

Trabajo Fin Grado

ENSAYOS

Aplicación para la gestión de la actividad de grupos musicales durante los ensayos



Autor: Jose Javier Bailón Ortiz

Fecha: Junio de 2024

Grado: Desarrollo de aplicaciones multiplataforma (22-24)

Centro: I.E.S. Padre Poveda (Guadix)

SUMARIO

1 – Descripción general y objetivo.....	4
2 – El proceso de desarrollo del proyecto.....	5
3 – Visión global de la solución.....	6
3.1 Requisitos funcionales.....	6
3.2 Arquitectura global y flujo de información.....	7
4 – El servidor.....	8
4.1 Descripción general y tecnologías usadas en el servidor.....	8
4.2 Arquitectura interna del servidor.....	8
4.2.1 Ejemplos del código.....	10
4.2.2 Diagramas de clases.....	13
4.3 Modelo y base de datos del servidor.....	16
4.3.1 Diseño conceptual de BD en el servidor.....	16
4.3.2 Diseño lógico de BD en el servidor.....	17
4.4 Casos de uso del servidor.....	18
4.5 Almacenamiento en el servidor.....	19
4.6 Endpoints del servicio web.....	19
Endpoints de autorización.....	20
Endpoints de grupos.....	21
Endpoints de canciones.....	22
Endpoints de notas.....	23
Endpoints de audios.....	24
Otros endpoints.....	26
4.7 Pruebas realizadas.....	28
4.8 Requisitos y manual de despliegue.....	29
4.8.1 Configuración y despliegue CON Docker (recomendada).....	29
4.8.2 Configuración y despliegue SIN Docker.....	30
5 – El cliente.....	31
5.1 Descripción general y tecnologías usadas en el cliente.....	31
5.2 Arquitectura interna del cliente.....	31
5.2.1 Estructura de los paquetes del modelo.....	33
5.2.2 Activities de la app.....	33
5.2.3 Permisos de la app.....	33
5.3 Modelo y base de datos del cliente.....	34
5.3.1 Diseño conceptual de BD en el cliente.....	34
5.3.2 Diseño lógico de BD en el cliente.....	34
5.4 Algoritmo de sincronización.....	35
5.5 Almacenamiento en el cliente.....	36
5.6 Casos de uso del cliente.....	36
5.7 GUI, navegación y elementos gráficos.....	38
5.7.1 Mockup de la interfaz gráfica.....	38
5.7.2 Mapa de navegación.....	39
5.7.3 Recursos gráficos.....	39
5.8 Pruebas realizadas.....	40
5.9 Requisitos de la aplicación.....	40
5.10 Manual de uso de la aplicación móvil.....	41
5.10.1 Descripción.....	41
5.10.2 Inicio de la aplicación y creación del usuario.....	41
5.10.3 Listado de grupos.....	42

5.10.4 Ver detalle de grupo, agregar canciones y otros usuarios al grupo.....	43
5.10.5 Ver detalle de canción y agregar notas.....	44
5.10.6 Crear y visualizar notas.....	44
5.10.7 Sincronizar los datos con la nube y otros usuarios.....	46
6. Bibliografía.....	48

1 – Descripción general y objetivo

Durante la actividad de los grupos musicales es común hacer grabaciones de las obras tanto completas como de partes específicas generándose multitud de anotaciones en texto y audio para registrar el estado en que se encuentra cada canción. Se suele hacer utilizando los dispositivos móviles de los componentes del grupo durante las sesiones de ensayo. Esas anotaciones y audios se distribuyen entre los miembros del grupo usando aplicaciones de mensajería como WhatsApp o Telegram resultando en una falta de control de las diferentes anotaciones que se traduce en la pérdida de las anotaciones y la dificultad para hallar un audio o nota concreta conforme se crean versiones de las canciones. Además es bastante habitual que los músicos formen parte de más de un grupo musical teniendo que gestionar la distribución y acceso de la información en diferentes lugares dependiendo del grupo musical del que estén buscando los audios y textos.

Otra dificultad añadida es el hecho de la ausencia de conexión a internet en algunos lugares de ensayo teniendo que posponerse la distribución de la información y surgiendo la necesidad de llevar personalmente el control sobre las notas y archivos de audio desde que se toman hasta el posterior envío al resto de miembros.

La aplicación desarrollada durante este proyecto tiene como objetivo solventar este problema. Es un asunto que conozco de primera mano lo cual ha hecho posible analizar los requerimientos junto con otros usuarios que han podido aportarme una visión real y diversa sobre las funcionalidades que se necesitan en casos así.

Afrontar este problema me ha permitido consolidar y extender lo ya aprendido en varios de los módulos del grado de Desarrollo de Aplicaciones Multiplataforma, especialmente en los módulos de Programación, Bases de Datos, Entornos de Desarrollo, Acceso a Datos, Procesos y Servicios y Programación Multimedia y dispositivos Móviles. También me ha permitido iniciarme en algunos aspectos importantes de cara al entorno laboral con el uso de algunos frameworks como Spring, Retrofit y Room, una visión más profunda del patrón MVVM y la necesidad de desarrollar un algoritmo y diseño de la estructura de los datos con los que sincronizar de manera diferida la información entre usuarios.

2 – El proceso de desarrollo del proyecto

El planteamiento del proyecto se ha hecho teniendo en cuenta la limitación de tiempo disponible buscando un balance entre máxima funcionalidad de la aplicación y cumplimiento de la fecha de entrega.

En un primer momento se hizo un análisis preliminar de los requisitos de la aplicación que permitió identificar y planificar las partes principales del desarrollo de la aplicación. Se identificaron cinco puntos principales: Análisis detallado de requerimientos, diseño del modelo de datos, diseño del sistema de sincronización, implementación del servidor e implementación del cliente móvil.

La metodología usada ha sido una mezcla entre el tipo en cascada con retroalimentación y la metodología AGILE. Los análisis de requerimientos, modelo de datos y diseño de sincronización se hizo al inicio como fases independientes y secuenciales pero con cierta retroalimentación. La implementación del servidor y la implementación del cliente móvil se hizo siguiendo metodología AGILE surgiendo puntualmente retroalimentación hacia otras fases del desarrollo.

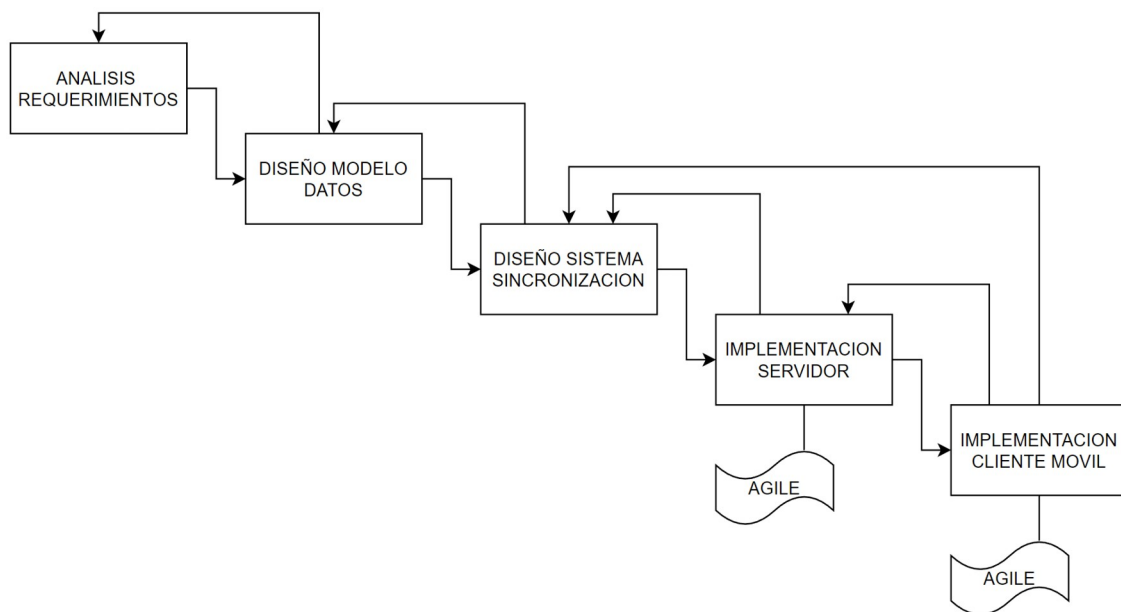


Figura 1: Esquema del proceso de desarrollo

Durante el avance del proyecto recibí las impresiones de diferentes usuarios que fueron aportando ideas que supusieron la modificaron los requerimientos quitando algunos y agregando otros.

La implementación del código tanto del servidor como cliente móvil se hicieron siguiendo la técnica de TDD. En el caso del servidor usando Postman para la implementación de las pruebas y en el caso del cliente móvil haciendo pruebas manuales en emuladores y dispositivos reales.

El desarrollo del proyecto se ha hecho con un ordenador con sistema Windows, DrawIo para diagramas, los IDE Netbeans y Android Studio, dispositivos móviles emulados y reales principalmente con Android 11 y 14, Postman para pruebas del servidor, y GIT para el control de versiones. Finalmente el despliegue del servidor se ha hecho usando Docker Composer.

3 – Visión global de la solución

3.1 Requisitos funcionales

El análisis de requerimientos final determina los siguientes puntos:

- La aplicación permite almacenar grupos, canciones, notas y audios.
- Los grupos se comparten entre uno o más usuarios y solo ellos pueden acceder a la información del grupo.
- Los propios miembros del grupo pueden modificar qué otros usuarios pueden acceder al grupo.
- La información se introduce y modifica desde dispositivos móviles.
- Se puede hacer la visualización, introducción y modificación de los datos sin acceso a internet.
- La aplicación provee un sistema propio para grabado de audio.
- La aplicación puede incorporar archivos de audio existentes fuera de la aplicación.
- La aplicación debe poder aprovechar otros sistemas de grabación que estuviesen presentes en el dispositivo móvil.
- La aplicación provee un sistema para reproducir los audios.
- La aplicación puede compartir las notas y audio fácilmente hacia otras aplicaciones.
- Los datos introducidos y modificados en un teléfono deben sincronizarse con los del resto de usuarios del grupo musical y debe hacerse cuando el usuario así lo quiera.
- Para la sincronización se debe tener acceso a internet.
- Si durante la sincronización surgen conflictos de ediciones hechas de manera simultánea se le presenta al usuario la manera de resolver el conflicto sin pérdida de datos.

3.2 Arquitectura global y flujo de información

Dados estos requisitos el diseño global de la aplicación se ha dividido en 2 partes principalmente. Un servidor y clientes. Los clientes sincronizan la información entre sí a través de ese servidor.

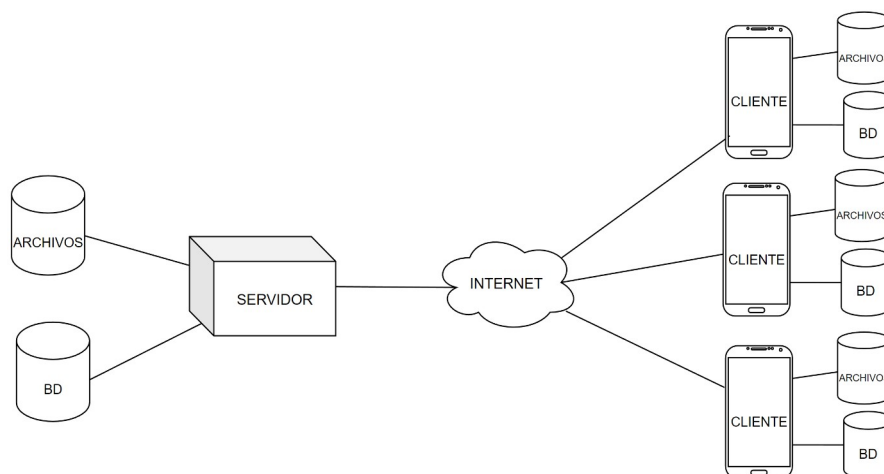


Figura 2: Esquema general de la aplicación

El flujo de información en este esquema sucedería de la manera siguiente:

1. El usuario A introduce datos en su teléfono móvil.
2. El usuario A pide la sincronización.
3. Los datos entre el teléfono de usuario A y lo que hay en el servidor se sincronizan.
4. El usuario B pide a su teléfono móvil sincronizar.
5. Se sincronizan los datos entre el teléfono del usuario B y el servidor.

Tras eso el usuario B tiene en su teléfono todo los datos, incluidas las modificaciones realizadas por el usuario A, y en el servidor estarían sincronizado también con los datos que hubiese introducido el usuario B.

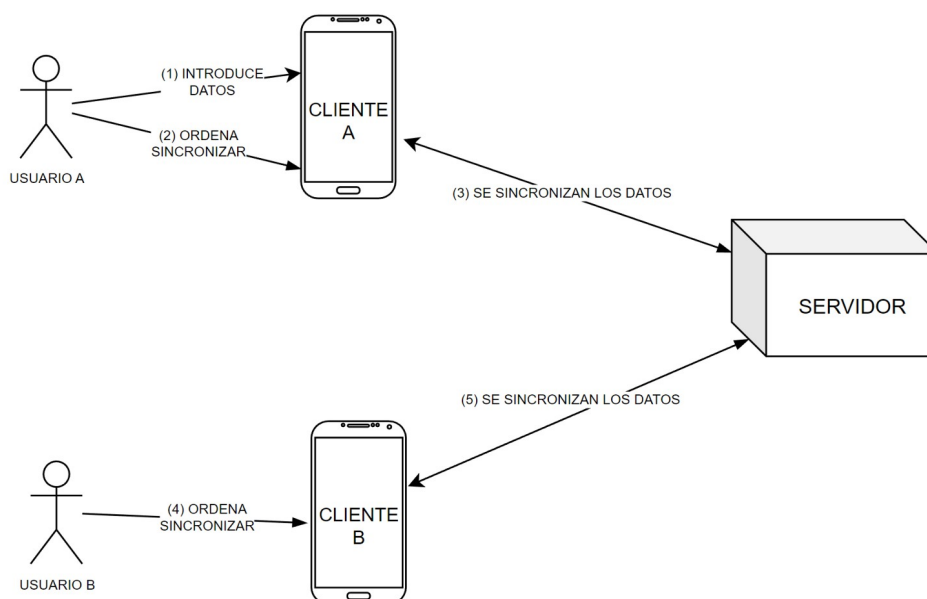


Figura 3: Flujo de información en el uso de la aplicación

4 – El servidor

4.1 Descripción general y tecnologías usadas en el servidor

El servidor es un servicio web encargado de almacenar y suministrar información y archivos. Está implementado en **Java 19** con **Spring** mediante **Spring Boot3** en un proyecto **Maven**.

