PROYECTO INTEGRADO (ó TFG)

Autor: Daniel Mamani Torres

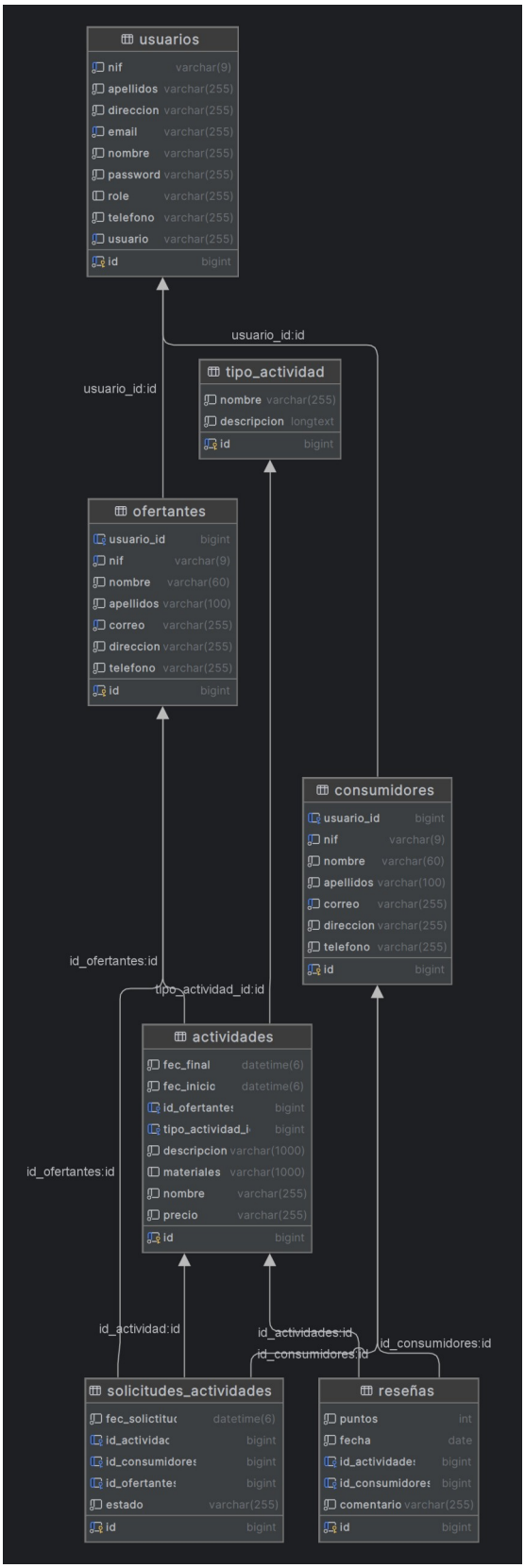
INDICE

1. Resumen del Proyecto

- **Descripción**:
  - La Plataforma de Gestión de Actividades de Ocio es una aplicación web diseñada para ayudar a los usuarios a organizar y planificar sus actividades recreativas y de entretenimiento. La plataforma permite la creación, búsqueda y gestión de eventos y actividades de ocio, facilitando a los usuarios la planificación de su tiempo libre de manera efectiva. Los usuarios pueden descubrir nuevas actividades, unirse a eventos, y compartir sus experiencias con otros.

- Tecnologías Utilizadas:
  - Frontend:
    - HTML5
    - CSS3
    - Angular
    - Bootstrap para el diseño responsivo
  - Backend:
    - Node.js
    - Base de datos MySQL
    - 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.
    - Laragon para la gestion de la base de datos (Docker no opera correctamente)

2. Arquitectura del Sistema



Base de Datos: `eventos`

La base de datos contiene varias tablas. A continuación, se describe la estructura de la tabla `actividades`, que es una de las tablas principales.

Tabla `actividades`

- **Columnas**:
  - `fec_final` (datetime(6)): Fecha y hora de finalización de la actividad.
  - `fec_inicio` (datetime(6)): Fecha y hora de inicio de la actividad.
  - `id` (bigint): Identificador único de la actividad.
  - `id_ofertantes` (bigint, nullable): Identificador del ofertante de la actividad.
  - `tipo_actividad_id` (bigint, nullable): Identificador del tipo de actividad.
  - `descripcion` (varchar(1000)): Descripción detallada de la actividad.
  - `materiales` (varchar(1000), nullable): Materiales necesarios para la actividad.
  - `nombre` (varchar(255)): Nombre de la actividad.

Tabla `consumidores`

- **Columnas**:
  - `id` (bigint): Identificador único del consumidor.
  - `usuario_id` (bigint, nullable): Identificador del usuario asociado.
  - `nif` (varchar(9)): NIF del consumidor.
  - `nombre` (varchar(60)): Nombre del consumidor.
  - `apellidos` (varchar(100)): Apellidos del consumidor.
  - `correo` (varchar(255)): Correo electrónico del consumidor.
  - `direccion` (varchar(255)): Dirección del consumidor.
  - `telefono` (varchar(255)): Teléfono del consumidor.

Tabla `ofertantes`

- **Columnas**:
  - `id` (bigint): Identificador único del ofertante.
  - `usuario_id` (bigint, nullable): Identificador del usuario asociado.
  - `nif` (varchar(9)): NIF del ofertante.
  - `nombre` (varchar(60)): Nombre del ofertante.
  - `apellidos` (varchar(100)): Apellidos del ofertante.

- `correo` (varchar(255)): Correo electrónico del ofertante.
- `direccion` (varchar(255)): Dirección del ofertante.
- `telefono` (varchar(255)): Teléfono del ofertante.

Tabla `reseñas`

- **Columnas**:
  - `puntos` (int): Puntuación de la reseña.
  - `fecha` (date): Fecha de la reseña.
  - `id` (bigint): Identificador único de la reseña.
  - `id_actividades` (bigint, nullable): Identificador de la actividad reseñada.
  - `id_consumidores` (bigint, nullable): Identificador del consumidor que hizo la reseña.
  - `comentario` (varchar(255)): Comentario de la reseña.

Tabla `solicitudes_actividades`

- **Columnas**:
  - `fec_solictitud` (datetime(6)): Fecha y hora de la solicitud.
  - `id` (bigint): Identificador único de la solicitud.
  - `id_actividad` (bigint, nullable): Identificador de la actividad solicitada.
  - `id_consumidores` (bigint, nullable): Identificador del consumidor que hizo la solicitud.
  - `id_ofertantes` (bigint, nullable): Identificador del ofertante relacionado.
  - `estado` (varchar(255)): Estado de la solicitud.

Tabla `tipo_actividad`

- **Columnas**:
  - `id` (bigint): Identificador único del tipo de actividad.
  - `nombre` (varchar(255)): Nombre del tipo de actividad.
  - `descripcion` (longtext): Descripción del tipo de actividad.

Tabla `usuarios`

- **Columnas**:
  - `id` (bigint): Identificador único del usuario.

- `nif` (varchar(9)): NIF del usuario.
- `apellidos` (varchar(255)): Apellidos del usuario.
- `direccion` (varchar(255)): Dirección del usuario.
- `email` (varchar(255)): Correo electrónico del usuario.
- `nombre` (varchar(255)): Nombre del usuario.
- `password` (varchar(255)): Contraseña del usuario.
- `role` (varchar(255), nullable): Rol del usuario.
- `telefono` (varchar(255)): Teléfono del usuario.
- `usuario` (varchar(255)): Nombre de usuario.

Descripción de las Tablas y sus Relaciones

Tabla `actividades`

Propósito: Almacena información sobre las actividades disponibles.

- **Relaciones**:
  - `id_ofertantes`: Clave foránea que se refiere a `ofertantes.id`, indicando qué ofertante ofrece la actividad.
  - `tipo_actividad_id`: Clave foránea que se refiere a `tipo_actividad.id`, especificando el tipo de la actividad.

Tabla `consumidores`

Propósito: Almacena información sobre los consumidores que participan en actividades.

- **Relaciones**:
  - `usuario_id`: Clave foránea que se refiere a `usuarios.id`, relacionando un consumidor con un usuario.

Tabla `ofertantes`

Propósito: Almacena información sobre los ofertantes que organizan actividades.

- **Relaciones**:
  - `usuario_id`: Clave foránea que se refiere a `usuarios.id`, relacionando un ofertante con un usuario.

Tabla `reseñas`

Propósito: Almacena reseñas de actividades realizadas por consumidores.

