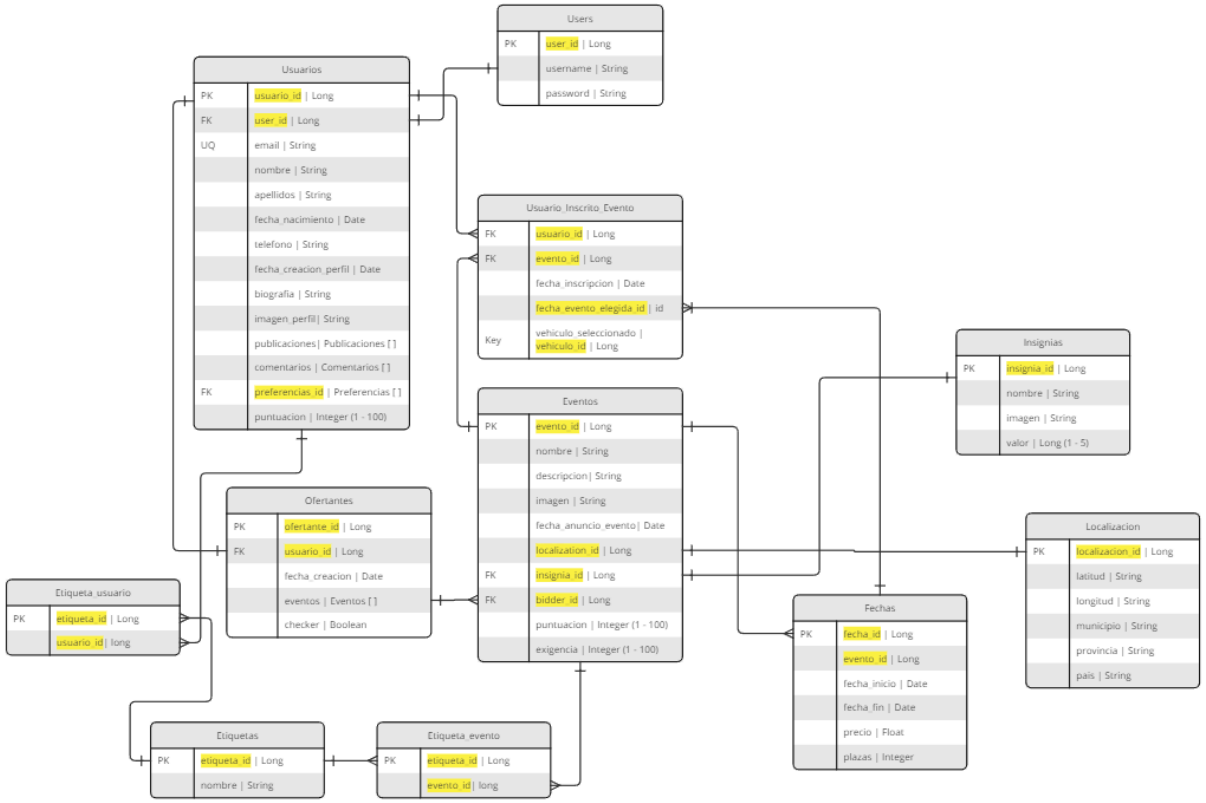
En la web habrá 4 apartados principalmente:

- Inicio
- Eventos
- Ofertantes
- Perfil

Siendo este último donde podremos realizar todo tipo de actualizaciones de nuestro perfil (actualizarlo, borrarlo, cambiar de consumidor a ofertante o viceversa, etc...). En la página inicial o de inicio podremos encontrar un poco de todo en general relacionado con la página web, en eventos encontraremos todos los eventos disponibles, como también (si nos metemos en alguno de ellos) podremos verlo tanto con más detalles cómo consumirlo. Por último, en ofertantes podremos encontrar todos los ofertantes que dispone la página web, los cuales nos aparecerán y si nos metemos en ellos podremos ver qué eventos están ofertando cada uno de ellos.

Base de datos

En cuanto a la base de datos que vamos a estar usando será una base de datos MariaDB la cual alojaremos en un contenedor docker y que tendrá la siguiente estructura:



Si nos fijamos podemos ver que la tabla ofertante contiene un usuario. Con esto evitamos la repetición de datos y ganaremos en organización y optimización de la base de datos.