Almacena los archivos de audio usando el **sistema de archivos** y para el resto de datos usa una base de datos gestionada por un servidor **MySQL** a la que accede mediante **Spring Data** con **Hibernate**.

La guarda de acceso a los endpoints del servicio web es gestionado por el módulo **Spring Security** mediante tokens de acceso.

La comunicación entre el servidor y los clientes se establece mediante llamadas/respuestas **HTTP** cuyo cuerpo son objetos **JSON** con los datos a transferir. Cuando es necesario un token de acceso para una de las peticiones, este se establece en la petición en un header *Authorization* que contiene un **Bearer token**. La conversión entre las entidades de Java y los objetos JSON se hace mediante la librería **Jackson**.

La implementación hace uso extensivo de la librería **Lombok** que reduce el código a implementar manualmente y aprovecha el sistema de inyección de dependencias suministrada por el propio Spring.

Para el despliegue se han preparado dos scripts de **Docker** y **Docker Compose** que por un lado crean una imagen Docker con el servidor en sí y por otro lado el sistema completo compuesto por dos imágenes Docker, una con el programa en Java y otro con el servidor de MySQL.

4.2 Arquitectura interna del servidor

El **servicio web** del servidor tiene endpoints a los que los usuarios hacen peticiones enviando información en formato JSON. Tras recibirse la llamada a un endpoint el servidor verifica la pertenencia al usuario que hace la petición de las entidades que serán afectadas. Las respuestas devuelven la información a través del cuerpo de la respuesta en formato JSON y por el propio código de estatus de la respuesta. El nivel de detalle del JSON de las respuestas es controlado con anotaciones para evitar el filtrado de datos no deseados como por ejemplo las claves de usuario. Esas anotaciones son también usadas para modificar si las entidades hijas son o no mostradas en las respuestas de las peticiones.

Los endpoints del servicio web están divididos en dos grupos según el acceso. El acceso está controlado por **Spring Security**:

- Acceso anónimo: Login y Registro
- Acceso restringido mediante un token JWT de acceso para el resto de endpoints.

Cuando llega una petición lo primero que sucede es la comprobación de Spring Security. Si Spring Security valida correctamente la autorización de la petición, entonces la pasa a los controladores que es donde se definen los endpoints.

Cuando la petición pasa al **controlador** se produce una validación inicial en la entrada a los controladores comprobando si los datos recibidos en el cuerpo de las peticiones están bien conformados.

Después los controladores hacen llamadas a **servicios** donde está la lógica de negocio y estos a su vez ejecutan sus acciones de persistencia usando **repositorios** CRUD para resolver el acceso a los datos. Finalmente los repositorios usan el propio sistema de archivos para los audios y un servidor MySql para obtener y modificar la información de la base de datos.

Además del servicio web en sí, el servidor también cuenta con una **tarea programada** que se encarga de ir comprobando cada cierto tiempo la limpieza del almacén de archivos de audio eliminando los archivos no vinculados a alguna entidad audio de la base de datos. Pese a que cuando una entidad audio es borrada o modificada directamente los archivos de audio vinculados son eliminados, esto no sucede cuando los audios se borran en cascada por haberse eliminado entidades ascendentes. Se diseñó así con vista a minimizar el impacto en el sistema de borrados evitando tener que comprobar para cada borrado el sistema de archivos y la base de datos de forma extensa.

La tarea programada de limpieza asegura que de manera periódica se eliminan los archivos de audio que ya no están en uso.

El servidor cuenta también con la escritura de **log de actividad** en archivo además de mostrarlo por consola quedando un registro de las actividades de cada usuario en el servidor.

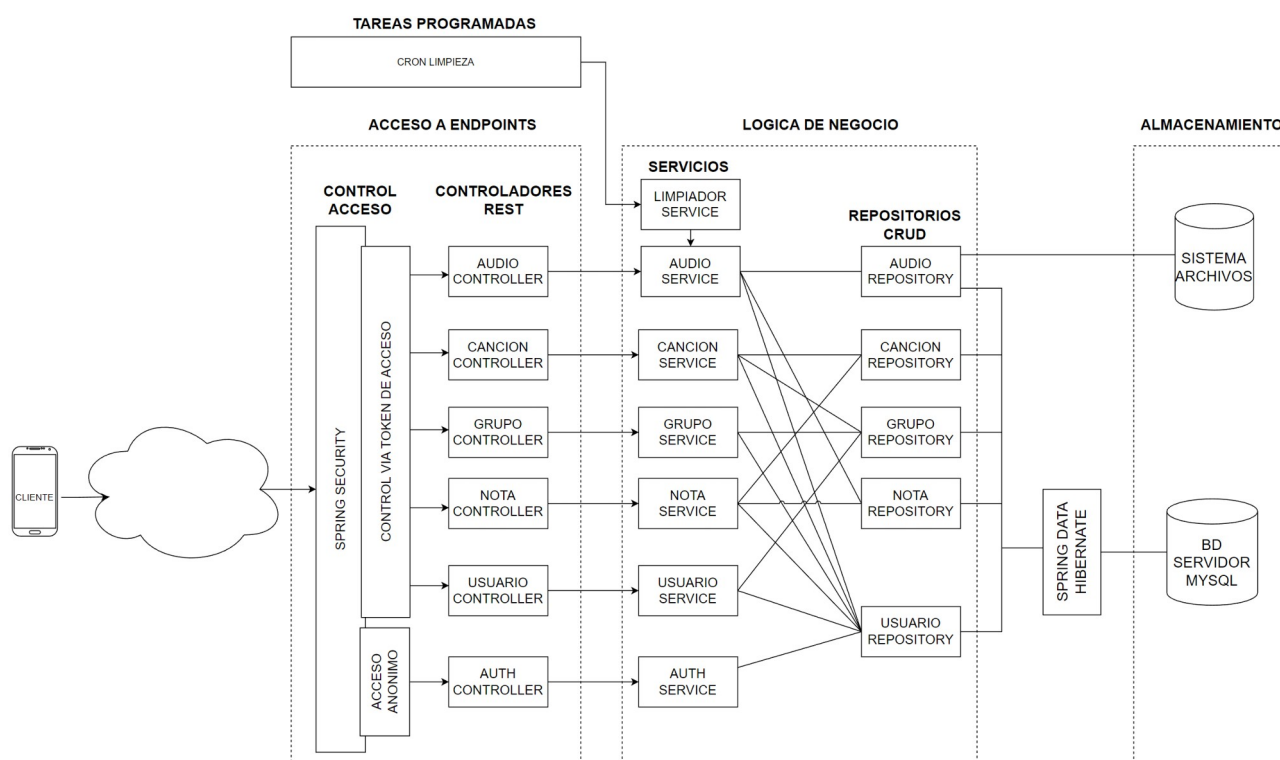


Figura 4: Estructura interna del servidor

4.2.1 Ejemplos del código

Ejemplo de controlador y uno de los endpoints que se definen dentro de él. En este caso el GrupoController encargado de definir los endpoints de acceso a los grupos. En el ejemplo el endpoint de tipo POST que sirve para crear un nuevo grupo en el servidor:

```
@RestController
@RequestMapping("/grupo")
@RequiredArgsConstructor
public class GrupoController {
    private final IGrupoService grupoService;
    @JsonView(Vista.Esencial.class)
    @PostMapping()
    public Grupo create(@RequestBody @Valid Grupo request, @AuthenticationPrincipal UserPrincipal principal){
        return grupoService.create(request, principal.getUserId());
    }
    [...]
}
```

Los controladores se comunican con los servicios según una interfaz. En el ejemplo anterior se puede ver que el atributo **grupoService** es de tipo **IGrupoService** que es en realidad una interfaz. La implementación de esa interfaz se encuentra en **GrupoServiceImpl**.

En el siguiente ejemplo de servicio, se puede ver la creación de un nuevo grupo y la asignación del usuario que solicita la creación de ese grupo como miembro con acceso a dicho grupo. Lo hace usando los repositorios de grupos y usuarios (El uso de los repositorios por parte de los servicios se puede ver en [Figura 4: Estructura interna del servidor](#)):

```
public class GrupoServiceImpl implements IGrupoService {

    private final ResolutorPermisos resolutorPermisos;
    private final UsuarioRepository repositorioUsuario;
    private final GrupoRepository repositorioGrupo;
    private Logger logger = Logger.getLogger(GrupoServiceImpl.class.getName());

    @Override
    @Transactional
    public Grupo create(Grupo grupo, Long idUsuario) {
        Optional<Usuario> usuario = repositorioUsuario.findById(idUsuario);
        Optional<Grupo> grupoLocal = repositorioGrupo.findById(grupo.getId());
        if (grupoLocal.isPresent()) {
            throw new ResponseStatusException(HttpStatus.CONFLICT);
        }
        if (usuario.isPresent()) {
            Usuario u = usuario.get();
```

```

        grupo.setVersion(grupo.getVersion()+1);
        grupo.getUsuarios().add(u);
        u.getGrupos().add(grupo);
        repositorioGrupo.save(grupo);
        repositorioUsuario.save(u);
        logger.log(Level.INFO, "Grupo creado: {0}", grupo.getId().toString() + " usuario " + idUsuario);
        return grupo;
    } else {
        return null;
    }
}
[...]
```

Spring suministra repositorios CRUD basándose en las entidades que se hayan definido y es necesario únicamente especificar qué entidad es la que maneja el repositorio y el tipo de dato que es la clave primaria.

El siguiente ejemplo es del repositorio de grupos y se encuentra en ***GrupoRepository***.

```

public interface GrupoRepository extends CrudRepository<Grupo, UUID>{
}
```

La entidad correspondiente a este repositorio es la entidad ***model.entity.Grupo***:

```

@Entity
@Getter
@Setter
public class Grupo {

    @Id
    @JsonView(Vista.Esencial.class)
    @NotNull
    private UUID id;

    @JsonView(Vista.Esencial.class)
    @NotEmpty
    private String nombre;

    @JsonView(Vista.Esencial.class)
    @NotNull
    private String descripcion;

    @JsonView(Vista.Esencial.class)
    @NotNull
    private int version;

    @JsonView(Vista.Esencial.class)
    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")
    @NotNull
```

```

private Date fecha;

@ManyToMany
@JoinTable(
    name = "usuario_grupo",
    joinColumns = @JoinColumn(name="grupo_id"),
    inverseJoinColumns = @JoinColumn(name = "usuario_id")
)
@JsonView(Vista.Esencial.class)
private Set<Usuario> usuarios = new HashSet<>();

@OneToMany(mappedBy="grupo", cascade = CascadeType.ALL)
@JsonView(Vista.Completa.class)
private Set<Cancion> canciones;

@Override
public String toString() {
    return "Grupo{" + "id=" + id + ", nombre=" + nombre + ", descripcion=" + descripcion + ", version="
+ version + '}';
}
}