- **Relaciones**:
  - `id_actividades`: Clave foránea que se refiere a `actividades.id`, indicando la actividad reseñada.
  - `id_consumidores`: Clave foránea que se refiere a `consumidores.id`, indicando el consumidor que realizó la reseña.

Tabla `solicitudes_actividades`

Propósito: Almacena las solicitudes de participación en actividades realizadas por consumidores.

- **Relaciones**:
    - id_actividad: Clave foránea que se refiere a actividades.id, indicando la actividad solicitada.
    - id_consumidores: Clave foránea que se refiere a consumidores.id, indicando el consumidor que realizó la solicitud.
    - id_ofertantes: Clave foránea que se refiere a ofertantes.id, indicando el ofertante relacionado con la solicitud.

Tabla tipo_actividad

Propósito: Almacena los diferentes tipos de actividades que pueden ofrecerse.

- **Relaciones**:
    - Relacionada con la tabla actividades a través de tipo_actividad_id.

Tabla usuarios

Propósito: Almacena información básica de los usuarios, incluyendo tanto consumidores como ofertantes.

- **Relaciones**:
    - Relaciona consumidores y ofertantes a través de usuario_id.

Resumen de Relaciones

- **Usuarios y Consumidores**: Un usuario puede ser un consumidor (relacionado a través de usuario_id en la tabla consumidores).
- **Usuarios y Ofertantes**: Un usuario puede ser un ofertante (relacionado a través de usuario_id en la tabla ofertantes).
- **Actividades y Ofertantes**: Una actividad es ofrecida por un ofertante (relacionado a través de id_ofertantes en la tabla actividades).
- **Actividades y Tipo de Actividad**: Una actividad tiene un tipo específico (relacionado a través de tipo_actividad_id en la tabla actividades).
- **Reseñas y Actividades**: Una reseña está relacionada con una actividad específica (a través de id_actividades).
- **Reseñas y Consumidores**: Una reseña es realizada por un consumidor (a través de id_consumidores).
- **Solicitudes y Actividades**: Una solicitud está relacionada con una actividad (a través de id_actividad).

- **Solicitudes y Consumidores**: Una solicitud es realizada por un consumidor (a través de `id_consumidores`).
- **Solicitudes y Ofertantes**: Una solicitud puede estar relacionada con un ofertante (a través de `id_ofertantes`).

Esta estructura permite gestionar y relacionar usuarios, actividades, reseñas y solicitudes de manera organizada y eficiente.

Flujo de Datos en la Base de Datos de Eventos

1. Registro de Usuarios

Tablas Involucradas: `usuarios`

- **Descripción**: Los nuevos usuarios se registran proporcionando información básica como NIF, nombre, apellidos, dirección, email, contraseña, rol y teléfono. Estos datos se almacenan en la tabla `usuarios`.

2. Registro de Consumidores y Ofertantes

Tablas Involucradas: `consumidores`, `ofertantes`

- **Descripción**:
  - Los usuarios pueden registrarse como consumidores o ofertantes.
  - **Consumidores**: Cuando un usuario decide registrarse como consumidor, se crea una entrada en la tabla `consumidores` con un `usuario_id` que referencia al `id` del usuario en la tabla `usuarios`.
  - **Ofertantes**: De manera similar, cuando un usuario se registra como ofertante, se crea una entrada en la tabla `ofertantes` con un `usuario_id` que referencia al `id` del usuario en la tabla `usuarios`.

3. Creación de Tipos de Actividades

Tablas Involucradas: `tipo_actividad`

- **Descripción**: Los administradores pueden definir diferentes tipos de actividades, que se almacenan en la tabla `tipo_actividad` con un `nombre` y una `descripción`.

4. Publicación de Actividades

Tablas Involucradas: `actividades`

- **Descripción**:
  - Los ofertantes pueden crear nuevas actividades, proporcionando detalles como fechas de inicio y fin (`fec_inicio`, `fec_final`), descripción, materiales necesarios, nombre de la actividad y precio.
  - Cada actividad está vinculada a un `ofertante` mediante `id_ofertantes` y a un `tipo_actividad` mediante `tipo_actividad_id`.

5. Solicitud de Participación en Actividades

Tablas Involucradas: `solicitudes_actividades`

- **Descripción**:
  - Los consumidores pueden solicitar participar en actividades.
  - Se crea una entrada en la tabla `solicitudes_actividades` que incluye la fecha de solicitud (`fec_solictitud`), el `id` de la actividad solicitada (`id_actividad`), el `id` del consumidor que realiza la solicitud (`id_consumidores`) y el estado de la solicitud.

6. Participación en Actividades

Tablas Involucradas: `actividades`, `solicitudes_actividades`

- **Descripción**:
  - Una vez que la solicitud de un consumidor es aprobada, el consumidor puede participar en la actividad.
  - El estado de la solicitud en `solicitudes_actividades` se actualiza para reflejar la participación.

7. Evaluación y Reseñas de Actividades

Tablas Involucradas: `reseñas`

- **Descripción**:
  - Después de participar en una actividad, los consumidores pueden dejar reseñas.
  - Se crea una entrada en la tabla `reseñas` que incluye la puntuación (`puntos`), la fecha de la reseña (`fecha`), el `id` de la actividad reseñada (`id_actividades`), el `id` del consumidor que realizó la reseña (`id_consumidores`) y un comentario (`comentario`).

Resumen del Flujo de Datos

1. **Registro de Usuarios**: Un usuario se registra y se guarda en `usuarios`.
2. **Registro de Consumidores/Ofertantes**: El usuario se registra como consumidor (`consumidores`) o ofertante (`ofertantes`).
3. **Definición de Tipos de Actividades**: Los tipos de actividades se definen en `tipo_actividad`.

4. **Creación de Actividades**: Los ofertantes crean actividades que se almacenan en `actividades`.
5. **Solicitud de Actividades**: Los consumidores solicitan participar en actividades, creando entradas en `solicitudes_actividades`.
6. **Participación en Actividades**: Las solicitudes son aprobadas y los consumidores participan en las actividades.
7. **Evaluación de Actividades**: Los consumidores dejan reseñas en `reseñas` después de participar en las actividades.

3. Configuración del Entorno de Desarrollo

Instalación: Instrucciones paso a paso para configurar el entorno de desarrollo.

- **Clonar el Repositorio**: Desde tu terminal o cmd, ejecutas el siguiente comando:

  git clone https://github.com/dammamtor/judas-tfg.git

- **Instalación de Dependencias**:
  - Desde el frontend, haz npm install para que se te descargue las dependencias necesarias de angular.
  - Desde el backend, en la carpeta raíz, ejecutas: mvn spring-boot:run
- **Ejecución del Proyecto**:
  - Desde el frontend, ng serve –open.
  - Desde el backend, el mismo comando del punto anterior (si empiezas de cero, el comando te descarga las dependencias necesarias y luego ejecuta la aplicación

5. Estructura del Proyecto

A continuación vamos a explicar la estructura de los directorios tanto del backend como del frontend.

Backend:
Directorios principales

- **controller**: Contiene las clases que manejan las solicitudes HTTP. Estas clases se encargan de recibir las peticiones del cliente, procesarlas y devolver una respuesta adecuada.
- **model**: Incluye las clases que representan el modelo de datos de la aplicación. Este paquete puede contener entidades JPA que se mapean a tablas de la base de datos.
    - **DTO**: Contiene clases Data Transfer Object (DTO) que se utilizan para transferir datos entre las capas de la aplicación.
- **repository**: Contiene las interfaces de repositorio que interactúan con la base de datos. Estas interfaces extienden de `JpaRepository` u otras interfaces de Spring Data JPA.
- **services**: Contiene las interfaces y las implementaciones de los servicios que contienen la lógica de negocio de la aplicación.

Estructura detallada

controller

- **ActividadController.java**: Controlador para manejar las solicitudes relacionadas con actividades.
- **ConsumidoresController.java**: Controlador para manejar las solicitudes relacionadas con consumidores.
- **OfertantesController.java**: Controlador para manejar las solicitudes relacionadas con ofertantes.
- **TipoActividadController.java**: Controlador para manejar las solicitudes relacionadas con tipos de actividades.
- **UsuarioController.java**: Controlador para manejar las solicitudes relacionadas con usuarios.

model

- **Actividades.java**: Clase de modelo para representar la entidad de actividades.
- **Consumidores.java**: Clase de modelo para representar la entidad de consumidores.