Script de creacion

```
CREATE TABLE `Users` (
  `id` integer PRIMARY KEY,
  `username` varchar(255) UNIQUE NOT NULL,
  `password` varchar(255) NOT NULL
);

CREATE TABLE `Insignias` (
  `id` integer PRIMARY KEY,
  `nombre` varchar(255) NOT NULL,
  `imagen` varchar(255),
  `valor` integer NOT NULL
);

CREATE TABLE `Localizacion` (
  `id` integer NOT NULL,
  `latitud` varchar(255),
  `longitud` varchar(255),
  `municipio` varchar(255),
  `provincia` varchar(255),
  `pais` varchar(255)
);

CREATE TABLE `Vehiculos` (
  `id` integer PRIMARY KEY,
  `nombre` varchar(255) NOT NULL,
  `imagen` varchar(255)
);

CREATE TABLE `Etiquetas` (
  `id` integer PRIMARY KEY,
  `nombre` varchar(255) UNIQUE NOT NULL
);

CREATE TABLE `Usuarios` (
  `id` integer PRIMARY KEY,
  `user_id` integer,
  `email` varchar(255) UNIQUE,
  `nombre` varchar(255) NOT NULL,
  `apellidos` varchar(255),
  `fecha_nacimiento` date NOT NULL,
  `telefono` varchar(255),
  `fecha_creacion_perfil` datetime NOT NULL,
```

```
  `biografia` varchar(255),
  `imagen_perfil` varchar(255),
  `puntuacion` integer
);

CREATE TABLE `UsuarioResenias` (
  `id` integer PRIMARY KEY,
  `usuario_emisor_id` integer,
  `usuario_receptor_id` integer,
  `valoracion` integer
);

CREATE TABLE `Publicaciones` (
  `id` integer PRIMARY KEY,
  `usuario_id` integer,
  `titulo` varchar(255) NOT NULL,
  `descripcion` varchar(255),
  `fecha_subida` datetime NOT NULL,
  `likes` integer NOT NULL,
  `imagen` varchar(255)
);

CREATE TABLE `Comentarios` (
  `id` integer PRIMARY KEY,
  `usuario_id` integer,
  `publicacion_id` integer,
  `fecha` datetime NOT NULL,
  `texto` varchar(255) NOT NULL
);

CREATE TABLE `Ofertantes` (
  `id` integer PRIMARY KEY,
  `usuario_id` integer,
  `fecha_creacion` datetime NOT NULL,
  `checker` bool NOT NULL
);

CREATE TABLE `Eventos` (
  `id` integer PRIMARY KEY,
  `localizacion_id` integer,
  `insignia_id` integer,
  `ofertante_id` integer,
  `nombre` varchar(255) NOT NULL,
  `descripcion` varchar(255) NOT NULL,
  `imagen` varchar(255),
  `fecha_anuncio_evento` datetime NOT NULL,
```

```
  `puntuacion` integer,
  `exigencia` integer NOT NULL
);

CREATE TABLE `Fechas` (
  `id` integer PRIMARY KEY,
  `evento_id` integer,
  `fecha_inicio` date NOT NULL,
  `fecha_fin` date NOT NULL,
  `precio` float NOT NULL,
  `plazas` integer
);

CREATE TABLE `UsuariosEventosResenias` (
  `id` integer PRIMARY KEY,
  `usuario_id` integer,
  `evento_id` integer,
  `puntuacion` integer NOT NULL,
  `fecha_resenia` datetime NOT NULL,
  `informacion` varchar(255)
);

CREATE TABLE `UsuariosInscritosEventos` (
  `id` integer PRIMARY KEY,
  `usuario_id` integer,
  `evento_id` integer,