```

4.2.2 Diagramas de clases

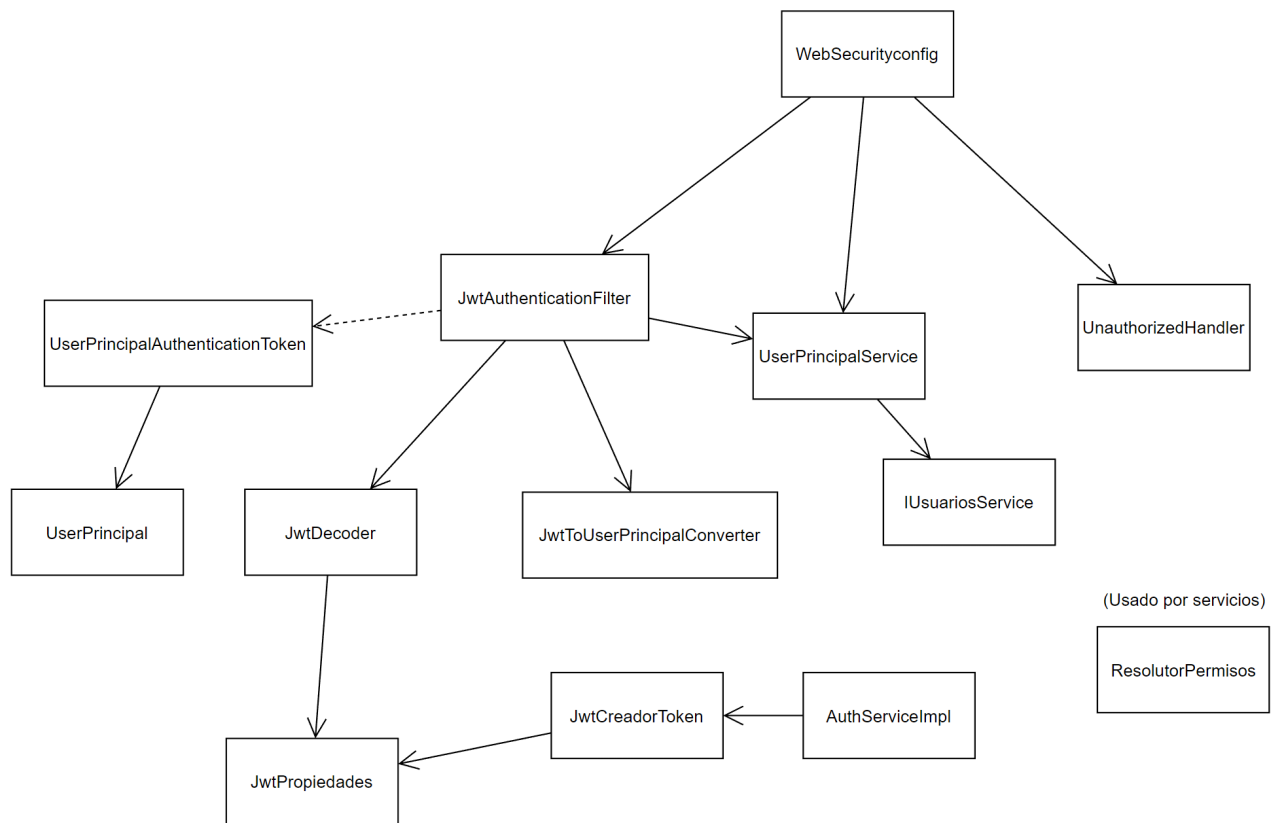


Figura 5: Relación entre las clases del paquete security

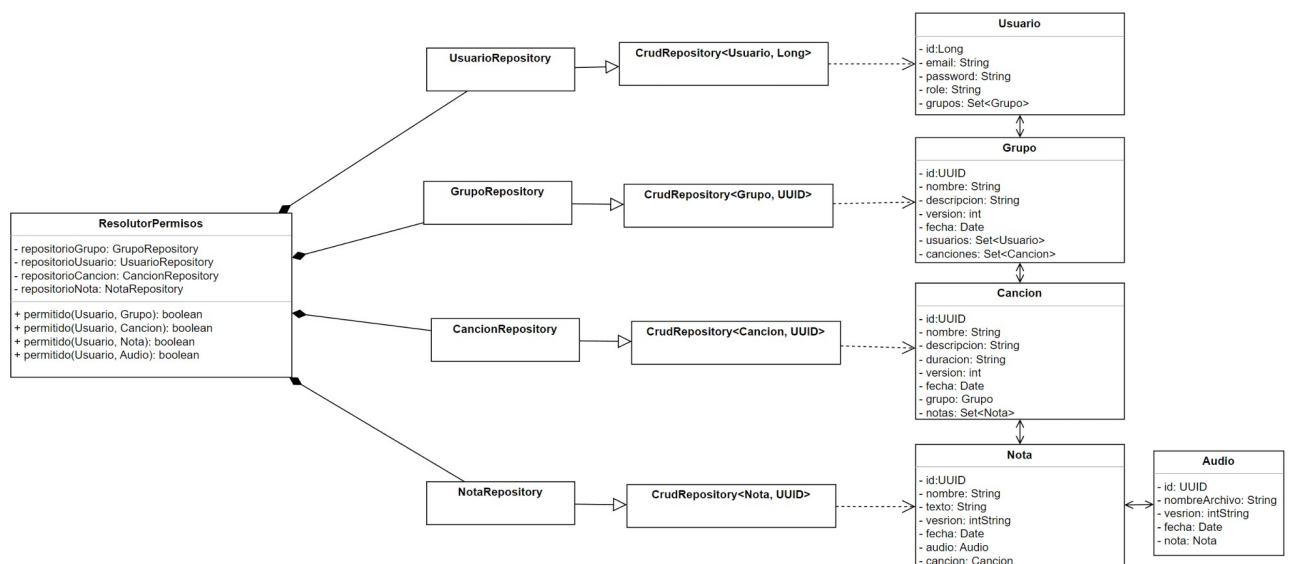


Figura 6: Clases relacionadas con la clase ResolutorPermisos. La clase ResolutorPermisos es usada por los servicios para determinar si se tiene permiso de acceso a una entidad concreta.

En este esquema se puede apreciar también las clases de entidades de la base de datos y su relación con los repositorios y entre sí.

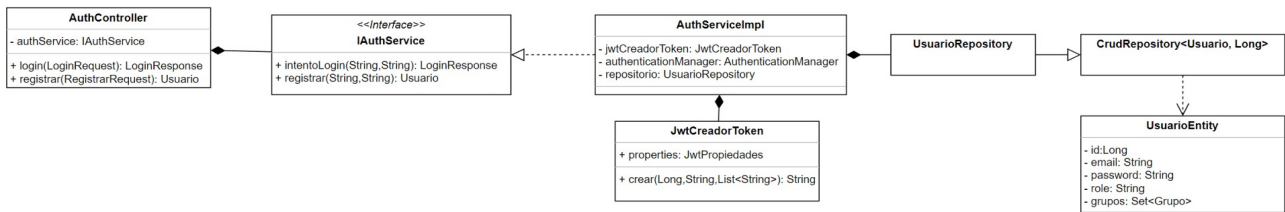


Figura 7: Clases relacionadas con Authcontroller

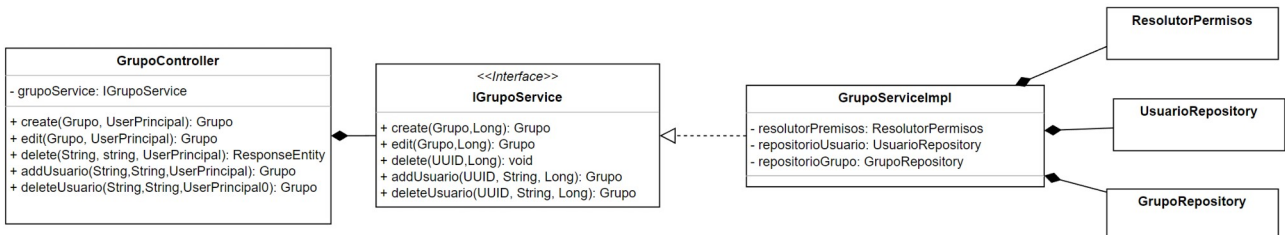


Figura 8: Clases relacionadas con GrupoController

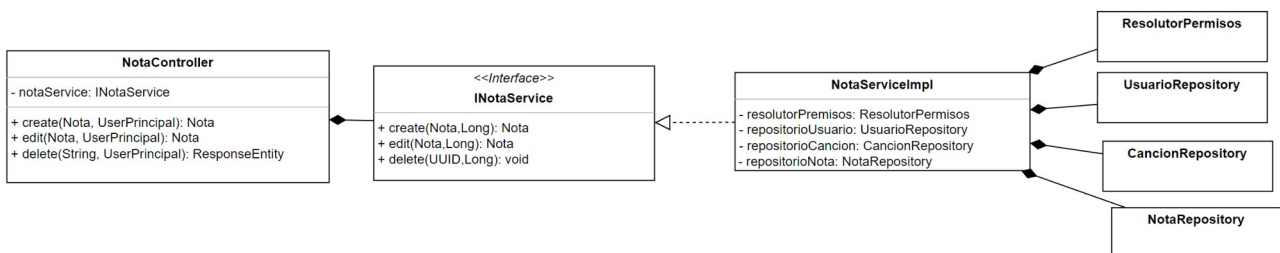


Figura 9: Clases relacionadas con NotaController

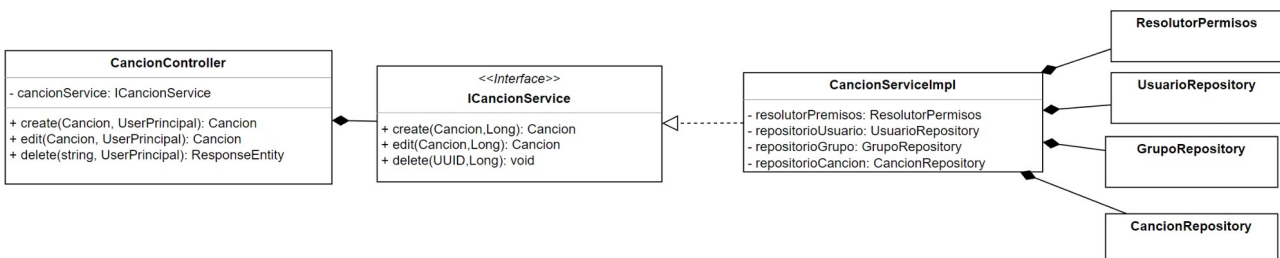


Figura 10: Clases relacionadas con CancionController

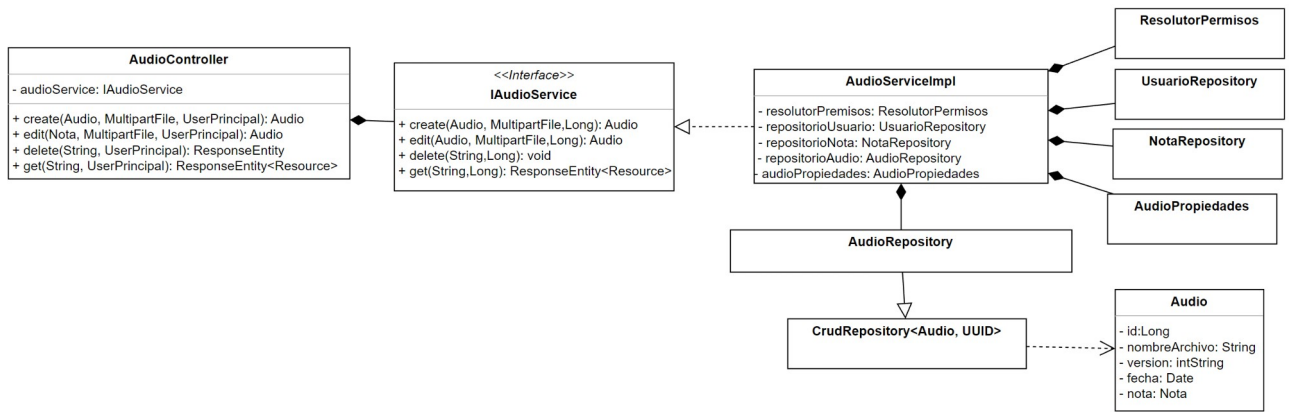


Figura 11: Clases relacionadas con AudioController

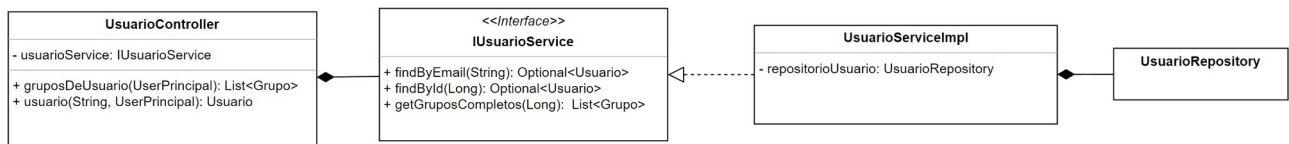


Figura 12: Clases relacionadas con UsuarioController

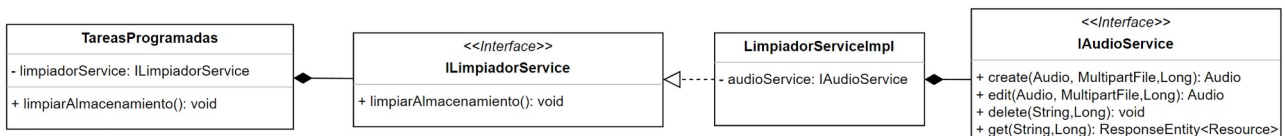


Figura 13: Clases relacionadas con la tarea de limpieza de almacenamiento

4.3 Modelo y base de datos del servidor

Para el diseño de la base de datos, además de los requerimientos de las entidades en sí, ha influido el hecho de que se trata de una aplicación distribuida donde las entidades no se crean de manera centralizada. Esa creación y modificación distribuida y la posterior sincronización ha determinado que las claves primarias de las entidades se hagan con "UUID" en vez de con enteros y se han agregado campos de "version" para el control de algunos aspectos de la sincronización:

- El tipo de dato UUID elegido para las claves primarias permite la unicidad práctica de las claves primarias sin que la creación sea centralizada lo cual hace posible la creación de entidades en modo offline. Esto es un requisito fundamental de la aplicación. Con UUID se consigue sin caer en un costoso sistema de referencias en el momento de las sincronizaciones. En su momento se valoraron ambas opciones, UUID e Int, este último con proceso de referencias duales para cliente y servidor, pero tras un estudio se determinó que UUID era mejor opción. Mientras un Int se guarda en 4 bytes un UUID se guarda en 16 bytes. La diferencia de tamaño no se espera que sea problemática para el tamaño y desempeño de la base de datos y elimina la tendencia a error y el coste computacional que se podría generar si se tuviesen tablas de equivalencia de identificadores que emparejaran las entidades locales con las del servidor.
- El campo de versión que tienen algunas entidades es necesario para el algoritmo de sincronización como se verá en secciones posteriores.

Tanto el tipo de dato UUID como el campo de versión es una característica que se podrá encontrar también en la versión de la base de datos de los clientes móviles.

Las entidades están especificadas en el paquete **model.entity**.

Spring data se encarga directamente de la construcción del esquema de la base de datos en el servidor.

Seguidamente se detalla la estructura de la base de datos del servidor.

4.3.1 Diseño conceptual de BD en el servidor

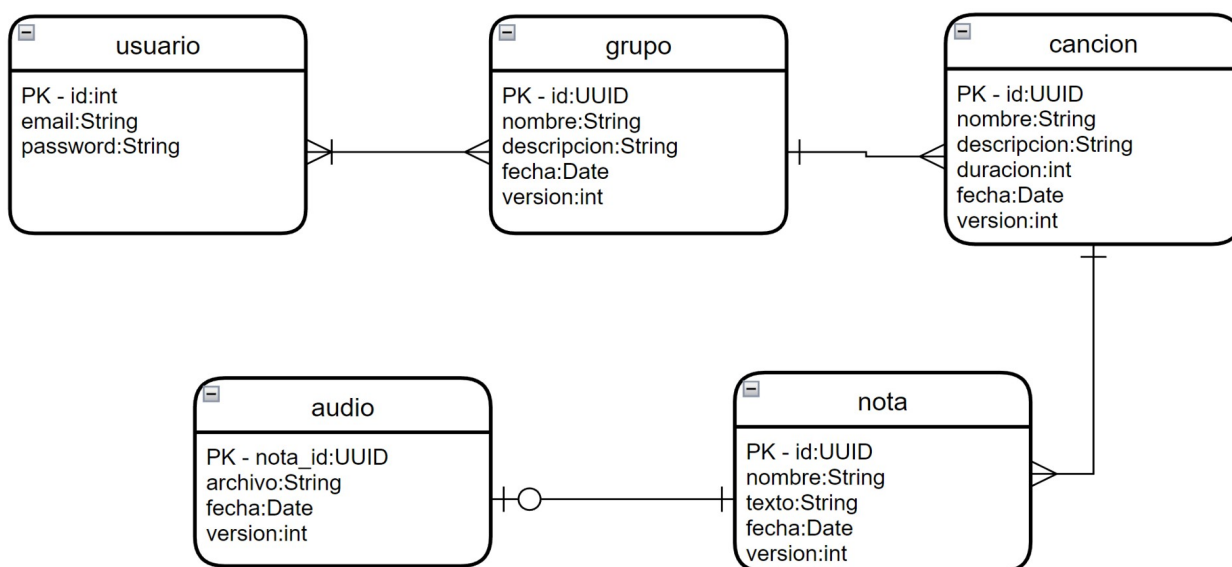


Figura 14: Diseño conceptual BD en servidor

4.3.2 Diseño lógico de BD en el servidor

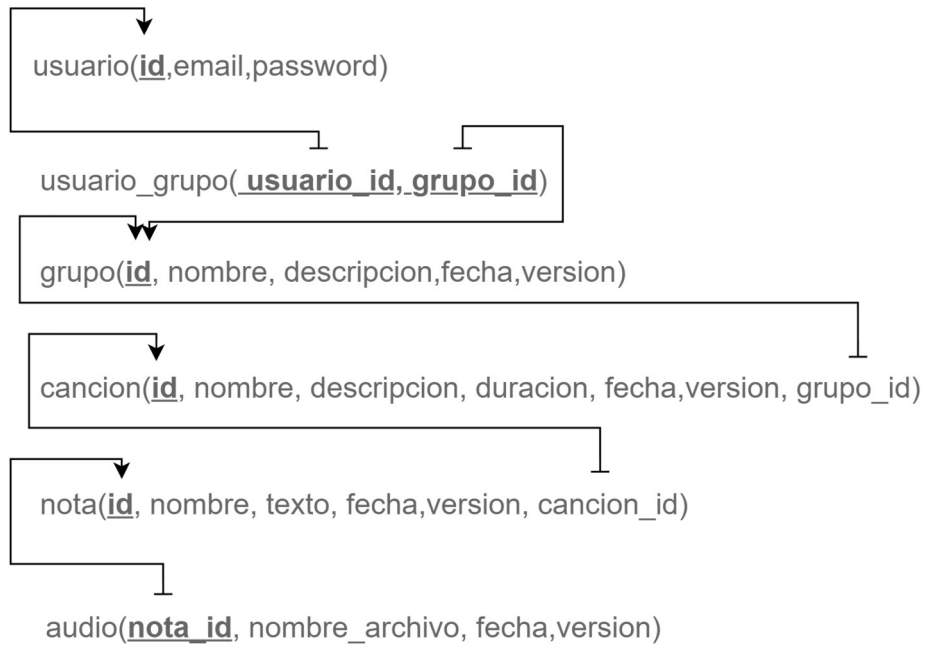


Figura 15: Diseño lógico de BD en servidor

El diseño final de la base de datos incluye eliminación y actualización en cascada para todas las claves foráneas, y ningún trigger.

