Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки Кафедра
інформаційних систем та технологій

ЛАБОРАТОРНА РОБОТА №2

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Online radio station»

Виконав
Перевірив:

студент групи ІА–33:
Мягкий М.Ю.

Якименко Ярослав

Київ 2025

Вступ

У даній лабораторній роботі розглядаються основи проєктування програмного забезпечення на прикладі розробки архітектури для веб-сервісу "Онлайн-радіостанція".

Метою роботи є опанування ключових інструментів та методологій проєктування, зокрема використання мови UML для візуалізації архітектури, розробки сценаріїв використання для визначення функціональних вимог, а також застосування патернів проєктування для побудови гнучкої системи.

У ході роботи було спроєктовано систему, що дозволяє користувачам слухати аудіопотоки, керувати плейлистами та завантажувати треки. Реалізовано механізм взаємодії з базою даних (MongoDB) для зберігання треків та метаданих користувачів.

Теоретичні відомості

У даній лабораторній роботі для проєктування системи «Веб-сервіс онлайн-радіостанції» були застосовані фундаментальні концепції об'єктно-орієнтованого аналізу та проєктування (ООАП). Це дозволило створити надійну, гнучку та підтримувану систему. Ключовими

інструментами, які було використано, є уніфікована мова моделювання (UML) та принципи проєктування баз даних.

UML — це загальновизнаний стандарт для візуального моделювання програмних систем. За допомогою UML ми можемо специфікувати, візуалізувати та документувати архітектуру програмного забезпечення на різних рівнях абстракції:

1. Концептуальний рівень: На цьому рівні система розглядається з точки зору користувача та бізнес-логіки.

- Для цього була розроблена діаграма варіантів використання (Use Case Diagram).
- Вона визначає основні функціональні вимоги системи та її взаємодію з різними акторами: Guest (Гість), Listener (Слухач), DJ (Діджей) та Admin (Адміністратор).

2. Логічний рівень: Цей рівень описує статичну структуру та динамічну поведінку системи.

- Для опису статичної структури була створена діаграма класів (Class Diagram).
- Вона деталізує класи (User, Track, Playlist, Station, File) та їх взаємозв'язки, що є основою для подальшої реалізації.
- Діаграма також включає сутності-репозиторії (UserRepository, TrackRepository, PlaylistRepository), які відповідають за взаємодію з базою даних.

3. Фізичний рівень: На цьому рівні описується фізичне розгортання компонентів системи.

- У нашому випадку, це включає логічну структуру бази даних, яка була спроєктована на основі логічної моделі.
- Оскільки в реалізації використовується MongoDB, діаграма показує колекції (users, tracks, playlists) та зв'язки між документами через ідентифікатори (ID), що забезпечує цілісність даних та ефективну організацію зберігання медіа-контенту.

Діаграма варіантів використання (Use Case Diagram)

Діаграма варіантів використання є відправною точкою в аналізі вимог. Вона фіксує функціональність системи та її межі. Основними елементами діаграми є:

- Актор (Actor): Будь-яка зовнішня сутність, що взаємодіє із системою (користувач, інша система). В нашому випадку це Guest, Listener, DJ та Admin, кожен з яких має свій набір прав та можливостей.
- Варіант використання (Use Case): Послідовність дій, яку система виконує для досягнення певної мети для актора. Наприклад, Listen to Stream (Слухати потік), Upload Track (Завантажити трек) або Create Playlist (Створити плейлист).
- Зв'язки (Relationships): Описують взаємодію між акторами та варіантами використання. Зокрема, <<include>> показує, що один варіант використання включає в себе інший (наприклад, Upload Track включає File Validation).

Діаграма класів та База даних

Діаграма класів є основним інструментом для моделювання статичної структури системи. Вона візуалізує класи, їхні атрибути (назва треку, виконавець, жанр) та операції, а також зв'язки між ними. Для зберігання даних системи була спроєктована модель бази даних. У даній роботі колекції спроєктовані таким чином, щоб забезпечити швидкий доступ до аудіофайлів та метаданих, використовуючи технологію GridFS для зберігання великих файлів.

Хід роботи

1) Аналіз вимог та проєктування діаграми варіантів використання

На першому етапі проєктування було проведено аналіз функціональних вимог до системи. Було визначено, що веб-сервіс «Онлайн-радіостанція» повинен надавати користувачу інструменти для прослуховування потокового аудіо, керування особистими плейлистами, а також засоби для адміністрування контенту (завантаження треків, керування ефіром).

На основі цього аналізу було виділено ключових акторів, які взаємодіють із

системою:

- Слухач (Listener) — зареєстрований користувач, який може слухати радіостанції, створювати власні плейлисти, додавати треки до «Улюбленого» та переглядати історію прослуховування.
- Діджей (DJ) — особа, яка має розширені права для керування контентом. Вона може завантажувати нові музичні треки, редагувати метадані, формувати розклад ефіру та керувати потоками станцій.
- Адмін (Admin) — особа з найвищими правами, що керує обліковими записами користувачів (зокрема блокуванням порушників) та налаштуваннями системи.
- Гість (Guest) — незареєстрований користувач, який може переглядати список доступних станцій, здійснювати пошук та пройти реєстрацію/авторизацію.
- Автентифікований користувач (Authenticated User) — узагальнений актор, що об'єднує Слухача, Діджея та Адміна. Вони мають доступ до спільних функцій, таких як вихід із системи (Logout) та редагування профілю.