- **Ofertantes.java**: Clase de modelo para representar la entidad de ofertantes.
- **Reseñas.java**: Clase de modelo para representar la entidad de reseñas.
- **SolicitudesActividades.java**: Clase de modelo para representar la entidad de solicitudes de actividades.
- **TipoActividad.java**: Clase de modelo para representar la entidad de tipos de actividad.
- **Usuario.java**: Clase de modelo para representar la entidad de usuarios.
- **DTO**: Subdirectorio que contiene las clases DTO:
    - **ActividadesRequestDTO.java**: Clase DTO para las solicitudes relacionadas con actividades.

repository

- **ActividadesRepository.java**: Repositorio para la entidad de actividades.
- **ConsumidoresRepository.java**: Repositorio para la entidad de consumidores.
- **OfertantesRepository.java**: Repositorio para la entidad de ofertantes.
- **ReseñasRepository.java**: Repositorio para la entidad de reseñas.
- **SolicitudesActividadesRepository.java**: Repositorio para la entidad de solicitudes de actividades.
- **TipoActividadRepository.java**: Repositorio para la entidad de tipos de actividad.
- **UsuarioRepository.java**: Repositorio para la entidad de usuarios.

services

- **Interfaces**: Definen los métodos que deben implementar los servicios.
    - **ActividadesServices.java**, **ConsumidoresServices.java**, **OfertantesServices.java**, **ReseñasServices.java**, **SolicitudesActividadesServices.java**, **TipoActividadServices.java**, **UsuarioServices.java**.
- **Implementaciones**: Proveen la lógica de negocio y la implementación de las interfaces de servicios.
    - **ActividadesServicesImpl.java**, **ConsumidoresServicesImpl.java**, **OfertantesServicesImpl.java**, **ReseñasServicesImpl.java**, **SolicitudesActividadesServicesImpl.java**, **TipoActividadServicesImpl.java**, **UsuarioServicesImpl.java**

Frontend:
En el directorio `src/app` se encuentran los componentes principales, los servicios, los modelos y los archivos de configuración y enrutamiento:

- **Archivos principales**:
  - `app.component.css`: Archivo de estilos globales para el componente principal.
  - `app.component.html`: Archivo de plantilla HTML para el componente principal.
  - `app.component.spec.ts`: Pruebas unitarias para el componente principal.
  - `app.component.ts`: Lógica del componente principal.
  - `app.config.ts`: Archivo de configuración para la aplicación.
  - `app.routes.ts`: Archivo de configuración de rutas para la aplicación.
- **Directorios**:
  - `components`: Contiene todos los componentes individuales de la aplicación.
  - `models`: Contiene los modelos de datos utilizados en la aplicación.
  - `services`: Contiene los servicios que proporcionan funcionalidades específicas.

2. Componentes

En el directorio `components` se encuentran los distintos componentes de la aplicación. Cada componente tiene su propio subdirectorio que contiene los archivos necesarios para su funcionamiento:

- **Subdirectorios de componentes**:
  - `aceptar-solicitudes`
    - `aceptar-solicitudes.component.css`: Estilos específicos del componente.
    - `aceptar-solicitudes.component.html`: Plantilla HTML del componente.
    - `aceptar-solicitudes.component.spec.ts`: Pruebas unitarias del componente.
    - `aceptar-solicitudes.component.ts`: Lógica del componente.
  - `actividades`
    - `actividades.component.css`
    - `actividades.component.html`
    - `actividades.component.spec.ts`
    - `actividades.component.ts`
  - Y así sucesivamente para cada componente...

3. Modelos

En el directorio `models` se encuentran las definiciones de los modelos de datos que utiliza la aplicación:

- **Archivos de modelos**:
  - `Actividades.ts`
  - `Consumidores.ts`
  - `DTO/ActividadesRequestDTO.ts`: DTO (Data Transfer Object) específico para solicitudes de actividades.
  - Otros modelos como `Ofertantes.ts`, `Reseñas.ts`, `ReviewPelado.ts`, etc.

4. Servicios

En el directorio `services` se encuentran los servicios que proporcionan funcionalidades específicas y que son utilizados por los componentes para interactuar con la lógica de negocio y los datos:

- **Archivos de servicios**:
  - `actividades.service.ts`: Proporciona funcionalidades relacionadas con actividades.
  - `auth.service.ts`: Maneja la autenticación.
  - `consumidores.service.ts`: Proporciona funcionalidades para manejar consumidores.
  - `ofertantes.service.ts`: Proporciona funcionalidades para manejar ofertantes.
  - `tipo-actividades.service.ts`: Proporciona funcionalidades para manejar tipos de actividades.
  - `usuario.service.ts`: Proporciona funcionalidades para manejar usuarios.

4. Desarrollo del Backend

Endpoint: Listar todas las actividades

- **Método HTTP:** GET
- **URL:** /api/v1/actividades
- **Parámetros:** Ninguno

Endpoint: Listar actividades por tipo

- **Método HTTP:** GET
- **URL:** /api/v1/actividades/{tipo}
- **Parámetros:**
    - tipo (Path Variable): El tipo de actividades que se desean listar.

Endpoint: Eliminar actividad por ID

- **Método HTTP:** DELETE
- **URL:** /api/v1/actividades/{id}
- **Parámetros:**
    - id (Path Variable): El ID de la actividad que se desea eliminar.

Endpoint: Listar todos los consumidores

- **Método HTTP:** GET
- **URL:** /api/v1/consumidores
- **Parámetros:** Ninguno

Endpoint: Crear un nuevo consumidor

- **Método HTTP:** POST
- **URL:** /api/v1/consumidores
- **Parámetros:**
    - @RequestBody Consumidores consumidor: El objeto Consumidores en el cuerpo de la petición.

Endpoint: Obtener un consumidor por ID

- **Método HTTP:** GET
- **URL:** /api/v1/consumidores/{id}
- **Parámetros:**
    - id (Path Variable): El ID del consumidor.

Endpoint: Obtener un consumidor por nombre de usuario

- **Método HTTP:** GET
- **URL:** /api/v1/consumidores/usuario/{user}
- **Parámetros:**
    - user (Path Variable): El nombre de usuario del consumidor.

Endpoint: Actualizar un consumidor por ID

- **Método HTTP:** PUT
- **URL:** /api/v1/consumidores/{id}
- **Parámetros:**
    - id (Path Variable): El ID del consumidor.
    - @RequestBody Consumidores detallesConsumidor: El objeto Consumidores con los detalles actualizados en el cuerpo de la petición.

Endpoint: Eliminar un consumidor por ID

- **Método HTTP:** DELETE
- **URL:** /api/v1/consumidores/{id}
- **Parámetros:**
    - id (Path Variable): El ID del consumidor.

Endpoint: Obtener solicitudes de actividades por ID de consumidor

- **Método HTTP:** GET
- **URL:** /api/v1/consumidores/{idConsumidor}/solicitudes-actividades
- **Parámetros:**
    - idConsumidor (Path Variable): El ID del consumidor.

Endpoint: Enviar solicitud de actividad

- **Método HTTP:** POST
- **URL:** /api/v1/consumidores/{idConsumidor}/solicitudes-actividades/{idActividad}
- **Parámetros:**
    - idConsumidor (Path Variable): El ID del consumidor.
    - idActividad (Path Variable): El ID de la actividad.

Endpoint: Cancelar solicitud de actividad

- **Método HTTP:** DELETE
- **URL:** /api/v1/consumidores/{idConsumidor}/solicitudes-actividades/{idSolicitud}
- **Parámetros:**
    - idConsumidor (Path Variable): El ID del consumidor.
    - idSolicitud (Path Variable): El ID de la solicitud.

Endpoint: Publicar reseña de actividad

- **Método HTTP:** POST
- **URL:** /api/v1/consumidores/{idConsumidor}/opinion-actividades/{idActividad}
- **Parámetros:**
    - idConsumidor (Path Variable): El ID del consumidor.
    - idActividad (Path Variable): El ID de la actividad.

    - `@RequestBody Reseñas reviewSend`: El objeto `Reseñas` en el cuerpo de la petición.

Endpoint: Obtener reseñas por ID de consumidor

- **Método HTTP:** GET
- **URL:** /api/v1/consumidores/{idConsumidor}/opinion-actividades
- **Parámetros:**
    - `idConsumidor` (Path Variable): El ID del consumidor.