4.4 Casos de uso del servidor

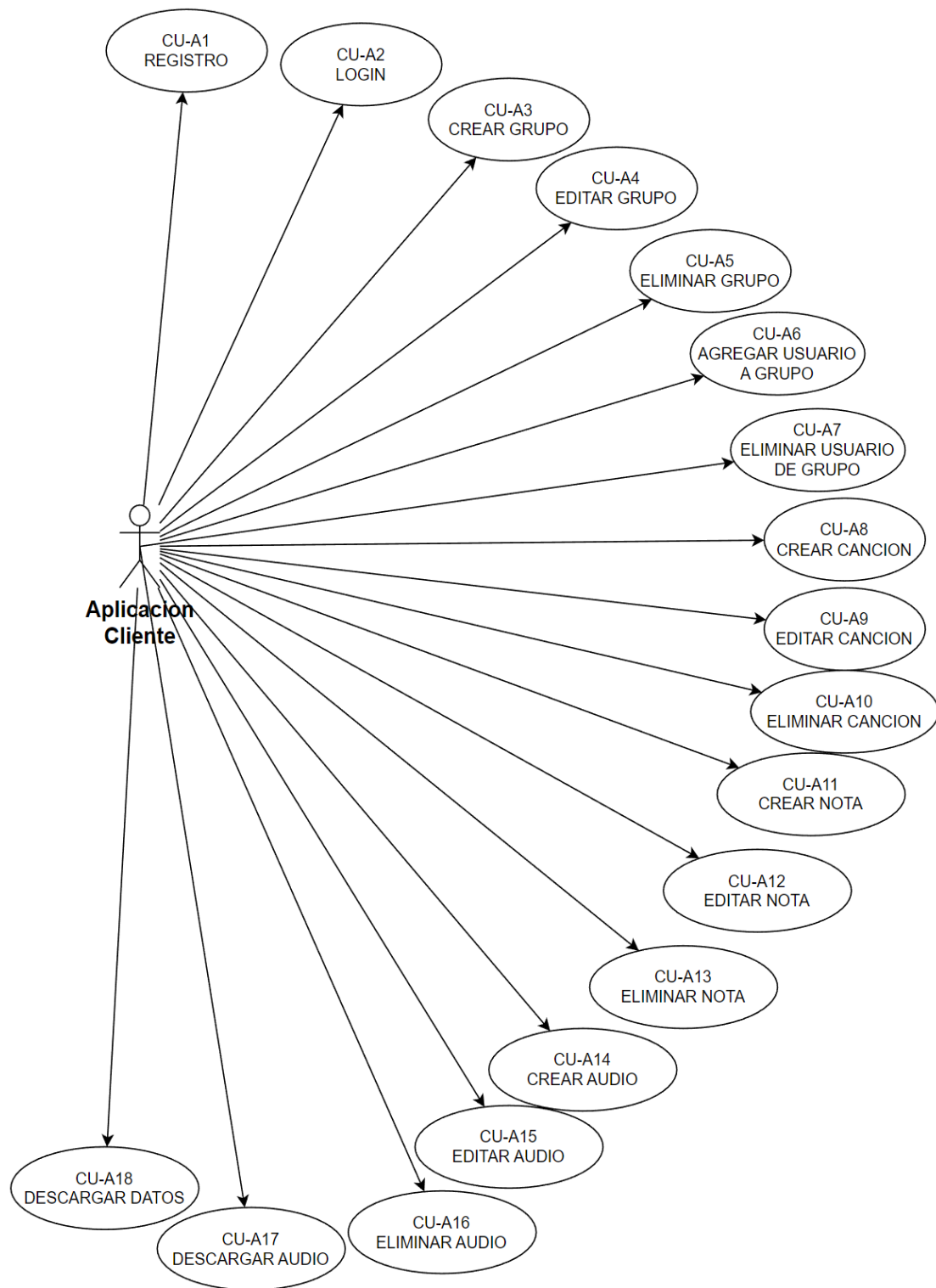


Figura 16: Diagrama de casos de uso del servidor

Características principales de los casos de uso:

- **CU-A1 REGISTRO:** Registro de un nuevo usuario en el sistema. Debe responder con un OK si va bien o avisar de que los datos no son válidos por existir ya un usuario como el que se pretende registrar.
- **CU-A2 LOGIN:** Comprobar los datos de las credenciales aportadas. Devolver un token de acceso si son correctos o avisar de credenciales erróneas en otro caso.
- **CU-A3 a CU-A16:** Inserción, modificación y borrado de diferentes entidades:
 - **Si es creación:** Responder con un OK si se tienen permisos para acceder a la entidad padre y efectuar la inserción. Avisar de la falta de autorización en otro caso.
 - **Si es modificación:** Responder con un OK si se tienen permisos para modificar entidades padre y efectuar la modificación. Avisar de la falta de autorización si no se tiene autorización. Comprobar si la versión que se pretende modificar corresponde con la de la petición. Devolver el estado actual de la entidad si no coincide la versión y avisar de que no se ha producido la modificación.
 - **Si es eliminación:** Responder con un OK si se tienen permisos para eliminar la entidad. Eliminarla y eliminar las entidades hijas. Si no se tienen permisos para eliminar la entidad avisar de la falta de autorización.
- **CU-A17 DESCARGAR AUDIO:** Comprobar si se tienen permisos. Si se tienen responder con el archivo asociado al audio. Si no se tienen permisos avisar de ese error.
- **CU-A18 DESCARGAR DATOS:** Devolver un listado de grupos a los que pertenece el usuario. Incluir también las entidades hijas relacionadas con cada grupo.

4.5 Almacenamiento en el servidor

El servidor almacena datos en tres sitios.

Por un lado los datos de las entidades los almacena en una base de datos llamada *db_ensayos*. Esta base de datos estará en el servidor MySQL definido en la configuración.

En el sistema de archivos almacena tanto los archivos de audio como los archivos log que se vayan generando. Ambas localizaciones se pueden definir en la configuración. Los logs en "{almacenamiento}/log/servidor_ensayo.log" y los audios en "{almacenamiento}/audio".

4.6 Endpoints del servicio web

Los endpoints pueden tener un JSON como cuerpo tanto en peticiones como respuestas.

Las respuestas siguen el siguiente patrón:

- **200 OK:** *La operación se ha realizado con éxito.*
- **400 Bad Request:** Si un endpoint requiere un cuerpo en la petición y NO recibe ese cuerpo.
- **401 Forbidden:** Si un endpoint que SÍ requiere un header de autorización NO lo tiene o si el token es erróneo.
- **404 Not Found:** Para todos los endpoints, si no existe una entidad con esa id o no se tienen permisos de acceso a esa entidad (solo los usuarios de un grupo pueden acceder a entidades

de ese grupo)

- **CREACION 409 Conflict:** Para los endpoints de creación, si ya existe una entidad con el mismo ID. (*UUID debe garantizar de facto que esto no suceda*)
- **EDICION 409 Conflict:** Para los endpoints de edición, si el parámetro de versión de la entidad no coincide. El cuerpo de la respuesta contendrá una representación JSON con la entidad actual. **Comportamiento necesario para el algoritmo de sincronización.**

Los endpoints son los siguientes:

Endpoints de autorización

RUTA	/auth/login	
USO	Obtener acceso	
MÉTODO	POST	
AUTHORIZATION	Ninguna	
CUERPO	{ "email": "{email}", "password": "{contraseña}" }	
RESPUESTAS		
CODIGO	SIGNIFICADO	CUERPO
200	Acceso correcto. El token del cuerpos se puede usar para acceso a endpoints restringidos por Bearer token	{ "accessToken":"{token}" }
401	Acceso incorrecto	

RUTA	/auth/registrar	
USO	Registrar un usuario	
MÉTODO	POST	
AUTHORIZATION	Ninguna	
CUERPO	{ "email": "{email}" "password": "{contraseña}" }	
RESPUESTAS		
CODIGO	SIGNIFICADO	CUERPO
200	Registro correcto correcto	{ "id": {n}, "email": "{email}", "role": "usuario" }
409	Email ya usado	

Endpoints de grupos

RUTA		/grupo
USO		Crear un grupo
MÉTODO		POST
AUTHORIZATION		Bearer token
CUERPO		JSON de entidad Grupo
RESPUESTAS		
CODIGO	SIGNIFICADO	CUERPO
200	Creacion correcta	JSON de entidad Grupo creada

RUTA	/grupo	
USO	Editar un grupo	
MÉTODO	PUT	
AUTHORIZATION	Bearer token	
CUERPO	JSON de entidad Grupo	
RESPUESTAS		
CODIGO	SIGNIFICADO	CUERPO
200	Edicion correcta	JSON de entidad Grupo editada
409	Version de entidad errónea	JSON de entidad Grupo actual en servidor

RUTA		/grupo/{idgrupo}
USO		Eliminar un grupo
MÉTODO		DELETE
AUTHORIZATION		Bearer token
CUERPO		Nada. Variables en ruta.
RESPUESTAS		
CODIGO	SIGNIFICADO	CUERPO
200	Eliminación correcta	Nada

RUTA		/grupo/{idgrupo}/{emailusuario}
USO		Agregar un usuario a un grupo
MÉTODO		POST
AUTHORIZATION		Bearer token
CUERPO		Nada. Variables en ruta.
RESPUESTAS		
CODIGO	SIGNIFICADO	CUERPO
200	Usuario agregado al grupo	JSON de entidad Grupo tras el cambio.

RUTA	/grupo/{idgrupo}/{emailusuario}	
USO	Eliminar un usuario d un grupo	
MÉTODO	DELETE	
AUTHORIZATION	Bearer token	
CUERPO	Nada. Variables en ruta.	
RESPUESTAS		
CODIGO	SIGNIFICADO	CUERPO
200	Usuario eliminado del grupo	JSON de entidad Grupo tras el cambio.

Endpoints de canciones

RUTA	/cancion/{idgrupo}	
USO	Crear una cancion dentro de un grupo	
MÉTODO	POST	
AUTHORIZATION	Bearer token	
CUERPO	JSON de entidad Cancion. (id del grupo al que pertenece la cancion en la ruta)	
RESPUESTAS		
CODIGO	SIGNIFICADO	CUERPO
200	Creación correcta	JSON de entidad Cancion creada

RUTA	/cancion	
USO	Editar cancion	
MÉTODO	PUT	
AUTHORIZATION	Bearer token	
CUERPO	JSON de entidad Cancion	
RESPUESTAS		
CODIGO	SIGNIFICADO	CUERPO
200	Edicion correcta	JSON de entidad Cancion editada
409	Version de entidad errónea	JSON de entidad Cancion actual en servidor

RUTA	/cancion/{idcancion}	
USO	Eliminar una cancion	
MÉTODO	DELETE	
AUTHORIZATION	Bearer token	
CUERPO	Nada. Variables en ruta.	
RESPUESTAS		
CODIGO	SIGNIFICADO	CUERPO
200	Eliminación correcta	Nada

Endpoints de notas

RUTA	/nota/{idcancion}	
USO	Crear una nota dentro de una cancion	
MÉTODO	POST	
AUTHORIZATION	Bearer token	
CUERPO	JSON de entidad Nota. (id de la cancion al que pertenece la nota en la ruta)	
RESPUESTAS		
CODIGO	SIGNIFICADO	CUERPO
200	Creación correcta	JSON de entidad Nota creada

RUTA	/nota	
USO	Editar nota	
MÉTODO	PUT	
AUTHORIZATION	Bearer token	
CUERPO	JSON de entidad Nota	
RESPUESTAS		
CODIGO	SIGNIFICADO	CUERPO
200	Edicion correcta	JSON de entidad Nota editada
409	Version de entidad errónea	JSON de entidad Nota actual en servidor

RUTA	/nota/{idnota}	
USO	Eliminar una nota	
MÉTODO	DELETE	
AUTHORIZATION	Bearer token	
CUERPO	Nada. Variables en ruta.	
RESPUESTAS		
CODIGO	SIGNIFICADO	CUERPO
200	Eliminación correcta	Nada

Endpoints de audios

RUTA	/audio/{idaudio}	
USO	Descargar archivo de audio	
MÉTODO	GET	
AUTHORIZATION	Bearer token	
CUERPO	Nada. Variables en ruta.	
RESPUESTAS		
CODIGO	SIGNIFICADO	CUERPO
200	Descarga correcta	Respuesta con header "Content-Disposition" con el nombre del archivo y cuerpo "application/octet-stream" con el archivo a descargar.
404	Además de los casos generales, sucede también si no se encuentra el archivo.	Nada