  `vehiculo_id` integer,
  `fecha_inscripcion` datetime NOT NULL,
  `fecha_elegida` integer
);

CREATE TABLE `VehiculosEventos` (
  `id` integer PRIMARY KEY,
  `vehiculo_id` integer,
  `evento_id` integer,
  `cantidad` integer
);

CREATE TABLE `EtiquetasUsuarios` (
  `id` integer PRIMARY KEY,
  `etiqueta_id` integer,
  `usuario_id` integer
);

CREATE TABLE `EtiquetasEventos` (
  `id` integer PRIMARY KEY,
```

```
  `etiqueta_id` integer,
  `evento_id` integer
);

ALTER TABLE `Usuarios` ADD FOREIGN KEY (`user_id`) REFERENCES `Users`
(`id`);

ALTER TABLE `UsuarioResenias` ADD FOREIGN KEY (`usuario_emisor_id`)
REFERENCES `Usuarios` (`id`);

ALTER TABLE `UsuarioResenias` ADD FOREIGN KEY (`usuario_receptor_id`)
REFERENCES `Usuarios` (`id`);

ALTER TABLE `Publicaciones` ADD FOREIGN KEY (`usuario_id`) REFERENCES
`Usuarios` (`id`);

ALTER TABLE `Comentarios` ADD FOREIGN KEY (`usuario_id`) REFERENCES
`Usuarios` (`id`);

ALTER TABLE `Comentarios` ADD FOREIGN KEY (`publicacion_id`) REFERENCES
`Publicaciones` (`id`);

ALTER TABLE `Ofertantes` ADD FOREIGN KEY (`usuario_id`) REFERENCES
`Usuarios` (`id`);

ALTER TABLE `Eventos` ADD FOREIGN KEY (`localizacion_id`) REFERENCES
`Localizacion` (`id`);

ALTER TABLE `Eventos` ADD FOREIGN KEY (`insignia_id`) REFERENCES
`Insignias` (`id`);

ALTER TABLE `Eventos` ADD FOREIGN KEY (`ofertante_id`) REFERENCES
`Ofertantes` (`id`);

ALTER TABLE `Fechas` ADD FOREIGN KEY (`evento_id`) REFERENCES `Eventos`
(`id`);

ALTER TABLE `UsuariosEventosResenias` ADD FOREIGN KEY (`usuario_id`)
REFERENCES `Usuarios` (`id`);

ALTER TABLE `UsuariosEventosResenias` ADD FOREIGN KEY (`evento_id`)
REFERENCES `Eventos` (`id`);

ALTER TABLE `UsuariosInscritosEventos` ADD FOREIGN KEY (`usuario_id`)
REFERENCES `Usuarios` (`id`);
```

```sql
ALTER TABLE `UsuariosInscritosEventos` ADD FOREIGN KEY (`evento_id`)
REFERENCES `Eventos` (`id`);

ALTER TABLE `UsuariosInscritosEventos` ADD FOREIGN KEY (`vehiculo_id`)
REFERENCES `Vehiculos` (`id`);

ALTER TABLE `UsuariosInscritosEventos` ADD FOREIGN KEY (`fecha_elegida`)
REFERENCES `Fechas` (`id`);

ALTER TABLE `VehiculosEventos` ADD FOREIGN KEY (`vehiculo_id`)
REFERENCES `Vehiculos` (`id`);

ALTER TABLE `VehiculosEventos` ADD FOREIGN KEY (`evento_id`) REFERENCES
`Eventos` (`id`);

ALTER TABLE `EtiquetasUsuarios` ADD FOREIGN KEY (`etiqueta_id`)
REFERENCES `Etiquetas` (`id`);

ALTER TABLE `EtiquetasUsuarios` ADD FOREIGN KEY (`usuario_id`)
REFERENCES `Usuarios` (`id`);

ALTER TABLE `EtiquetasEventos` ADD FOREIGN KEY (`etiqueta_id`)
REFERENCES `Etiquetas` (`id`);

ALTER TABLE `EtiquetasEventos` ADD FOREIGN KEY (`evento_id`) REFERENCES
`Eventos` (`id`);
```