Endpoint: Eliminar reseña por ID

- **Método HTTP:** DELETE
- **URL:** /api/v1/reseñas/{idReseña}
- **Parámetros:**
    - `idReseña` (Path Variable): El ID de la reseña.

Endpoint: Listar todos los ofertantes

- **Método HTTP:** GET
- **URL:** /api/v1/ofertantes
- **Parámetros:** Ninguno

Endpoint: Crear un nuevo ofertante

- **Método HTTP:** POST
- **URL:** /api/v1/ofertantes
- **Parámetros:**
    - @RequestBody Ofertantes ofertante: El objeto Ofertantes en el cuerpo de la petición.

Endpoint: Obtener un ofertante por ID

- **Método HTTP:** GET
- **URL:** /api/v1/ofertantes/{id}
- **Parámetros:**
    - id (Path Variable): El ID del ofertante.

Endpoint: Actualizar un ofertante por ID

- **Método HTTP:** PUT
- **URL:** /api/v1/ofertantes/{id}
- **Parámetros:**
    - id (Path Variable): El ID del ofertante.
    - @RequestBody Ofertantes detallesOf: El objeto Ofertantes con los detalles actualizados en el cuerpo de la petición.

Endpoint: Eliminar un ofertante por ID

- **Método HTTP:** DELETE
- **URL:** /api/v1/ofertantes/{id}
- **Parámetros:**
    - id (Path Variable): El ID del ofertante.

Endpoint: Obtener una actividad específica de un ofertante

- **Método HTTP:** GET
- **URL:** /api/v1/ofertantes/{ofertanteId}/actividades/{actividadId}
- **Parámetros:**
    - ofertanteId (Path Variable): El ID del ofertante.
    - actividadId (Path Variable): El ID de la actividad.

Endpoint: Crear una nueva actividad para un ofertante

- **Método HTTP:** POST
- **URL:** /api/v1/ofertantes/{ofertanteId}/actividades
- **Parámetros:**
    - ofertanteId (Path Variable): El ID del ofertante.
    - @RequestBody ActividadesRequestDTO actividadDTO: El objeto ActividadesRequestDTO en el cuerpo de la petición.

Endpoint: Eliminar una actividad de un ofertante

- **Método HTTP:** DELETE
- **URL:** /api/v1/ofertantes/{ofertanteId}/actividades/{actividadId}
- **Parámetros:**
    - ofertanteId (Path Variable): El ID del ofertante.
    - actividadId (Path Variable): El ID de la actividad.

Endpoint: Actualizar una actividad de un ofertante

- **Método HTTP:** PUT
- **URL:** /api/v1/ofertantes/{ofertanteId}/actividades/{actividadId}
- **Parámetros:**
    - ofertanteId (Path Variable): El ID del ofertante.
    - actividadId (Path Variable): El ID de la actividad.
    - @RequestBody ActividadesRequestDTO actividadDTO: El objeto ActividadesRequestDTO en el cuerpo de la petición.

Endpoint: Obtener solicitudes de actividades por ofertante

- **Método HTTP:** GET

- **URL:** /api/v1/ofertantes/{ofertanteId}/solicitudes-actividades
- **Parámetros:**
    - ofertanteId (Path Variable): El ID del ofertante.

Endpoint: Aceptar una solicitud de actividad por ID

- **Método HTTP:** PUT
- **URL:** /api/v1/ofertantes/{ofertanteId}/solicitudes-actividades/{idSolicitud}
- **Parámetros:**
    - ofertanteId (Path Variable): El ID del ofertante.
    - idSolicitud (Path Variable): El ID de la solicitud de actividad.

Endpoint: Obtener reseñas por ID del ofertante

- **Método HTTP:** GET
- **URL:** /api/v1/ofertantes/{ofertanteId}/opinion-actividades
- **Parámetros:**
    - ofertanteId (Path Variable): El ID del ofertante.

Endpoint: Obtener ofertante por nombre de usuario

- **Método HTTP:** GET
- **URL:** /api/v1/ofertantes/usuario/{user}
- **Parámetros:**
    - user (Path Variable): El nombre de usuario.

Endpoint: Listar tipos de actividad

- **Método HTTP:** GET
- **URL:** /api/v1/tipos
- **Parámetros:** Ninguno

Endpoint: Guardar tipo de actividad

- **Método HTTP:** POST
- **URL:** /api/v1/tipos
- **Parámetros:**
    - Datos del tipo de actividad en el cuerpo de la solicitud (en formato JSON).

Endpoint: Obtener tipo de actividad por ID

- **Método HTTP:** GET
- **URL:** /api/v1/tipos/{id}
- **Parámetros:**
    - id (Path Variable): El ID del tipo de actividad.

Endpoint: Actualizar tipo de actividad por ID

- **Método HTTP:** PUT
- **URL:** /api/v1/tipos/{id}
- **Parámetros:**
    - id (Path Variable): El ID del tipo de actividad.
    - Datos actualizados del tipo de actividad en el cuerpo de la solicitud (en formato JSON).

Endpoint: Eliminar tipo de actividad por ID

- **Método HTTP:** DELETE
- **URL:** /api/v1/tipos/{id}
- **Parámetros:**
    - id (Path Variable): El ID del tipo de actividad.

Endpoint: Listar usuarios

- **Método HTTP:** GET
- **URL:** /api/v1/usuarios
- **Parámetros:** Ninguno

Endpoint: Obtener usuario por username

- **Método HTTP:** GET
- **URL:** /api/v1/usuarios/{username}
- **Parámetros:**
    - username (Path Variable): El nombre de usuario del usuario.

Endpoint: Registrar usuario, consumidor y ofertante

- **Método HTTP:** POST
- **URL:** /api/v1/usuarios
- **Parámetros:**
    - Datos del usuario en el cuerpo de la solicitud (en formato JSON).

Endpoint: Iniciar sesión

- **Método HTTP:** POST
- **URL:** /api/v1/usuarios/iniciar-sesion
- **Parámetros:**
    - user (Request Parameter): El nombre de usuario.
    - password (Request Parameter): La contraseña.
    - tipoSesion (Request Parameter): El tipo de sesión (1 para consumidor, 2 para ofertante).

Endpoint: Eliminar usuario por ID

- **Método HTTP:** DELETE
- **URL:** /api/v1/usuarios/{id}
- **Parámetros:**
    - id (Path Variable): El ID del usuario a eliminar.

Endpoint: Actualizar usuario por ID

- **Método HTTP:** PUT
- **URL:** /api/v1/usuarios/{id}
- **Parámetros:**
    - id (Path Variable): El ID del usuario a actualizar.
    - Datos actualizados del usuario en el cuerpo de la solicitud (en formato JSON).

5. Desarrollo del frontend

Components

AceptarSolicitudesComponent

Propiedades:

- `ofertanteId`: Almacena el ID del ofertante relacionado con la solicitud.
- `idSolicitud`: Almacena el ID de la solicitud que se está aceptando.
- `usuario`: Almacena el nombre de usuario actual. Inicialmente vacío, pero se obtiene del servicio `AuthService` en `ngOnInit()`.
- `mensaje`: Una variable que podría contener algún mensaje relacionado con la aceptación de la solicitud, pero en este código no parece utilizarse.
- `confirmado`: Un indicador booleano que parece destinado a registrar si la solicitud ha sido confirmada. Sin embargo, no se utiliza en el código proporcionado.

2. **Constructor:**

- Se inyectan instancias de `ActivatedRoute`, `Router`, `OfertantesService`, y `AuthService` para su uso dentro del componente.

3. **ngOnInit():**

- Se ejecuta cuando el componente está inicializado.
- Obtiene `ofertanteId` e `idSolicitud` de los parámetros de la URL.
- Obtiene el nombre de usuario actual del servicio de autenticación y lo almacena en `usuario`.
- Imprime los valores de `ofertanteId`, `idSolicitud` y `usuario` en la consola.

4. **aceptarSolicitud(ofertanteId, idSolicitud):**

- Método invocado cuando se desea aceptar una solicitud.
- Utiliza el servicio `OfertantesService` para enviar una solicitud de aceptación al servidor.
- Si la solicitud se completa con éxito, imprime la respuesta en la consola y luego redirige al usuario a la página de inicio (`volveraHome()`).
- Si hay un error, imprime el error en la consola.

5. **volveraHome():**

- Redirige al usuario a la página de inicio de los ofertantes, probablemente pasando el nombre de usuario como parámetro en la URL.