RUTA	/audio	
USO	Crear un audio	
MÉTODO	POST	
AUTHORIZATION	Bearer token	
CUERPO	Multipart: <ul style="list-style-type: none">"archivo": MultipartFile con el archivo de audio"datos": JSON con la entidad audio para la base de datos	
RESPUESTAS		
CODIGO	SIGNIFICADO	CUERPO
200	Creación correcta	JSON de entidad Audio creada
415	Tipo de archivo incorrecto (solo se permite audio/mpeg)	nada
500	Si se produce un error guardando el archivo.	Nada

RUTA	/audio	
USO	Editar un audio	
MÉTODO	PUT	
AUTHORIZATION	Bearer token	
CUERPO	Multipart: <ul style="list-style-type: none">• "archivo": MultipartFile con el archivo de audio• "datos": JSON con la entidad audio para la base de datos	
RESPUESTAS		
CODIGO	SIGNIFICADO	CUERPO
200	Edicion correcta	JSON de entidad Audio editada
409	Version de entidad errónea	JSON de entidad Audio actual en servidor
415	Tipo de archivo incorrecto (solo se permite audio/mpeg)	nada
500	Si se produce un error guardando el archivo.	Nada

RUTA	/nota/{idaudio}	
USO	Eliminar un audio	
MÉTODO	DELETE	
AUTHORIZATION	Bearer token	
CUERPO	Nada. Variables en ruta.	
RESPUESTAS		
CODIGO	SIGNIFICADO	CUERPO
200	Eliminación correcta	Nada
500	Si se produce un error eliminando el archivo.	Nada

RUTA	/nota/{idaudio}	
USO	Eliminar un audio	
MÉTODO	DELETE	
AUTHORIZATION	Bearer token	
CUERPO	Nada. Variables en ruta.	
RESPUESTAS		
CODIGO	SIGNIFICADO	CUERPO
200	Eliminación correcta	Nada
500	Si se produce un error eliminando el archivo.	Nada

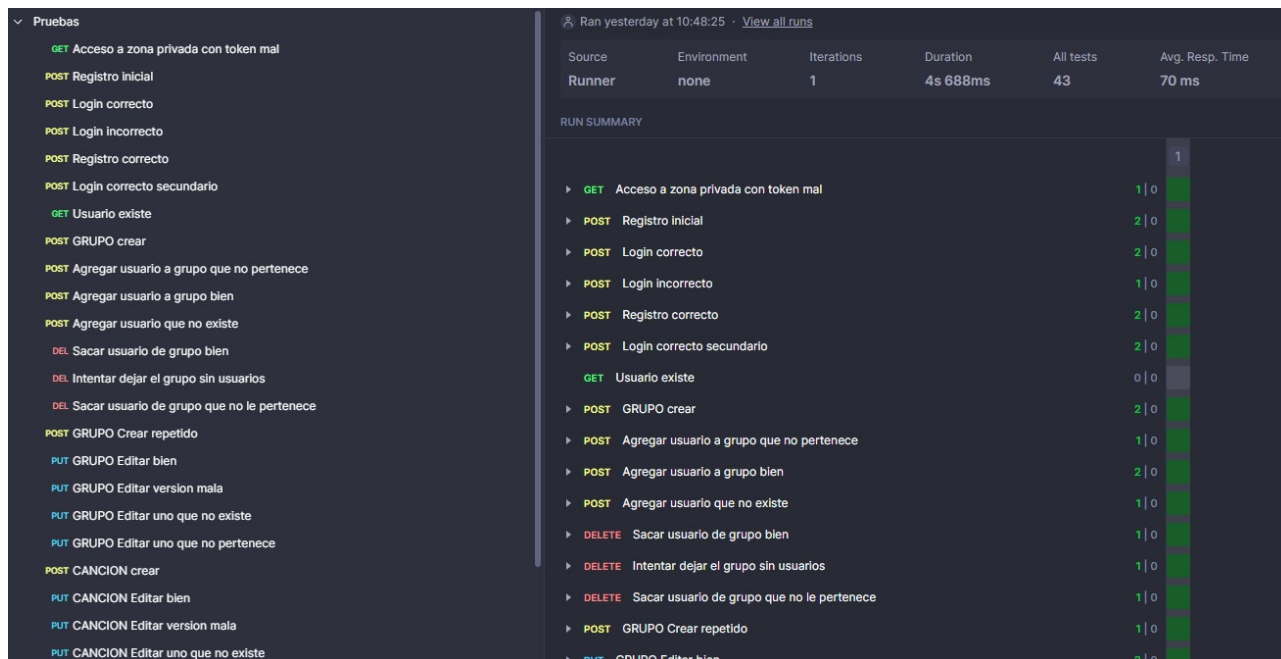
Otros endpoints

RUTA	/usuario/grupos	
USO	Descargar estructura JSON con todos los grupos y sus descendientes pertenecientes al usuario que hace la petición.	
MÉTODO	GET	
AUTHORIZATION	Bearer token	
CUERPO	Nada.	
RESPUESTAS		
CODIGO	SIGNIFICADO	CUERPO
200	Descarga correcta	Respuesta JSON con un listado de entides grupo. Cada grupo con sus entidades usuarios y canciones. Las canciones con sus entidades notas y las notas con sus entidades audio.

RUTA	/usuario/existe/{email}	
USO	Comprobar si un email está asociado a algún usuario.	
MÉTODO	GET	
AUTHORIZATION	Bearer token	
CUERPO	Nada. Variable en ruta.	
RESPUESTAS		
CODIGO	SIGNIFICADO	CUERPO
200	El usuario existe	JSON con los datos públicos del usuario
404	El usuario no existe	Nada.

4.7 Pruebas realizadas

Para la realización de pruebas del servicio web se ha usado Postman con un flujo de peticiones encadenadas que van visitando cada endpoint. Cada petición tiene un script post-response con una batería de pruebas que analizan la respuesta. El diseño del flujo comprueba la creación de varios usuarios y la interacción de estos creando usuarios, grupos, canciones, notas y audios que se van modificando y eliminando. Los scripts van comprobando si se responde según lo esperado a los casos de creación, edición y eliminación correctas, los casos de entidades inexistentes, de entidades con versiones equivocadas y con o sin permisos por parte del usuario cubriendo todas las respuestas que pueden ofrecer los endpoints.



Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	none	1	4s 688ms	43	70 ms

RUN SUMMARY	
	1
▶ GET Acceso a zona privada con token mal	1 0
▶ POST Registro Inicial	2 0
▶ POST Login correcto	2 0
▶ POST Login incorrecto	1 0
▶ POST Registro correcto	2 0
▶ POST Login correcto secundario	2 0
▶ GET Usuario existe	0 0
▶ POST GRUPO crear	2 0
▶ POST Agregar usuario a grupo que no pertenece	1 0
▶ POST Agregar usuario a grupo bien	2 0
▶ POST Agregar usuario que no existe	1 0
▶ DELETE Sacar usuario de grupo bien	1 0
▶ DELETE Intentar dejar el grupo sin usuarios	1 0
▶ DELETE Sacar usuario de grupo que no le pertenece	1 0
▶ POST GRUPO Crear repetido	1 0
▶ PUT GRUPO Editar bien	2 0

Figura 17: Visión parcial del flujo de peticiones de las pruebas del servidor

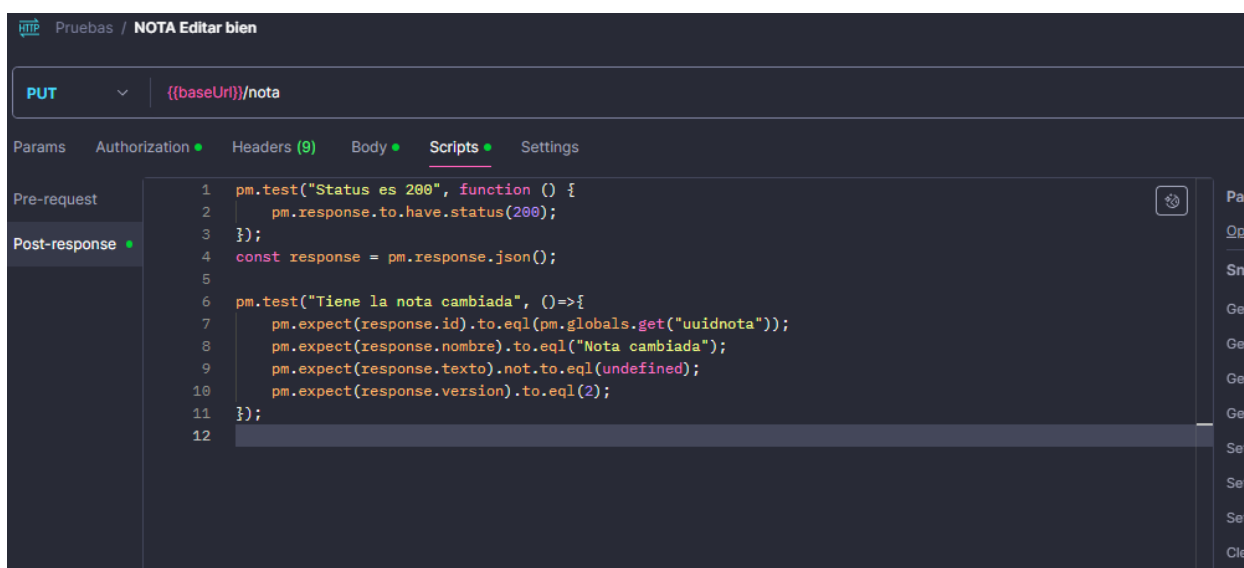


Figura 18: Detalle de script post-response de una de las pruebas

4.8 Requisitos y manual de despliegue

4.8.1 Configuración y despliegue CON Docker (recomendada)

Si se opta por el despliegue con Docker es necesario tener instalado Docker y Docker-compose.

Junto al código fuente del servidor de la aplicación se suministran dos scripts y un archivo de entorno para hacer el despliegue con Docker.

Se trata de los archivos *"Dokerfile"*, *"docker-compose.yml"* y *".env"*.

En el archivo *".env"* se pueden configurar los parámetros del servidor y de los contenedores de Docker. Es utilizado por *"docker-compose.yml"* para crearlos y por el propio archivo de configuración de la aplicación(application.yml) para definir los parámetros de trabajo.

El script de docker-compose.yml genera dos imágenes: Una con la aplicación del servidor de la aplicación y otra imagen con un servidor MySQL.

Los parámetros de la configuración de *.env* son:

- **MYSQL_HOST:** Host de la base de datos. Por defecto el nombre del contendor mysqlldb
- **MYSQL_HOST_PORT:** Puerto expuesto en el host para el servidor MySQL
- **MYSQL_DATABASE:** Nombre de la base de datos
- **MYSQL_ROOT_PASSWORD:** Password para root en MySQL
- **MYSQL_USER:** Usuario de MySQL
- **MYSQL_PASSWORD:** Password de MySQL
- **RUTA_DB_HOST:** Ruta del host al que se mapearán los datos de persistencia de MySQL
- **RUTA_ALMACENAMIENTO_API:** Ruta de almacenamiento dentro de la imagen Docker para el servidor de la aplicación
- **RUTA_ALMACENAMIENTO_HOST:** Ruta de almacenamiento del host al que se vinculará el volumen Docker al que se mapeará RUTA_ALMACENAMIENTO_API
- **PUERTO_API:** Puerto usado por la aplicación.
- **PUERTO_API_MYSQL:** Puerto que usará la aplicación como puerto de acceso a MySQL
- **KEY_ENCRYPTACION:** Clave de encriptación que se usará para los token JWT

Una vez definido el archivo *.env* debe asegurarse el haber compilado el proyecto de Maven y que se ha generado el archivo *"target/ensayos_server.jar"*. Tras eso se pueden construir las imágenes y levantar los contenedores yendo a la carpeta del proyecto y ejecutando la instrucción:

```
docker-compose up -build
```

Se construirá la imagen Docker para el servidor de la aplicación, se descargará una imagen Docker de MySQL y levantará un contenedor para cada imagen.

La imagen de la aplicación necesita que el servidor de MySQL esté completamente levantado. Hasta que eso no suceda la aplicación reintentará la conexión. Una vez iniciados ambos contenedores completamente el servidor estará listo para operar.

Log de salida de la aplicación una vez está operando correctamente:

```
2024-05-22 21:46:58
2024-05-22 21:46:58
2024-05-22 21:46:58  .  _ _ _ _ _
2024-05-22 21:46:58  /\ / _ _ _ _ _ ( _ ) _ _ _ _ _ \ \ \ \ \
2024-05-22 21:46:58  ( ( ) \ _ _ _ _ _ | ' _ | ' _ | ' _ | ' _ \ _ _ | \ \ \ \ \
2024-05-22 21:46:58  \ \ / _ _ _ _ _ | | _ | | | | | | | ( | | ) ) ) )
2024-05-22 21:46:58  ' | _ _ _ _ _ . _ | _ | _ | _ | _ \ _ _ , / / / / /
2024-05-22 21:46:58  =====|_|=====|_|/_/_/_/_/_
2024-05-22 21:46:58  :: Spring Boot ::                (v3.2.4)
2024-05-22 21:46:58

[...]

2024-05-22 21:47:02 2024-06-02T19:47:02.797Z INFO 1 --- [Ensayos Servidor] [main]
o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path ''
2024-05-22 21:47:02 2024-06-02T19:47:02.863Z INFO 1 --- [Ensayos Servidor] [main]
j.ensayos.servidor.ServidorApplication : Started ServidorApplication in 4.24 seconds (process running for
4.599)
```

4.8.2 Configuración y despliegue SIN Docker

Si se opta por desplegar el servidor sin Docker se debe tener Java 19 instalado. También un servidor de la base de datos MySQL o bien MaríaDB (Se ha probado con ambos).

Antes de compilar el servidor se debe establecer la configuración. Para ello hay que ir al archivo:

src/main/resources/aplication.yml

En este archivo hay que configurar principalmente:

- server: Definir puerto y dirección de escucha en la que se quiere poner el servidor a trabajar.
- secret-key: Clave de encriptación para los token JWT
- datasource: Parámetros de conexión con la base de datos
- logging.file.name: Ruta en la que se almacenarán los archivos log
- almacenamiento.ruta: Ruta en la que se almacenarán los archivos de audio

Una vez configurados esos aspectos compilar para generar el JAR y lanzarlo con la instrucción

```
java -jar ./target/ensayos_server.jar
```

5 – El cliente

5.1 Descripción general y tecnologías usadas en el cliente

El cliente es una app nativa de Android desde la cual los usuarios pueden registrarse en el sistema, e introducir y modificar información de grupos, canciones, notas y grabar y reproducir audio. Además pueden compartir las notas con texto y con audio mediante las aplicaciones de mensajería presentes en el móvil. Le pueden dar un uso offline y cuando lo deseen pueden sincronizar la información con el servidor. También pueden incluir a otros usuarios en los grupos musicales en los que están y estos usuarios tendrán acceso al grupo, pudiendo modificar la información referente a este. De este modo los usuarios podrán compartir y modificar la información mediante ediciones offline y sincronización de los datos con el servidor.