Servidor

AuthController

Este controlador maneja las operaciones relacionadas con la autenticación de usuarios, como el inicio de sesión y el registro.

- login(UserDTO userDTO)
    - Descripción: Método para autenticar a un usuario. Si las credenciales son válidas, se genera y devuelve un token JWT.
    - Respuesta: 200 OK con el token JWT y el ID del usuario si la autenticación es exitosa. 401 UNAUTHORIZED si las credenciales son incorrectas.
- register(RegisterDTO registerDTO)
    - Descripción: Método para registrar un nuevo usuario. Procesa la información de registro y devuelve una respuesta del servidor.
    - Respuesta: 200 OK con la respuesta del servidor sobre el resultado del registro.

EventoController

Este controlador gestiona las operaciones relacionadas con los eventos, como la obtención de todos los eventos, obtener un evento por su ID y agregar un nuevo evento.

- getAll()
    - Descripción: Método para obtener una lista de todos los eventos disponibles.
    - Respuesta: 200 OK con una lista de objetos EventoDTO.
- getById(Long id)
    - Descripción: Método para obtener un evento específico por su ID.
    - Respuesta: 200 OK con el objeto EventoDTO correspondiente al ID proporcionado.
- addEvent(CreateEventoDTO createEventoDTO)
    - Descripción: Método para agregar un nuevo evento.
    - Respuesta: 200 OK con el objeto EventoDTO del evento recién creado.

MediaController

Este controlador maneja las operaciones relacionadas con la gestión de archivos multimedia, como la carga y la obtención de archivos.

- uploadFile(MultipartFile multipartFile)

    - Descripción: Método para cargar un archivo al servidor. Devuelve la URL del archivo cargado.
    - Respuesta: 200 OK con un mapa que contiene la URL del archivo cargado.
- getFile(String filename)
    - Descripción: Método para obtener un archivo por su nombre.
    - Respuesta: 200 OK con el recurso del archivo solicitado.

OfertanteController

Este controlador gestiona las operaciones relacionadas con los ofertantes, como obtener ofertantes por ID de usuario o evento, agregar un nuevo ofertante, y eliminar ofertantes.

- getBidderByUserId(Long id)
    - Descripción: Método para obtener un ofertante por su ID de usuario.
    - Respuesta: 200 OK con el objeto OfertanteDTO correspondiente al usuario. Si no se encuentra, devuelve null.
- getBidderByEventId(Long id)
    - Descripción: Método para obtener un ofertante por su ID de evento.
    - Respuesta: 200 OK con el objeto OfertanteUsuarioDTO correspondiente al evento.
- create(OfertanteDTO ofertanteDTO)
    - Descripción: Método para agregar un nuevo ofertante.
    - Respuesta: 201 CREATED con el objeto OfertanteDTO del ofertante recién creado.
- removeBidderByUserId(Long id)
    - Descripción: Método para eliminar un ofertante por su ID de usuario.
    - Respuesta: 200 OK con un mapa que contiene la respuesta del servidor sobre el resultado de la eliminación.
- checker(CheckOfertanteDTO checkOfertanteDTO)
    - Descripción: Método para verificar un ofertante.
    - Respuesta: 200 OK con el objeto OfertanteDTO actualizado.

UsuarioController

Este controlador maneja las operaciones relacionadas con los usuarios, como agregar, actualizar, eliminar y obtener usuarios.

- create(UsuarioInsertDTO usuarioInsertDTO)
    - Descripción: Método para agregar un nuevo usuario.
    - Respuesta: 201 CREATED con el objeto UsuarioDTO del usuario recién creado.
- update(UsuarioUpdateDTO usuarioUpdateDTO)
    - Descripción: Método para actualizar la información de un usuario existente.

    - Respuesta: 201 CREATED con el objeto UsuarioDTO actualizado.
- remove(Long id)
    - Descripción: Método para eliminar un usuario por su ID.
    - Respuesta: 200 OK con un mapa que contiene la respuesta del servidor sobre el resultado de la eliminación. 404 NOT FOUND si hay un error en el acceso a datos.
- removeAll(Long id)
    - Descripción: Método para eliminar todos los datos relacionados con un usuario por su ID.
    - Respuesta: 200 OK con un mapa que contiene la respuesta del servidor sobre el resultado de la eliminación.
- getById(Long id)
    - Descripción: Método para obtener un usuario específico por su ID.
    - Respuesta: 200 OK con el objeto UsuarioDTO correspondiente al ID proporcionado.
- getAllUsers()
    - Descripción: Método para obtener una lista de todos los usuarios.
    - Respuesta: 200 OK con una lista de objetos UsuarioDTO.