ActividadesComponent

Propiedades:

- `actividades`: Almacena una lista de actividades.
- `tiposActividades`: Almacena una lista de tipos de actividades.
- `filtroTipo`: Almacena el nombre del tipo de actividad utilizado para filtrar las actividades.
- `idConsumidor`: Almacena el ID del consumidor actual.
- `ofertantes`: Almacena una lista de ofertantes.

2. **Constructor:**

- Se inyectan instancias de `Router`, `ActividadesService`, `TipoActividadesService`, `AuthService` y `OfertantesService`.

3. **ngOnInit():**

- Se ejecuta cuando el componente está inicializado.
- Invoca métodos para listar actividades, tipos de actividades, obtener el ID del consumidor y listar ofertantes.

4. **listarOfertantes():**

- Utiliza el servicio `OfertantesService` para obtener la lista de ofertantes y la asigna a la propiedad `ofertantes`.
- Imprime la lista de ofertantes en la consola.

5. **obtenerIdConsumidor():**

- Utiliza el servicio `AuthService` para obtener el ID del consumidor del almacenamiento local y lo asigna a `idConsumidor`.
- Imprime el ID del consumidor en la consola.

6. **listarActividades():**

- Utiliza el servicio `ActividadesService` para obtener la lista de actividades y la asigna a `actividades`.
- Imprime la lista de actividades en la consola.

7. **listarTipoActividades():**

- Utiliza el servicio `TipoActividadesService` para obtener la lista de tipos de actividades y la asigna a `tiposActividades`.
- Imprime la lista de tipos de actividades en la consola.

8. **filtrarPorTipo(event: any):**

- Filtra las actividades por el tipo seleccionado por el consumidor.
- Si se selecciona un tipo, utiliza el servicio `ActividadesService` para obtener las actividades filtradas por ese tipo y las asigna a `actividades`.
- Si se deselecciona el tipo, vuelve a listar todas las actividades.

9. **irASolicitudActividad(idConsumidor: number, idActividad: number):**

- Navega a la página de solicitud de actividad, pasando el ID del consumidor y el ID de la actividad como parámetros en la URL.

ConsumidorDataComponent
Propiedades:

- `consumidorId`: Almacena el ID del consumidor cuyos datos se están visualizando.
- `consumidor$`: Almacena los datos del consumidor recuperados del servicio. Inicialmente es `null`.
- `reviews$`: Almacena las reseñas asociadas al consumidor. Inicialmente es `null`.

2. **Constructor:**

- Se inyectan instancias de `ActivatedRoute` y `ConsumidoresService`.

3. **ngOnInit():**

- Se ejecuta cuando el componente está inicializado.
- Suscribe el componente a los cambios en los parámetros de la URL para obtener el ID del consumidor.
- Llama a los métodos `obtenerConsumidorPorId()` y `obtenerReseñasPorIdConsumidor()` pasando el ID del consumidor como parámetro.

4. **obtenerConsumidorPorId(id: number):**

- Utiliza el servicio `ConsumidoresService` para obtener los datos del consumidor por su ID.
- Asigna los datos del consumidor a `consumidor$`.
- Imprime los datos del consumidor en la consola.

5. **obtenerReseñasPorIdConsumidor(id: number):**

- Utiliza el servicio `ConsumidoresService` para obtener las reseñas asociadas al consumidor por su ID.
- Asigna las reseñas a `reviews$`.
- Imprime las reseñas en la consola.

CrearActividadComponent

Propiedades:

- `ofertanteId`: Almacena el ID del ofertante para el cual se está creando la actividad.
- `actividadForm`: Es un objeto de tipo `ActividadesRequestDTO` que contiene los datos de la actividad a crear, como nombre, descripción, fechas, precio, materiales y ID del tipo de actividad.
- `tiposActividades`: Almacena una lista de los tipos de actividades disponibles.

2. **Constructor:**

- Se inyectan instancias de `ActivatedRoute`, `Router`, `OfertantesService` y `TipoActividadesService`.

3. **ngOnInit():**

- Se ejecuta al inicializarse el componente.
- Obtiene el ID del ofertante de los parámetros de la URL.
- Carga los tipos de actividades disponibles utilizando el servicio `TipoActividadesService` y los almacena en `tiposActividades`.

4. **onSubmit():**

- Invocado cuando se envía el formulario para crear una nueva actividad.
- Utiliza el servicio `OfertantesService` para crear la actividad para el ofertante especificado.
- Una vez que la actividad se crea con éxito, imprime la actividad creada en la consola y luego redirige al usuario a la lista de actividades del ofertante.

5. **irAactividades():**

- Método utilizado para redirigir al usuario a la lista de actividades del ofertante después de que se haya creado la actividad.

CrearReviewComponent
Propiedades:

- `idConsumidor`: Almacena el ID del consumidor que está creando la reseña.
- `idActividad`: Almacena el ID de la actividad sobre la cual se está dejando la reseña.
- `usuario`: Almacena el nombre de usuario del consumidor.
- `review`: Es un objeto de tipo `ReviewPelado` que contiene los detalles de la reseña, como puntos, comentario y fecha.

2. **Constructor:**

- Se inyectan instancias de `ActivatedRoute`, `Router` y `ConsumidoresService`.

3. **ngOnInit():**

- Se ejecuta al inicializarse el componente.
- Obtiene el ID del consumidor, el ID de la actividad y el nombre de usuario de los parámetros de la URL.
- Inicializa el objeto de revisión y muestra información relevante en la consola para propósitos de depuración.

4. **crearYPublicarReview():**

- Invocado para crear y publicar la reseña utilizando el servicio `ConsumidoresService`.
- Llama al método `publicarReview` del servicio, pasando el ID del consumidor, el ID de la actividad y los detalles de la reseña.
- Una vez que la reseña se publica con éxito, imprime la revisión publicada en la consola y redirige al consumidor a la página de inicio.

5. **onSubmit():**

- Invocado cuando se envía el formulario para crear y publicar la reseña.
- Simplemente llama al método `crearYPublicarReview()`.

6. **irAHome():**

- Método utilizado para redirigir al consumidor a la página de inicio después de publicar la reseña.

HomeComponent

Constructor:

- Se inyectan instancias de `Router` y `ActivatedRoute`.

2. **`ngOnInit()`:**

- Este método se ejecuta cuando el componente se inicializa, pero en este caso no se realiza ninguna operación en particular en esta etapa.

3. **`register()`:**

- Este método se utiliza para redirigir a los usuarios a la página de registro (`/sign-up`).
- Cuando se llama a este método, se utiliza el servicio `Router` para navegar a la ruta de registro.

4. **`login()`:**

- Este método se utiliza para redirigir a los usuarios a la página de inicio de sesión (`/login`).
- Cuando se llama a este método, se utiliza el servicio `Router` para navegar a la ruta de inicio de sesión.

HomeConsumidorComponent

Propiedades:

- `consumidor`: Almacena los datos del consumidor actual.
- `usuario`: Almacena el nombre de usuario del consumidor actual.
- `idConsumidor`: Almacena el ID del consumidor actual.
- `idActividad`: Almacena el ID de la actividad seleccionada.
- `solicitudesActividades`: Almacena las solicitudes de actividades del consumidor.

2. **Constructor:**

- Se inyectan instancias de varios servicios, incluidos `UsuarioService`, `ConsumidoresService` y `AuthService`, así como también `Router` y `ActivatedRoute`.

3. **ngOnInit():**

- Se ejecuta cuando el componente se inicializa.
- Obtiene el nombre de usuario de los parámetros de la URL y realiza una serie de operaciones, como obtener los datos del consumidor, guardar y obtener el ID del consumidor del almacenamiento local y obtener las solicitudes de actividades del consumidor.

4. **obtenerConsumidorPorUsername(user: string):**

- Utiliza el servicio `ConsumidoresService` para obtener los datos del consumidor por su nombre de usuario.
- Almacena los datos del consumidor en `consumidor`.

5. **guardarIdConsumidorLocalStorage(user: string):**

- Utiliza el servicio `AuthService` para guardar el ID del consumidor en el almacenamiento local.

6. **obtenerIdConsumidorLocalStorage():**

- Utiliza el servicio `AuthService` para obtener el ID del consumidor del almacenamiento local.

7. **obtenerSolicitudesActividadesPorConsumidor(idConsumidor: number):**

- Utiliza el servicio `ConsumidoresService` para obtener las solicitudes de actividades del consumidor por su ID.
- Almacena las solicitudes de actividades en `solicitudesActividades`.