La aplicación está implementada en Java con un **target SDK 33**. Para la persistencia local usa el sistema de archivos del propio teléfono en el almacenamiento específico de la app evitando el requerimiento de permisos por parte del usuario. También usa una base de datos **SQLITE** a la cual se accede mediante el ORM **Room**. La comunicación con el servidor mediante **HTTP** se hace usando la librería **Retrofit**. Para las conversiones de entidades **Java ↔ JSON** se usa la librería **Gson**.

Las credenciales de acceso tras el registro/login se guardan en **SharedPreferences** encriptadas.

La aplicación está diseñada con navegación mediante **fragmentos** que se comunican con el modelo siguiendo el **patrón MVVM** y un uso en todos ellos de **LiveData**.

El acceso a los elementos de los layouts desde todos los fragmentos se hace usando **viewBinding**.

Una sola **Activity** se encarga de toda la aplicación exceptuando el grabador de sonido que está implementado en una **Activity** independiente.

5.2 Arquitectura interna del cliente

El cliente es una aplicación nativa de Android implementada en Java. Una activity principal se encarga de cargar un layout con un contenedor de fragmentos. Para cada sección de la aplicación se va cargando un fragmento según un mapa de navegación.

La arquitectura general de la app se establece según el patrón MVVM. Cada fragmento tiene asociado un layout y un Viewmodel que lo comunica con las diferentes partes del modelo.

En el modelo se pueden distinguir las siguientes partes/paquetes:

- **archivos:** Repositorio de acceso a archivos de audio.
- **database:** Acceso a base de datos mediante ORM Room.
- **grabacion:** Lógica de grabación de sonido.

- **network:** Servicios y repositorio de acceso al servicio web.
- **sharedpreferences:** Repositorio de acceso a las SharedPreferences.
- **sincronización:** Servicio para la sincronización con el servidor. Esta parte hace uso del resto de partes del modelo.

Los diferentes Viewmodel hacen uso de esas partes del modelo según necesitan y ofrecen los datos mediante LiveData a los fragmentos que observando esos LiveData actualizan los datos en los layouts de las vistas.

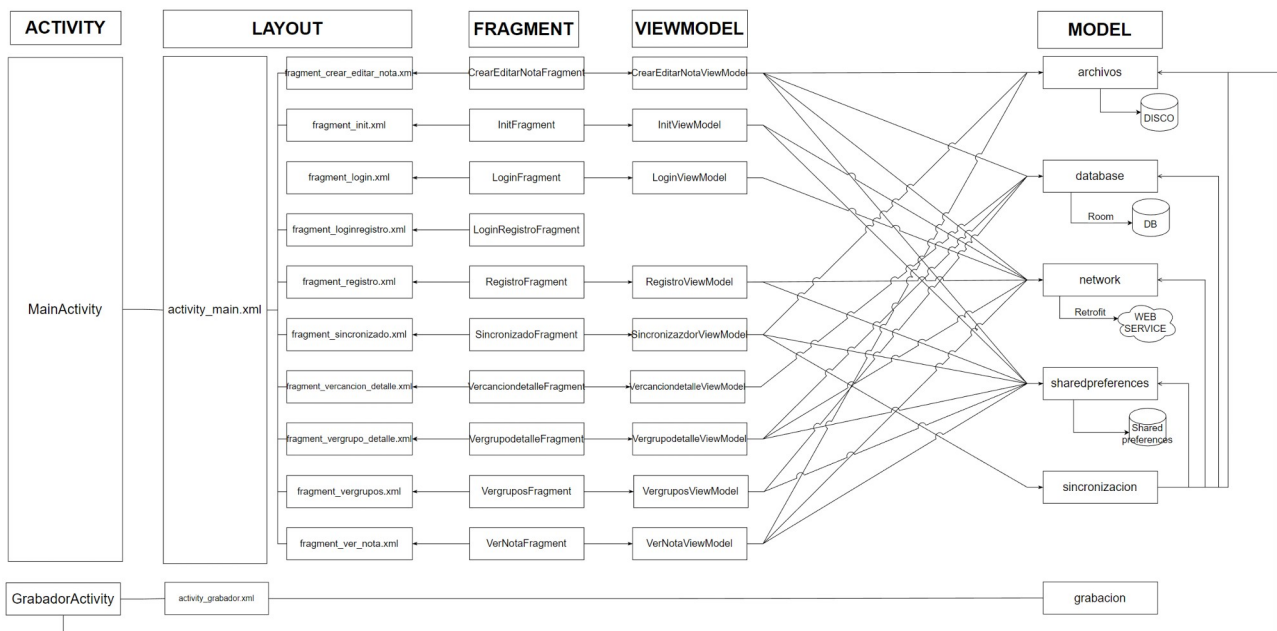


Figura 19: Relación entre activities, fragments, layouts, viewmodels y packages del modelo

La aplicación pide en un primer momento el registro o el inicio de sesión. Cuando se ha realizado correctamente guarda las credenciales de acceso en las **Shared Preferences** de manera encriptada.

Durante el uso de la aplicación el usuario puede ir generando y modificando datos de manera **offline** que se van guardando en la **base de datos local** de la app y el sistema de archivos del teléfono para los archivos de audio.

Cuando el usuario decide **sincronizar** entonces se pone en marcha el algoritmo de sincronización que recoge los datos que hay en el servidor y actualiza la **base de datos local** según se necesita. Además hace las llamadas al **webservice** para actualizar datos remotos según las modificaciones realizadas de forma offline.

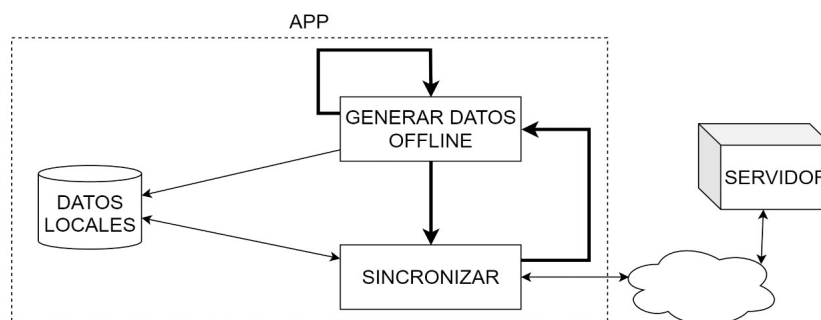


Figura 20: Flujo temporal de datos

5.2.1 Estructura de los paquetes del modelo

La estructura interna de cada parte del modelo es la siguiente:

- **Archivos:** Una clase *ArchivosRepo.java* que contiene toda la lógica de acceso a los archivos locales de audio para guardarlos y recogerlos desde el sistema de archivos propio de la app.
- **Database:** A través de Room se genera una instancia de la base de datos en la clase *AppDatabase*. Desde esa clase se pueden obtener los DAO de las entidades que están especificadas en el subpaquete *entity*. Los Viewmodel no acceden a todo esto directamente si no a través de unos servicios que ofrecen el acceso tanto de manera síncrona como de manera asíncrona a la base de datos. Esos servicios son *DatosLocalesAsincronos* y *DataosLocalesSincronos*. Esos servicios se encuentran en el paquete *database.service*.
- **Grabación:** Este paquete contiene la lógica de un grabador y un reproductor de audio a los que se accede a través de las interfaces *Grabador* y *Reproductor*. Las implementaciones se encuentran en *GrabadorImpl* y *ReroductorImpl*.
- **Network:** Este paquete es el encargado de acceder al webservice del servidor. Lo hace a través de una instancia de Retrofit. Los Viewmodel hacen uso de este paquete a través del servicio implementado en *network.service.APIService* y de los repositorios específicos *AudioApiRepo*, *AuthApiRepo* y *UsuarioApiRepo*.
- **Sharedpreferences:** Este paquete es un envoltorio de las shared preferences normales pero trabajando de manera encriptada y con algunos métodos de acceso rápido a algunos campos como los referentes a las credenciales.
- **Sincronizacion:** Este paquete contiene toda la lógica de sincronización. El punto de acceso para los Viewmodel es la clase *SincronizadorService* la cual genera un hilo independiente y a través de una interfaz de callback *ISincronizadorFeedbackHandler* va avisando del progreso y estado de la sincronización. Se sirve de las clases especificadas en los subpaquetes *comprobadores*, *conflictos* y *excepciones* para realizar su trabajo así como del resto de paquetes del modelo para efectuar sus operaciones.

5.2.2 Activities de la app

Toda la aplicación está contenida en *MainActivity* exceptuando la grabación de sonido. Cuando el usuario pide adquirir un sonido se le da a elegir entre seleccionar un audio desde archivo o usar una aplicación de grabación de sonido que haya en el dispositivo. La propia App de ensayos suministra su grabador de sonido que se encuentra implementado en la activity *GrabadorActivity* permitiendo así la grabación aun cuando el dispositivo carezca de aplicación para grabación de sonido.

5.2.3 Permisos de la app

La aplicación usa los permisos *INTERNET* y *RECORD_AUDIO*. El almacenamiento se hace en el área propia de la aplicación lo que evita tener que pedir permisos al usuario haciendo más ameno el uso de la app. Solamente se piden permisos para la grabación de sonido y solo si se usa el grabador incorporado en la propia aplicación.

5.3 Modelo y base de datos del cliente

La base de datos del cliente es una adaptación de la que hay en el servidor. Al igual que la del servidor, y por los mismos motivos ya expuestos, la claves primarias (exceptuando los usuarios) son de tipo UUID.

Además de los campos que hay en la base de datos del servidor se agregan los campos *borrado*, *abandonado* y *editado* que permiten un borrado lógico de las entidades y registrar si las entidades han sido modificadas. Esto es necesario como se verá en secciones posteriores para el algoritmo de sincronización.

5.3.1 Diseño conceptual de BD en el cliente

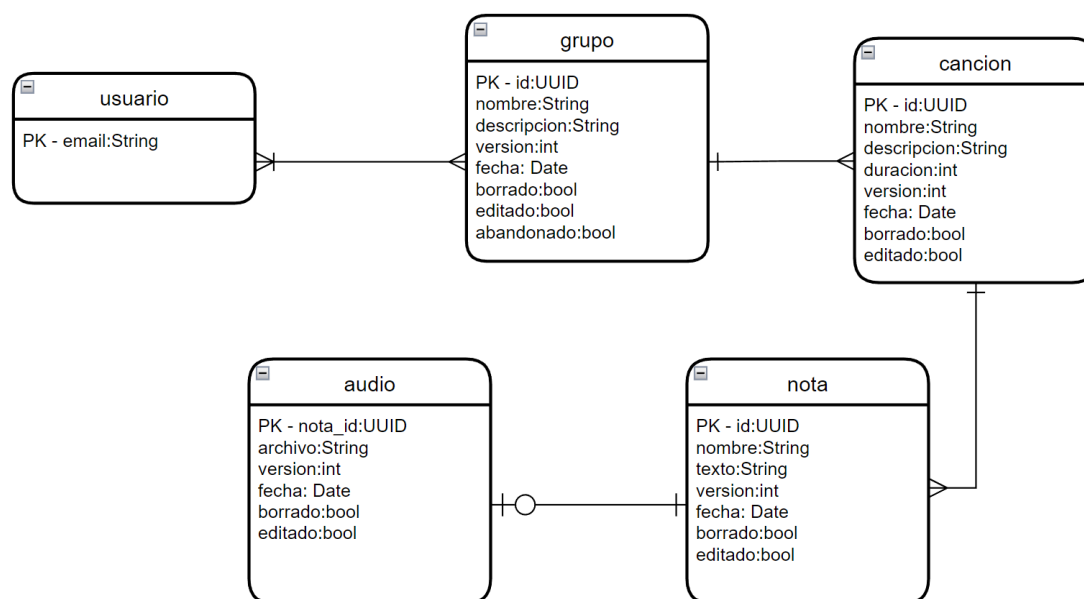


Figura 21: Diseño conceptual BD cliente

5.3.2 Diseño lógico de BD en el cliente

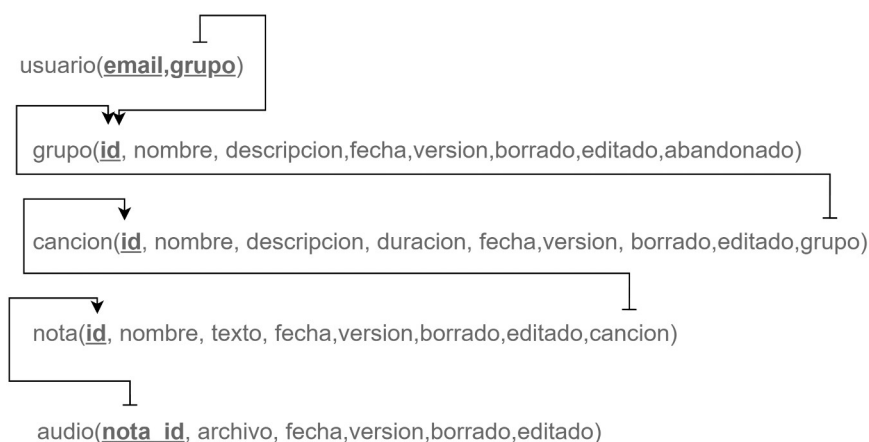


Figura 22: Diseño lógico BD cliente

5.4 Algoritmo de sincronización

El algoritmo de sincronización del sistema es ejecutado por la aplicación móvil.

5.5 Almacenamiento en el cliente

El almacenamiento en el cliente se produce en la base de datos antes mencionada para las entidades lógicas, en las Shared Preferences de la aplicación para guardar de manera encriptada las credenciales de acceso al servidor y directamente en disco en el área propia de la app para los archivos de audio.