8. **escribirReview(idConsumidor: number, idActividad: number):**

    - Redirige al consumidor a la página para escribir una reseña sobre una actividad específica.

9. **actualizarUsuario():**

    - Redirige al consumidor a la página para actualizar su perfil.

10. **eliminarUsuarioPorId(id: number):**

- Utiliza el servicio `UsuarioService` para eliminar el usuario por su ID.
- Una vez que el usuario se elimina con éxito, redirige al consumidor a la página principal.

HomeOfertanteComponent

Propiedades:

- `ofertante`: Almacena los datos del ofertante actual.
- `usuario`: Almacena el nombre de usuario del ofertante actual.
- `idOfertante`: Almacena el ID del ofertante actual.
- `solicitudesActividades`: Almacena las solicitudes de actividades recibidas por el ofertante.

2. **Constructor:**

- Se inyectan instancias de varios servicios, incluidos `UsuarioService`, `OfertantesService` y `AuthService`, así como también `Router` y `ActivatedRoute`.

3. **ngOnInit():**

- Se ejecuta cuando el componente se inicializa.
- Obtiene el nombre de usuario de los parámetros de la URL y realiza una serie de operaciones, como obtener los datos del ofertante, guardar y obtener el ID del ofertante del almacenamiento local, y obtener las solicitudes de actividades recibidas por el ofertante.

4. **obtenerOfertantePorUsername(user: string):**

- Utiliza el servicio `OfertantesService` para obtener los datos del ofertante por su nombre de usuario.
- Almacena los datos del ofertante en `ofertante`.

5. **guardarIdOfertanteLocalStorage(user: string):**

- Utiliza el servicio `AuthService` para guardar el ID del ofertante en el almacenamiento local.

6. **obtenerIdOfertanteLocalStorage():**

- Utiliza el servicio `AuthService` para obtener el ID del ofertante del almacenamiento local.

7. **obtenerSolicitudesActividadesPorOfertante(idConsumidor: number):**

- Utiliza el servicio `OfertantesService` para obtener las solicitudes de actividades recibidas por el ofertante por su ID.
- Almacena las solicitudes de actividades en `solicitudesActividades`.

8. **redirigirAceptarSolicitud(ofertanteId: number, idSolicitud: number):**

- Redirige al ofertante a la página para aceptar una solicitud específica.

9. **actualizarUsuario():**

- Redirige al ofertante a la página para actualizar su perfil.

10. **eliminarUsuarioPorId(id: number):**

- Utiliza el servicio `UsuarioService` para eliminar el usuario por su ID.
- Una vez que el usuario se elimina con éxito, redirige al ofertante a la página principal.

ListaActividadesOfertanteComponent

Propiedades:

- `usuario`: Almacena el nombre de usuario del ofertante.
- `ofertante`: Almacena los datos del ofertante actual.
- `review$`: Almacena las reseñas asociadas al ofertante.

2. **Constructor:**

- Se inyectan instancias de `ActivatedRoute`, `Router`, `OfertantesService` y `AuthService`.

3. **ngOnInit():**

- Se ejecuta cuando el componente se inicializa.
- Obtiene el nombre de usuario del ofertante de la sesión y llama al método `obtenerOfertantePorUsername()` para obtener los datos del ofertante.

4. **obtenerOfertantePorUsername(user: string):**

- Utiliza el servicio `OfertantesService` para obtener los datos del ofertante por su nombre de usuario.
- Almacena los datos del ofertante en `ofertante`.
- Llama al método `obtenerReseñasPorOfertanteId()` para obtener las reseñas asociadas al ofertante.

5. **obtenerReseñasPorOfertanteId(id: number):**

- Utiliza el servicio `OfertantesService` para obtener las reseñas asociadas al ofertante por su ID.
- Almacena las reseñas en `review$`.

6. **irAcorregirActividad(idOfertante: number, idActividad: number):**

- Redirige al ofertante a la página para corregir una actividad específica.

7. **eliminarActividad(ofertanteId: number, actividadId: number):**

- Utiliza el servicio `OfertantesService` para eliminar una actividad del ofertante por su ID de ofertante y el ID de la actividad.
- Una vez que la actividad se elimina con éxito, vuelve a llamar al método `obtenerOfertantePorUsername()` para actualizar la lista de actividades del ofertante.

ListaReviewsComponent

Propiedades:

- `idConsumidor`: Almacena el ID del consumidor.
- `consumidor`: Almacena los datos del consumidor actual.
- `userId`: Almacena el ID del usuario.
- `resenas`: Almacena las reseñas asociadas al consumidor.

2. **Constructor:**

- Se inyectan instancias de `ConsumidoresService`, `AuthService`, `Router` y `ActivatedRoute`.

3. **ngOnInit():**

- Se ejecuta cuando el componente se inicializa.
- Obtiene el ID del usuario de los parámetros de la URL y llama a los métodos `obtenerIdConsumidorLocalStorage()` y `obtenerResenasPorIdConsumidor()`.

4. **obtenerIdConsumidorLocalStorage():**

- Utiliza el servicio `AuthService` para obtener el ID del consumidor del almacenamiento local.

5. **obtenerResenasPorIdConsumidor():**

- Utiliza el servicio `ConsumidoresService` para obtener las reseñas asociadas al consumidor por su ID.
- Almacena las reseñas en `resenas`.

6. **borrarReview(idReseña: number):**

- Utiliza el servicio `ConsumidoresService` para eliminar una reseña por su ID.
- Una vez que la reseña se elimina con éxito, vuelve a llamar al método `obtenerResenasPorIdConsumidor()` para actualizar la lista de reseñas.

LoginComponent

Propiedades:

- `user`: Almacena el nombre de usuario ingresado por el usuario.
- `password`: Almacena la contraseña ingresada por el usuario.
- `tipoSesion`: Almacena el tipo de sesión, que indica si el usuario es un consumidor (tipoSesion '1') o un ofertante (tipoSesion '2').

2. **Constructor:**

- Se inyectan instancias de `UsuarioService`, `Router`, `ActivatedRoute` y `AuthService`.

3. **iniciarSesion():**

- Este método se llama cuando el usuario intenta iniciar sesión.
- Utiliza el servicio `UsuarioService` para iniciar sesión con el nombre de usuario, la contraseña y el tipo de sesión proporcionados.
- Si el inicio de sesión es exitoso, guarda el nombre de usuario en la sesión actual utilizando el servicio `AuthService`.
- Redirige al usuario a la página de inicio correspondiente según el tipo de sesión:
    - Si el tipo de sesión es '1', significa que el usuario es un consumidor, por lo que lo redirige a la página de inicio de consumidores.
    - Si el tipo de sesión es '2', significa que el usuario es un ofertante, por lo que lo redirige a la página de inicio de ofertantes.

NavbarComponent

Propiedades:

- `nombreUsuario`: Almacena el nombre de usuario actualmente en sesión.
- `idOfertante`: Almacena el ID del ofertante actualmente en sesión.
- `rolUsuario`: Almacena el rol del usuario actualmente en sesión.

2. **Constructor:**

- Se inyectan instancias de `Router` y `AuthService`.
- En el constructor se inicializan las propiedades `nombreUsuario`, `rolUsuario` e `idOfertante` con los valores obtenidos del servicio `AuthService`.

3. **ngOnInit():**

- Este método se llama cuando el componente se inicializa, pero en este caso está vacío ya que no se realizan operaciones adicionales.

4. **Métodos de redirección:**

- `irAConsumidores()`: Redirige al usuario a la página de consumidores si su rol es "CONSUMER", o a la página de ofertantes si su rol es "PROVIDER".
- `irAActividades()`: Redirige al usuario a la página de actividades si su rol es "CONSUMER", o a la página de lista de actividades del ofertante si su rol es "PROVIDER".
- `irAHome()`: Redirige al usuario a su página de inicio correspondiente según su rol.
- `irACrearActividad()`: Redirige al usuario a la página de creación de actividad del ofertante.
- `irAReviews()`: Redirige al usuario a la página de lista de reseñas de un consumidor.
- `cerrarSesion()`: Cierra la sesión del usuario utilizando el servicio `AuthService` y redirige al usuario a la página de inicio.

OfertanteDataComponent

Propiedades:

- `ofertanteId`: Almacena el ID del ofertante cuyos detalles se están mostrando.
- `ofertante$`: Almacena los datos del ofertante.
- `review$`: Almacena las reseñas asociadas al ofertante.