5.6 Casos de uso del cliente

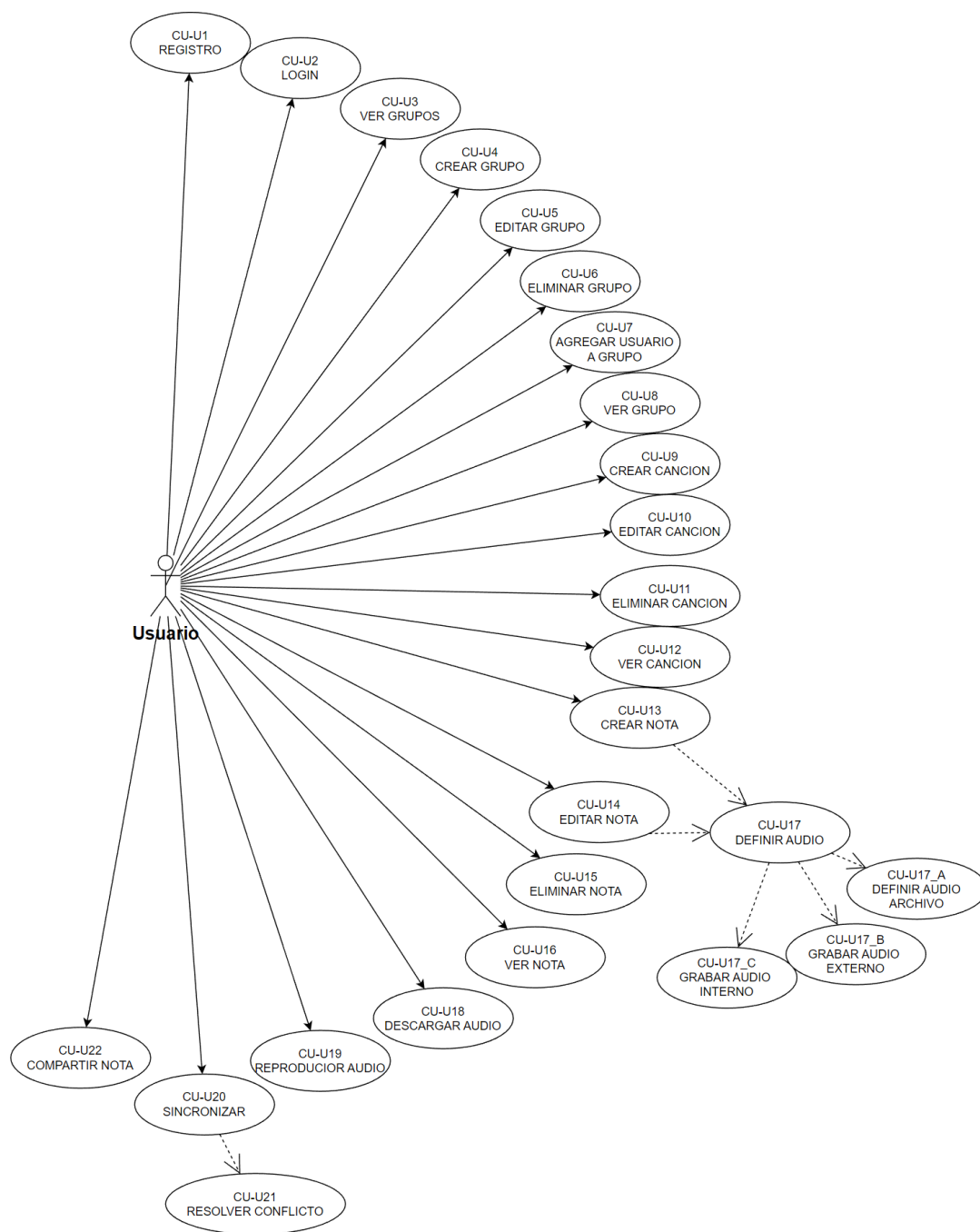


Figura 23: Casos de uso del cliente

Características principales de los casos de uso:

- **CU-U1 REGISTRO:** Registro de un nuevo usuario. Se le presenta al usuario una interfaz desde la que introducir los datos de registro y enviar al servidor esos datos verificando si se ha podido hacer o el mail estaba ocupado. Si es correcto se inicia sesión directamente.
- **CU-U2 LOGIN:** Se le presenta al usuario un formulario de inicio de sesión. El usuario lo rellena y se comprueban los datos de las credenciales aportadas. Una vez iniciada la sesión el usuario no debe volver a iniciarla.
- **CU-U3 a CU-U16:** Creación, modificación, borrado y visualización de diferentes entidades:
 - **Si es creación o modificación:** Se le muestra al usuario un formulario para ejecutar la acción. El usuario lo rellena y se ejecuta la acción. Se verifica que los datos de cada entidad son los mínimos necesarios y se guardan localmente los datos.
 - **Si es eliminación:** El usuario ordena eliminar una entidad. Se pide confirmación y si lo confirma se hace una eliminación local de la entidad según el algoritmo de sincronización. Puede ser borrado lógico o definitivo según la entidad tenga versión superior a 0 o 0 (ver detalle en sección de sincronización).
- **CU-U17 DEFINIR AUDIO:** El usuario pide definir un audio y se le muestra a elegir entre, elegir un archivo CU-U17_A, usar un método externo a la app para grabar sonido CU-U17_B y usar la grabadora propia de la aplicación CU-U17_C
- **CU-U17_C:** Se le muestra al usuario un grabador de sonido con el que puede generar una grabación con el dispositivo.
- **CU-U18 DESCARGAR AUDIO:** Si un usuario no tiene un archivo ordena descargarlo. Se descarga de audio haciéndolo disponible para poder ser escuchado.
- **CU-U19 REPRODUCIR AUDIO:** El usuario elige y reproduce un audio.
- **CU-U20 SINCRONIZAR:** El usuario pide sincronizar y los datos locales y remotos se sincronizan agregándose a cada lado (cliente y servidor) lo modificado y nuevo que haya en el otro lado. Si alguna modificación implica un conflicto se pasa al caso de uso CU-U21.
- **CU-U21 RESOLVER CONFLICTO:** Cuando hay un conflicto de sincronización entre una entidad remota y local se le presenta al usuario una interfaz para solventar ese conflicto sin que se pierdan datos a menos que el usuario así lo elija. El usuario debe poder ver los datos de ambas versiones para poder decidir la solución. Especifica cuál es la solución y se resuelve el conflicto.
- **CU-U-22 COMPARTIR NOTA:** El usuario comparte a través de una app externa la información y audio de una nota y la canción a la que pertenece.

La interfaz de usuario está compuesta por pantallas completas y diálogos. Cada pantalla está definida como un fragmento dentro de un mapa de navegación pudiendo ir de pantalla a pantalla según las opciones para ir pudiendo completar las tareas. Tanto las pantallas como los diálogos están especificados en layouts XML exceptuando los dialogos de creación de grupos y canciones que están implementadas por código.

Aquí un boceto de la interfaz gráfica realizado al inicio del desarrollo de la aplicación de cliente.



5.7.2 Mapa de navegación

Las pantallas de la aplicación están estructuradas como una navegación entre fragmentos. Esta navegación está implementada sobre un mapa de navegación que define desde qué fragmentos se puede ir a qué otros fragmentos, cuales son los puntos iniciales del stack de vistas y las transiciones animadas cuando se efectúa la navegación.

Aquí la versión visual del mapa de navegación en Android Studio:

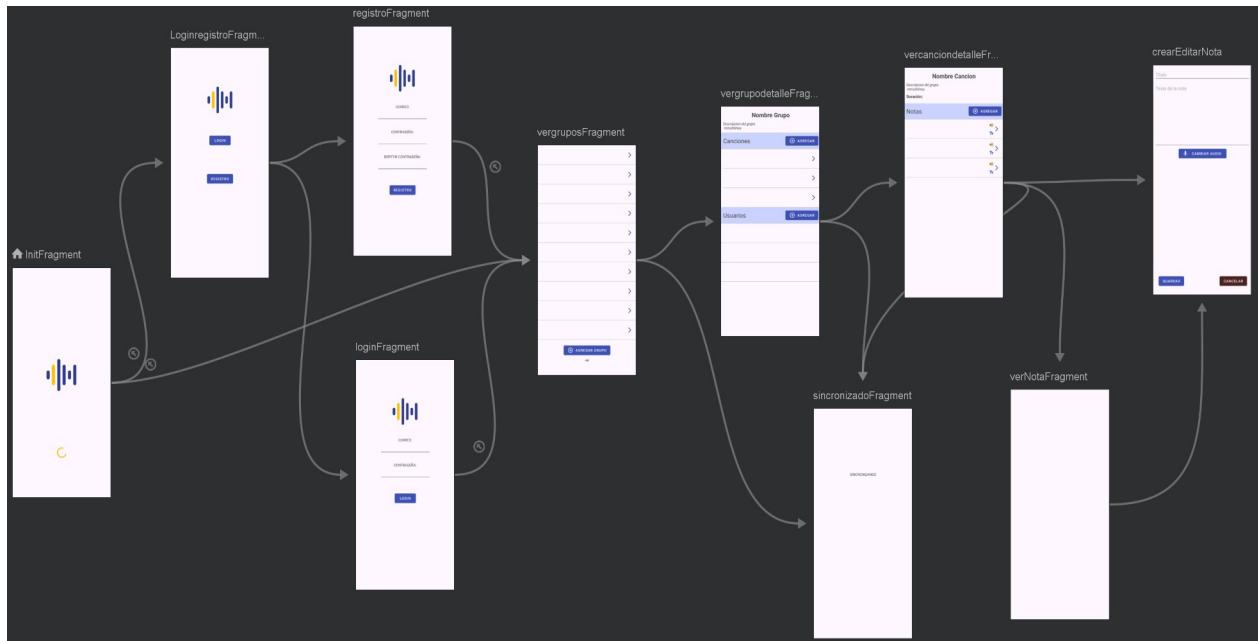


Figura 25: Mapa de navegación de fragmentos de Android Studio

5.7.3 Recursos gráficos

Los iconos que aparecen dentro de la aplicación han sido generados como drawables vectoriales directamente desde android studio. El icono de la aplicación se ha construido a partir de un icono de dominio público modificado y animado siendo luego exportado como Animated Vector Drawable para ser usado durante el splash screen.

5.8 Pruebas realizadas

Las pruebas de la aplicación se han realizado en emuladores api 28-34 y dispositivos reales api 30-34 conforme se iteraban los ciclos del desarrollo de la aplicación. Las pruebas se han realizado por parte tanto mía como de usuarios finales que han aportado información importante de detección de errores y de mejoras de usabilidad e interfaz.

Las pruebas de funcionamiento realizadas por mi directamente han consistido en probar las posibilidades de creación, modificado y eliminación de entidades en casos tanto locales de la aplicación como de su integración con el servicio web durante la sincronización cubriendo las combinaciones de casos modificados y sin modificar y diferentes atributos buscando los casos límite que pudieran dar lugar a errores.

Las pruebas realizadas por otros usuarios se han centrado en la usabilidad y la comprobación de la compatibilidad con diferentes dispositivos.

5.9 Requisitos de la aplicación

La aplicación se puede instalar en dispositivos con SO Android 9 API 28 en adelante.

Tras el registro inicial o el login, que deben ser con conexión a internet, se puede usar de manera local sin conexión a internet.

Para la sincronización con el servidor, y por tanto con otros usuarios, es necesaria la conexión a internet.

Los permisos que el usuario debe otorgar a la aplicación se reducen a la grabación de sonido y solo en el momento de usar el apartado de grabación propio de la aplicación.

5.10 Manual de uso de la aplicación móvil

5.10.1 Descripción

La aplicación le permitirá gestionar el producto de los ensayos de grupos musicales, canciones de esos grupos y notas de texto y audio asociadas a esas canciones.

Además podrá compartir los grupos con otros usuarios y generar de manera colaborativa la información sobre las canciones, notas y audios.

La aplicación dispone de un grabador de sonido y la capacidad de usar otras fuentes de audio como archivos de sonido presentes en su dispositivo u otros medios de grabación disponibles en el mismo.

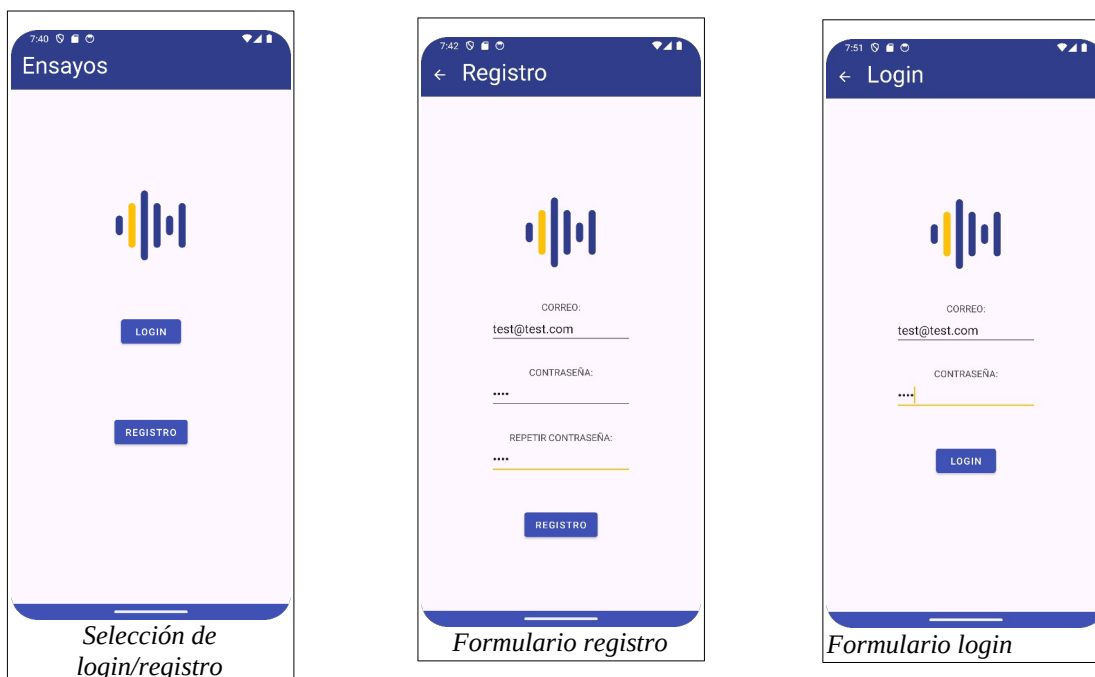
También podrá compartir notas y audios mediante apps de mensajería presentes en su dispositivo.

Puede usar la aplicación sin conexión a internet y cuando desee podrá sincronizar haciendo que la información generada esté disponible para el resto de miembros de los grupos que comparte. A su vez recogerá la información que otros usuarios han agregado a los grupos a los que pertenece.