2. **Constructor:**

- Se inyectan instancias de `ActivatedRoute` y `OfertantesService`.

3. **ngOnInit():**

- Este método se llama cuando el componente se inicializa.
- Utiliza el servicio `ActivatedRoute` para obtener el ID del ofertante de los parámetros de la URL.
- Llama al método `obtenerOfertantePorId()` para obtener los detalles del ofertante.

4. **obtenerOfertantePorId(id: number):**

- Utiliza el servicio `OfertantesService` para obtener los detalles del ofertante por su ID.
- Una vez que se obtienen los detalles del ofertante con éxito, llama al método `obtenerReseñasPorOfertanteId()` para obtener las reseñas asociadas a ese ofertante.

5. **obtenerReseñasPorOfertanteId(id: number):**

- Utiliza el servicio `OfertantesService` para obtener las reseñas asociadas a un ofertante por su ID.
- Almacena las reseñas obtenidas en la propiedad `review$`.

OfertantesComponent

Propiedades:

- `ofertantes`: Almacena la lista de ofertantes que se mostrará en la interfaz de usuario.

2. **Constructor:**

- Se inyectan instancias de `Router` y `OfertantesService`.

3. **ngOnInit():**

- Este método se llama cuando el componente se inicializa.
- Llama al método `listarOfertantes()` para obtener la lista de ofertantes.

4. **listarOfertantes():**

- Utiliza el servicio `OfertantesService` para obtener la lista de ofertantes.
- Almacena la lista de ofertantes en la propiedad `ofertantes`.
- Imprime la lista de ofertantes en la consola para fines de depuración.

5. **verDetalles(id: number):**

- Este método se utiliza para redirigir al usuario a la página de detalles de un ofertante específico.
- Utiliza el servicio `Router` para navegar a la ruta correspondiente, pasando el ID del ofertante como parámetro en la URL.

SignupComponent

Propiedades:

- `usuario`: Almacena los datos del usuario que se va a registrar.
- `textoBoton`: Almacena el texto que se mostrará en el botón de envío del formulario.
- `tituloForm`: Almacena el título que se mostrará en el formulario de registro.

2. **Constructor:**

- Se inyectan instancias de `UsuarioService`, `Router` y `ActivatedRoute`.
- Inicializa las propiedades `usuario`, `textoBoton` y `tituloForm`.

3. **guardarUsuario():**

- Este método se utiliza para enviar la solicitud de registro del usuario al servidor a través del servicio `UsuarioService`.
- Suscribe al observable devuelto por `addUsuario()` para manejar la respuesta del servidor.
- Si el registro es exitoso, imprime los datos del usuario agregado en la consola y redirige al usuario a la página de inicio.
- En caso de error, imprime el error en la consola.

4. **volverLista():**

- Redirige al usuario a la página de inicio.

5. **onSubmit(usuario: Usuarios):**

- Este método se llama cuando se envía el formulario de registro.
- Comprueba si el ID del usuario es `-1`, lo que indica que es un nuevo usuario que aún no ha sido registrado.
- Si es así, llama al método `guardarUsuario()` para registrar al usuario.

SolicitudesComponent

Propiedades:

- `idConsumidor`: Almacena el ID del consumidor que está realizando la solicitud.
- `idActividad`: Almacena el ID de la actividad para la cual se está realizando la solicitud.
- `mensaje`: Almacena un mensaje para informar al usuario sobre el resultado de la solicitud.
- `confirmado`: Indica si la solicitud ha sido confirmada o no.

2. **Constructor:**

- Se inyectan instancias de `ActivatedRoute`, `Router` y `ConsumidoresService`.

3. **ngOnInit():**

- Obtiene los IDs del consumidor y la actividad de los parámetros de la ruta.

4. **confirmarSolicitud():**

- Este método se llama cuando el usuario confirma la solicitud.
- Establece la propiedad `confirmado` en `true`.
- Llama al método `enviarSolicitud()` para enviar la solicitud al servidor.

5. **cancelar():**

- Este método se llama cuando el usuario decide cancelar la solicitud.
- Redirige al usuario de vuelta a la lista de actividades.

6. **enviarSolicitud():**

- Este método envía la solicitud de actividad al servidor a través del servicio `ConsumidoresService`.
- Suscribe al observable devuelto por `enviarSolicitudActividad()` para manejar la respuesta del servidor.
- Si la solicitud se envía con éxito, muestra un mensaje indicando que la solicitud se ha enviado correctamente y redirige al usuario de vuelta a la lista de actividades después de 2 segundos.
- En caso de error, muestra un mensaje de error y registra el error en la consola.

UpdateActividadComponent

Propiedades:

- `ofertanteId`: Almacena el ID del ofertante que está actualizando la actividad.
- `idActividad`: Almacena el ID de la actividad que se va a actualizar.
- `actividadForm`: Un objeto de tipo `ActividadesRequestDTO` que contiene los detalles de la actividad que se va a actualizar.

2. **Constructor:**

- Se inyectan instancias de `ActivatedRoute`, `Router` y `OfertantesService`.

3. **ngOnInit():**

- Obtiene los IDs del ofertante y de la actividad de los parámetros de la ruta.
- Utiliza estos IDs para llamar al método `obtenerActividadPorId()` del servicio `OfertantesService` y obtener los detalles de la actividad que se va a actualizar.
- Los detalles de la actividad se asignan al objeto `actividadForm`.

4. **onSubmit():**

- Este método se llama cuando el usuario envía el formulario de actualización.
- Llama al método `actualizarActividadOfertante()` del servicio `OfertantesService` para actualizar la actividad con los nuevos detalles proporcionados en `actividadForm`.
- Una vez que la actividad se actualiza con éxito, muestra un mensaje indicando que la actividad se ha actualizado correctamente y redirige al usuario de vuelta a la lista de actividades de ofertante después de 2 segundos.

5. **irAactividades():**

- Este método se encarga de redirigir al usuario de vuelta a la lista de actividades de ofertante después de actualizar la actividad.

UpdateUsuarioComponent

Propiedades:

- `usuario`: Almacena el nombre de usuario del usuario cuya información se está actualizando.
- `rolUsuario`: Almacena el rol del usuario en sesión.
- `usuarioUpdate`: Un objeto de tipo `Usuarios` que contiene los detalles del usuario que se va a actualizar.

2. **Constructor:**

- Se inyectan instancias de `ActivatedRoute`, `Router`, `AuthService` y `UsuarioService`.
- Se inicializa la propiedad `rolUsuario` con el rol del usuario en sesión obtenido del servicio de autenticación.

3. **ngOnInit():**

- Obtiene el nombre de usuario del parámetro de la ruta.
- Llama al método `obtenerUsuarioPorUsername()` del servicio `UsuarioService` para obtener los detalles del usuario que se va a actualizar.
- Rellena el objeto `usuarioUpdate` con los datos obtenidos del usuario.

4. **onSubmit():**

- Este método se llama cuando el usuario envía el formulario de actualización.
- Llama al método `actualizarUsuario()` del servicio `UsuarioService` para actualizar el usuario con los nuevos detalles proporcionados en `usuarioUpdate`.
- Una vez que el usuario se actualiza con éxito, muestra un mensaje indicando que el usuario se ha actualizado correctamente y redirige al usuario de vuelta a la página de inicio correspondiente según su rol.

5. **irAHome():**

- Este método se encarga de redirigir al usuario de vuelta a la página de inicio correspondiente según su rol después de actualizar su información.

Services

ActividadesService

Constructor:

- Se inyecta la instancia de `HttpClient`.

2. **listarActividades():**

- Este método hace una solicitud HTTP GET para recuperar todas las actividades.
- Devuelve un observable que emite un array de objetos `Actividades`.

3. **listarActividadesPorTipo(tipo: string):**

- Este método hace una solicitud HTTP GET para recuperar actividades filtradas por tipo.
- Toma un parámetro `tipo` que especifica el tipo de actividad a filtrar.
- Devuelve un observable que emite un array de objetos `Actividades`.

4. **buscarActividadesPorNombre(busqueda: string):**

- Este método hace una solicitud HTTP GET para buscar actividades por nombre.
- Toma un parámetro `busqueda` que especifica el término de búsqueda.
- Devuelve un observable que emite un array de objetos `Actividades`.