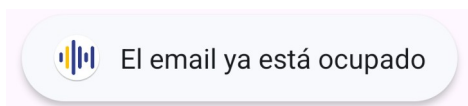
Si instala la aplicación en un nuevo dispositivo podrá recuperar la información que haya sincronizado previamente con el mismo usuario.

5.10.2 Inicio de la aplicación y creación del usuario

Cuando se arranca la aplicación por primera vez es necesario iniciar sesión si ya se tiene usuario o registrar un nuevo usuario si aún no lo tiene. Elija la opción LOGIN o REGISTRO según necesite.



Si intenta registrar un usuario ya existente se le avisará con un mensaje:



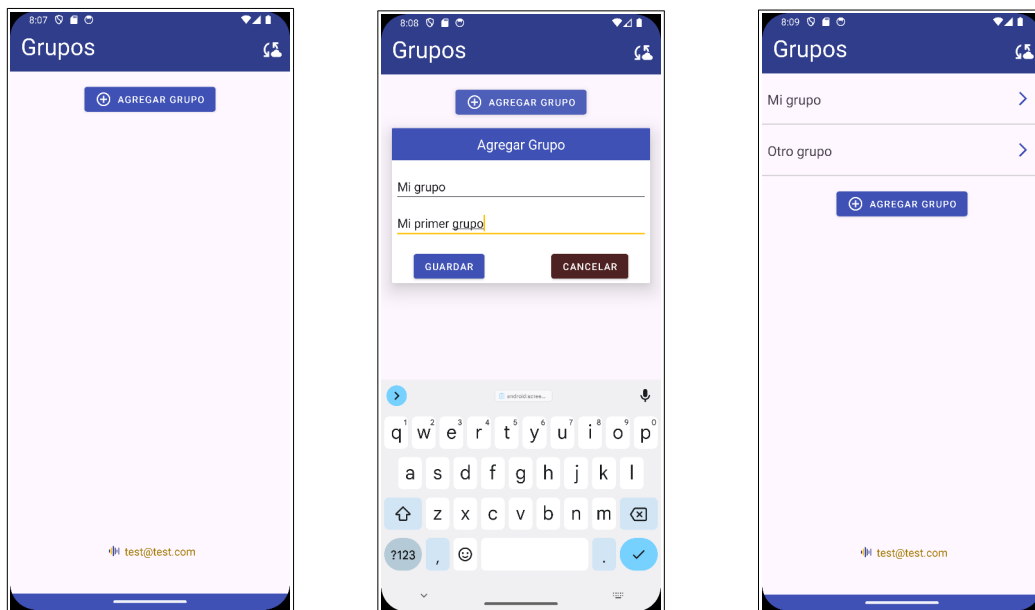
Si ya tiene un usuario creado puede iniciar sesión directamente lo cual le permitirá recuperar la información desde el servidor mediante la sincronización que se verá más adelante.

5.10.3 Listado de grupos

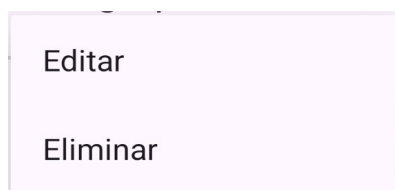
En esta pantalla primera pantalla puede crear grupos.

Pulse el botón AGREGAR GRUPO para crear un nuevo grupo. Se le presentará un formulario para introducir nombre y descripción. El nombre del grupo es obligatorio.

Puede generar los grupos que desee. Los grupos a los que pertenece aparecerán en un listado



Desde esta pantalla puede editar y eliminar grupos. Para ello haga una pulsación larga sobre el grupo que quiere editar/eliminar y se mostrará un menú:



NOTA: Todos los listados de grupos, canciones, usuarios, y notas tienen este menú para lanzar la edición o la eliminación

Para ver en detalle un grupo pulse sobre su nombre.

5.10.4 Ver detalle de grupo, agregar canciones y otros usuarios al grupo

Al entrar en la vista de detalle de un grupo podrá ver los detalles del grupo, las canciones que contiene y los usuarios que tienen acceso al grupo.



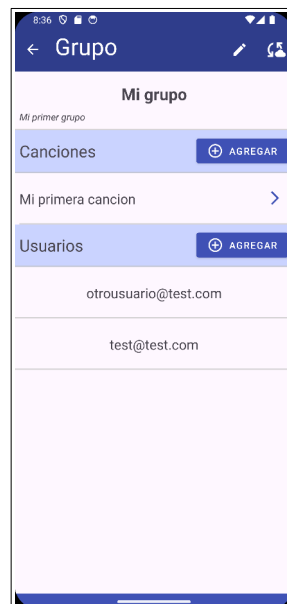
En un principio solo estará su usuario como miembro del grupo. Puede dar acceso al grupo a otros usuarios pulsando en el botón Usuarios AGREGAR. Se le mostrará un formulario donde podrá introducir a otro usuario. Los usuarios que agregue tendrán el mismo acceso al grupo y podrán hacer todas las acciones con el grupo.

NOTA: En cada pantalla de vista de detalle tendrá disponible el botón de editar que permite editar las características principales del grupo, canción o nota que esté viendo.

Ese botón se encuentra en la barra superior:



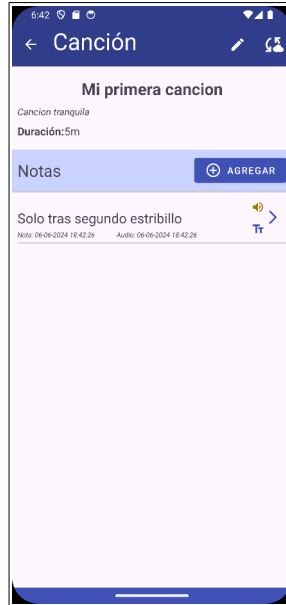
En esta pantalla podrá también agregar canciones al grupo. Para ello pulse el botón Canciones AGREGAR y se le mostrará un formulario para introducir los datos de la canción. Tras rellenar el formulario aparecerá la canción en el listado de canciones del grupo:



Para ver el detalle de una canción pulse su nombre.

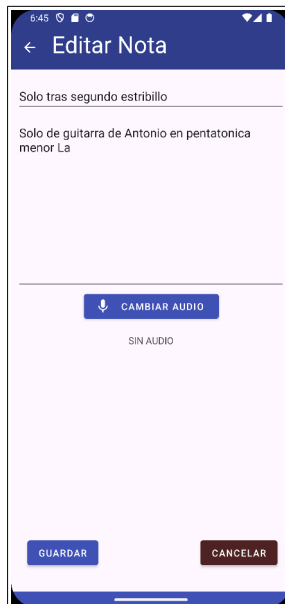
5.10.5 Ver detalle de canción y agregar notas

En la vista de detalle de una canción se puede ver la descripción y la duración. Además se puede ver el listado de notas asociadas a la canción y agregar nuevas notas pulsando el botón NOTAS AGREGAR. Los iconos a la derecha del nombre de una nota muestran si la nota contiene audio, texto o ambas:

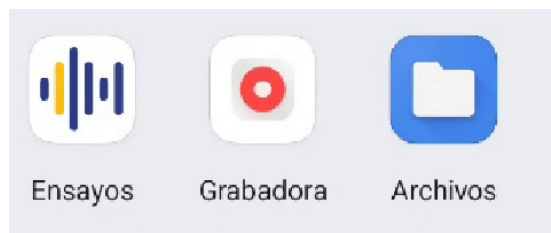


5.10.6 Crear y visualizar notas

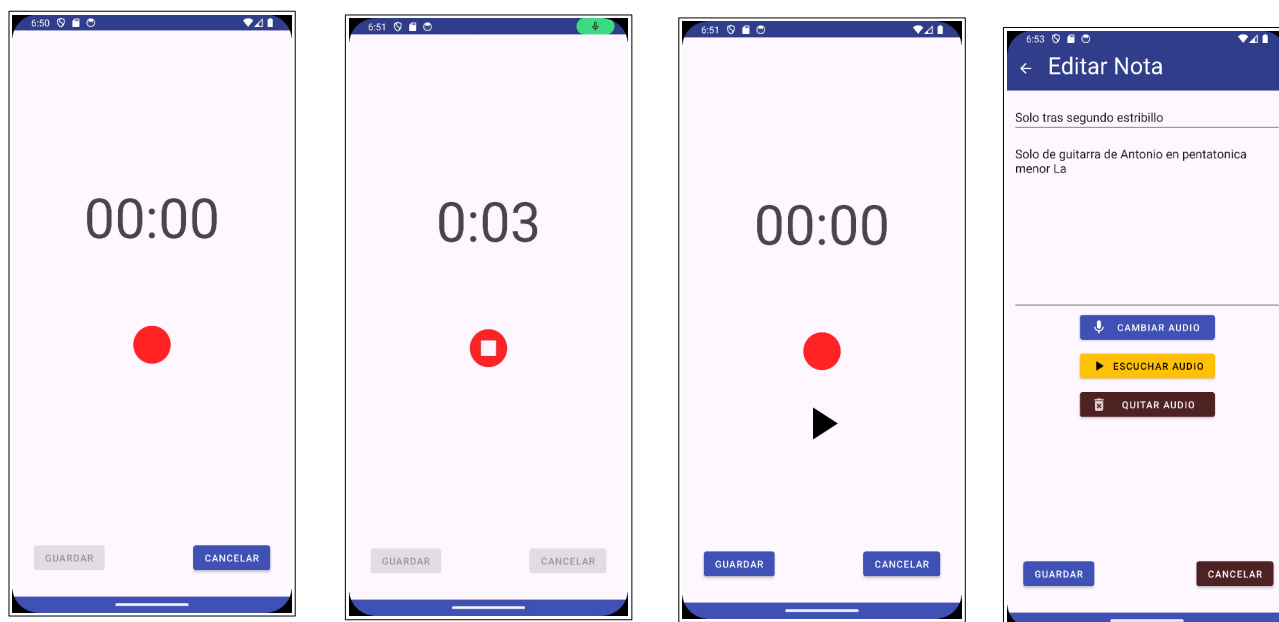
Cuando se va a crear una nota se verá la siguiente pantalla. En ella puede poner un título a una nota, un cuerpo de texto y asociar un sonido pulsando en el botón CAMBIAR AUDIO.



Cuando se va a cambiar un audio se mostrará un menú para elegir entre usar el propio grabador de la aplicación Ensayos, otros grabadores presentes en el dispositivo o elegir un archivo de audio:



El grabador de la aplicación ensayos tiene la siguiente vista. Pulse sobre el círculo rojo para comenzar la grabación y nuevamente en él para terminar la grabación. Cuando se haya grabado se podrá reproducir el sonido. Una vez guardado el sonido se asignará a la nota y se verán botones para escuchar el audio o para eliminarlo.



Cuando tenga el contenido de la nota definido pulse GUARDAR y aparecerá la nota en el listado de notas de la canción.

Para ver la nota en detalle pulse su nombre y le aparecerá su contenido:



En esta pantalla, además de ver el contenido de la nota puede reproducir el audio y compartir la nota con otras aplicaciones pulsando el botón de compartir:



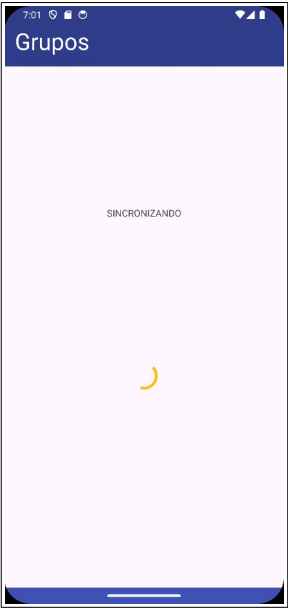
5.10.7 Sincronizar los datos con la nube y otros usuarios

Hasta ahora ha trabajado con los datos en el propio dispositivo. Para que sus datos se almacenen en la nube y estén disponibles para otros usuarios debe pulsar el botón de sincronización que encontrará en la barra superior en casi todas las pantallas:

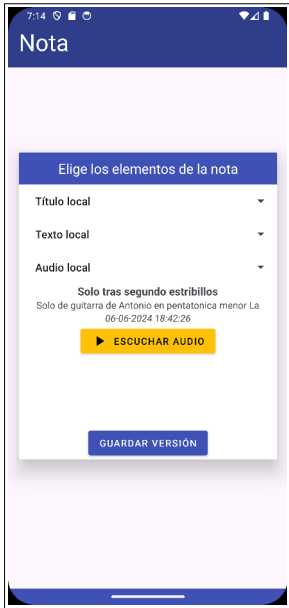


Cuando pulse este botón asegúrese de tener conexión a internet. Los datos se sincronizarán enviando los nuevos datos y cambios en grupos, canciones y notas que haya realizado y recibirá los cambios realizados por otros miembros del grupo.

Durante la sincronización aparecerá un icono de progreso y una serie de mensajes que le avisan de qué fase de la sincronización está realizándose:



Es posible que cuando se edita algún elemento suceda que otro usuario haya editado el mismo elemento. Para solventar ese conflicto durante la sincronización la aplicación le avisará y mostrará una ventana en la que puede elegir cual es el resultado final que quiere que se establezca. Para resolver los conflictos de grupos y canciones se le dará a elegir entre su versión, la versión remota de la nube o una mezcla de ambas. Para las notas se le dará a elegir qué título, cuerpo y audio quiere que se use para solventar el conflicto.



6. Bibliografía

Durante el desarrollo del proyecto se ha buscado información en los siguientes sitios:

- Spring boot: <https://spring.io/projects/spring-boot>
- Spring: <https://spring.io/>
- HTTP: <https://www.rfc-editor.org/rfc/rfc2616>
- HTTP: <https://developer.mozilla.org/en-US/docs/Web/HTTP>
- JWT: <https://datatracker.ietf.org/doc/html/rfc7519>
- Lombok: <https://projectlombok.org/api/>
- Android/Room: <https://developer.android.com/>
- Diverso: <https://stackoverflow.com/>
- Spring: <https://www.baeldung.com/>
- UUID: <https://www.rfc-editor.org/rfc/rfc9562.html>
- Java: <https://docs.oracle.com/en/java/javase/18/>
- Hibernate: <https://hibernate.org/orm/>
- Docker y Docker compose: <https://docs.docker.com/>
- Postman: <https://learning.postman.com/docs/getting-started/overview/>
- Retrofit: <https://square.github.io/retrofit/>
- Diverso: <https://www.youtube.com/>
- Gson: <https://github.com/google/gson>
- Maven: <https://maven.apache.org/>
- Apache web server: <https://httpd.apache.org/>
- Diverso: <https://issuetracker.google.com/issues>