5. **eliminarActividad(id: number):**

- Este método hace una solicitud HTTP DELETE para eliminar una actividad por su ID.
- Toma un parámetro `id` que especifica el ID de la actividad a eliminar.
- Devuelve un observable que emite un objeto con una propiedad `eliminar` que indica si la actividad se eliminó con éxito.

AuthService

Constructor:

- Se inyectan instancias de `ConsumidoresService` y `OfertantesService`.

2. **guardarNombreUsuarioEnSesion(nombreUsuario: string):**
   - Este método guarda el nombre de usuario en la sesión del navegador utilizando `sessionStorage`.
3. **obtenerNombreUsuarioEnSesion(): string | null:**
   - Este método recupera el nombre de usuario almacenado en la sesión del navegador.
4. **guardarRolUsuarioEnSesion(role: string):**
   - Guarda el rol de usuario en la sesión del navegador utilizando `sessionStorage`.
5. **obtenerRolUsuarioEnSesion(): string | null:**
   - Recupera el rol de usuario almacenado en la sesión del navegador.
6. **limpiarSesion():**
   - Borra todos los datos de la sesión del navegador.
7. **guardarIdConsumidorEnLocalStorage(user: string):**
   - Obtiene el ID del consumidor por su nombre de usuario y lo guarda en el almacenamiento local.
   - También guarda el rol del usuario en la sesión.
8. **guardarIdOfertanteEnLocalStorage(user: string):**
   - Obtiene el ID del ofertante por su nombre de usuario y lo guarda en el almacenamiento local.
   - También guarda el rol del usuario en la sesión.
9. **obtenerIdConsumidorDeLocalStorage(): number | null:**
   - Recupera el ID del consumidor almacenado en el almacenamiento local.
10. **obtenerIdOfertanteDeLocalStorage(): number | null:**
    - Recupera el ID del ofertante almacenado en el almacenamiento local.

ConsumidoresService

Constructor:

- Recibe una instancia de `HttpClient` para realizar solicitudes HTTP.

2. **obtenerConsumidorPorUsername(user: string): Observable<Consumidores>:**

- Obtiene un consumidor por su nombre de usuario.
- Realiza una solicitud HTTP GET a la URL correspondiente.

3. **listarConsumidores(): Observable<Consumidores[]>:**

- Obtiene la lista de todos los consumidores.
- Realiza una solicitud HTTP GET a la URL correspondiente.

4. **obtenerConsumidorPorId(id: number): Observable<Consumidores>:**

- Obtiene un consumidor por su ID.
- Realiza una solicitud HTTP GET a la URL correspondiente.

5. **enviarSolicitudActividad(idConsumidor: number, idActividad: number): Observable<SolicitudesActividades>:**

- Envía una solicitud de actividad por parte de un consumidor.
- Realiza una solicitud HTTP POST a la URL correspondiente.

6. **obtenerSolicitudesActividadesPorConsumidor(idConsumidor: number): Observable<SolicitudesActividades[]>:**

- Obtiene las solicitudes de actividades realizadas por un consumidor.
- Realiza una solicitud HTTP GET a la URL correspondiente.

7. **obtenerReseñaPorIDconsumidor(idConsumidor: number): Observable<Reseñas[]>:**

- Obtiene las reseñas realizadas por un consumidor.
- Realiza una solicitud HTTP GET a la URL correspondiente.

8. **borrarReseña(idReseña: number): Observable<{ eliminar: boolean }>:**

- Elimina una reseña por su ID.
- Realiza una solicitud HTTP DELETE a la URL correspondiente.

9. **publicarReview(idConsumidor: number, idActividad: number, review: ReviewPelado): Observable<ReviewPelado>:**

- Publica una reseña sobre una actividad realizada por un consumidor.

- Realiza una solicitud HTTP POST a la URL correspondiente.

OfertantesService

Constructor:

- Recibe una instancia de `HttpClient` para realizar solicitudes HTTP.

2. **obtenerOfertantePorUsername(user: string): Observable<Ofertantes>:**

- Obtiene un ofertante por su nombre de usuario.
- Realiza una solicitud HTTP GET a la URL correspondiente.

3. **obtenerSolicitudesActividadesPorOfertante(idOfertante: number): Observable<SolicitudesActividades[]>:**

- Obtiene las solicitudes de actividades realizadas por un ofertante.
- Realiza una solicitud HTTP GET a la URL correspondiente.

4. **listarOfertantes(): Observable<Ofertantes[]>:**

- Obtiene la lista de todos los ofertantes.
- Realiza una solicitud HTTP GET a la URL correspondiente.

5. **obtenerOfertantePorId(id: number): Observable<Ofertantes>:**

- Obtiene un ofertante por su ID.
- Realiza una solicitud HTTP GET a la URL correspondiente.

6. **obtenerReseñasPorOfertanteId(ofertanteId: number): Observable<Reseñas[]>:**

- Obtiene las reseñas realizadas sobre las actividades de un ofertante.
- Realiza una solicitud HTTP GET a la URL correspondiente.

7. **aceptarSolicitudPorId(ofertanteId: number, idSolicitud: number): Observable<{ aceptado: boolean }>:**

- Acepta una solicitud de actividad por parte de un consumidor.
- Realiza una solicitud HTTP PUT a la URL correspondiente.

8. **crearActividadParaOfertante(ofertanteId: number, actividadDTO: ActividadesRequestDTO): Observable<Actividades>:**

- Crea una nueva actividad para un ofertante.
- Realiza una solicitud HTTP POST a la URL correspondiente.

9. **actualizarActividadOfertante(ofertanteId: number, actividadId: number, actividadDTO: ActividadesRequestDTO): Observable<Actividades>:**

- Actualiza los detalles de una actividad perteneciente a un ofertante.

- Realiza una solicitud HTTP PUT a la URL correspondiente.

10. **obtenerActividadPorId(ofertanteId: number, actividadId: number): Observable<ActividadesRequestDTO>:**

    - Obtiene los detalles de una actividad por su ID.
    - Realiza una solicitud HTTP GET a la URL correspondiente.

11. **eliminarActividadOfertante(ofertanteId: number, actividadId: number): Observable<{ eliminado: boolean }>:**

    - Elimina una actividad perteneciente a un ofertante por su ID.
    - Realiza una solicitud HTTP DELETE a la URL correspondiente.

TipoActividadesService

Constructor:

- Recibe una instancia de `HttpClient` para realizar solicitudes HTTP.

2. **listarTipoActividades(): Observable<TipoActividad[]>:**

- Obtiene la lista de todos los tipos de actividades disponibles.
- Realiza una solicitud HTTP GET a la URL correspondiente.

3. **obtenerTipoPorId(id: number): Observable<TipoActividad>:**

- Obtiene un tipo de actividad por su ID.
- Realiza una solicitud HTTP GET a la URL correspondiente.

UsuarioService

Constructor:

- Recibe una instancia de `HttpClient` para realizar solicitudes HTTP.

2. **getListaUsuarios(): Observable<Usuarios[]>:**

- Obtiene la lista de todos los usuarios disponibles.
- Realiza una solicitud HTTP GET a la URL correspondiente.

3. **addUsuario(usuario: Usuarios): Observable<Object>:**

- Agrega un nuevo usuario.
- Toma un objeto `Usuarios` como parámetro que contiene la información del nuevo usuario.
- Realiza una solicitud HTTP POST a la URL correspondiente.

4. **iniciarSesion(user: string, password: string, tipoSesion: string): Observable<any>:**

- Inicia sesión de usuario.
- Toma el nombre de usuario (`user`), la contraseña (`password`), y el tipo de sesión (`tipoSesion`) como parámetros.
- Realiza una solicitud HTTP POST a la URL correspondiente con los datos proporcionados.

5. **eliminarUsuario(id: number): Observable<any>:**

- Elimina un usuario por su ID.
- Toma el ID del usuario como parámetro.
- Realiza una solicitud HTTP DELETE a la URL correspondiente.

6. **actualizarUsuario(id: number, usuario: Usuarios): Observable<any>:**

- Actualiza la información de un usuario por su ID.
- Toma el ID del usuario y un objeto `Usuarios` con la información actualizada como parámetros.
- Realiza una solicitud HTTP PUT a la URL correspondiente con los datos proporcionados.

7. **obtenerUsuarioPorUsername(username: string): Observable<Usuarios>:**

- Obtiene un usuario por su nombre de usuario.
- Toma el nombre de usuario como parámetro.

- Realiza una solicitud HTTP GET a la URL correspondiente con el nombre de usuario proporcionado.