Було визначено наступні варіанти використання, що описують основні функції системи:

- Для Слухача: Listen to Stream (Слухати потік), Create Playlist (Створити плейлист), Add to Favorites (Додати в улюблене).
- Для Діджея: Upload Track (Завантажити трек), Manage Schedule (Керувати розкладом), Delete Track (Видалити трек).
- Для Адміністратора: Manage Users (Керувати користувачами), Block User (Заблокувати користувача).
- Для Гостей: Login (Увійти), Registration (Зареєструватися), Search Station (Пошук станції).

Для відображення логічних залежностей між операціями було використано відношення включення (<<include>>). Наприклад, варіант використання Upload Track включає в себе File Validation (перевірка формату та розміру файлу), а Create Playlist включає Add Track.

Результатом цього етапу є діаграма варіантів використання, що візуалізує

взаємодію різних типів користувачів із системою.

2) Розробка сценаріїв варіантів використання

Для деталізації вимог було розроблено сценарії ключових прецедентів.

Сценарій №1: Прослуховування ефіру радіостанції

- Передумови: Користувач (Слухач або Гість) знаходиться на головній сторінці сервісу.
- Постумови:
    - Успіх: Встановлено з'єднання з сервером, почалося відтворення аудіо.
    - Провал: З'єднання не встановлено, відтворення неможливе.
- Взаємодіючі сторони: Користувач, Система "Radio Service".
- Короткий опис: Користувач обирає радіостанцію зі списку доступних і ініціює відтворення потоку.
- Основний перебіг подій:
    - Користувач переглядає список станцій (за жанрами або популярністю).
    - Користувач натискає кнопку "Play" на обраній станції.
    - Система надсилає запит на отримання потоку (Stream URL).
    - Система буферизує аудіодані та починає відтворення звуку.
    - Відображається інформація про поточний трек (виконавець, назва).
- Винятки
    - Виняток №1: Станція тимчасово недоступна (offline). Система виводить повідомлення: «Ефір наразі відсутній».
    - Виняток №2: Помилка мережі у користувача. Відтворення переривається, з'являється кнопка «Перепідключитися».

Сценарій №2: Завантаження нового треку в базу

- Передумови: Користувач авторизований як Діджей (DJ).
- Постумови:
  - Успіх: Аудіофайл збережено в сховище, метадані записані в БД.
  - Провал: Файл не завантажено.
- Короткий опис: Діджей додає нову музичну композицію до бібліотеки станції.
- Основний перебіг подій:
  - Діджей переходить в «Панель керування» і натискає «Upload Track».
  - Система відкриває діалогове вікно вибору файлу.
  - Діджей обирає файл (формат .mp3 або .wav) та заповнює форму метаданих (Artist, Title, Genre).
  - Діджей натискає «Зберегти».
  - Система валідує файл, завантажує його в GridFS та створює запис у колекції tracks.
  - Система виводить повідомлення: «Трек успішно додано».
- Винятки:
  - Виняток №1: Невірний формат файлу (наприклад, .exe). Система виводить: «Неприпустимий формат файлу».
  - Виняток №2: Розмір файлу перевищує ліміт. Система відхиляє завантаження.

3) Проєктування діаграми класів предметної області

На наступному етапі було створено модель предметної області, яка відображає основні сутності системи та зв'язки між ними. Ця модель дозволяє зрозуміти логічну структуру даних без урахування технічних деталей реалізації.

Центральними сутностями виступають:

- User (Користувач): Представляє будь-якого актора (Слухач, DJ, Адмін). Містить облікові дані та роль.
- Track (Трек): Сутність, що описує музичний файл. Включає атрибути: назва, виконавець, тривалість, жанр та посилання на фізичний файл.
- Playlist (Плейлист): Колекція треків, створена користувачем. Має назву та список ідентифікаторів треків.
- Station (Станція): Описує канал мовлення (назва, URL потоку, жанровий напрямок, поточний статус).
- File (Файл): Загальна сутність для зберігання бінарних даних (аудіофайли, обкладинки альбомів).

Зв'язки:

- Користувач може мати багато Плейлистів (One-to-Many).
- Плейлист може містити багато Треків (Many-to-Many).
- Кожен Трек завантажується конкретним Користувачем (Діджеєм).
- Станція транслює потік Треків.

4) Проєктування структури бази даних та реалізація патерну Repository

Для зберігання даних про користувачів, музичні треки та плейлисти було спроєктовано структуру бази даних (на базі MongoDB).

Основні колекції (таблиці):

- User: Зберігає інформацію про користувачів (id, username, password, email, role).
- Track: Містить метадані музичних творів (id, title, artist, genre, duration, ownerId, fileId). Поле fileId посилається на реальний файл у GridFS.
- Playlist: Зберігає списки відтворення (id, title, ownerId, список trackIds).
- File (fs.files / fs.chunks): Використовується механізмом GridFS для зберігання великих аудіофайлів, розбиваючи їх на частини (chunks).

Для взаємодії з цією базою даних було реалізовано патерн проєктування

Repository. Він дозволяє ізолювати бізнес-логіку застосунку від конкретних деталей роботи з БД (Spring Data MongoDB). Були створені інтерфейси UserRepository, TrackRepository, PlaylistRepository, які надають методи для пошуку, збереження та видалення даних (наприклад, findAllByGenre, findByOwnerId).

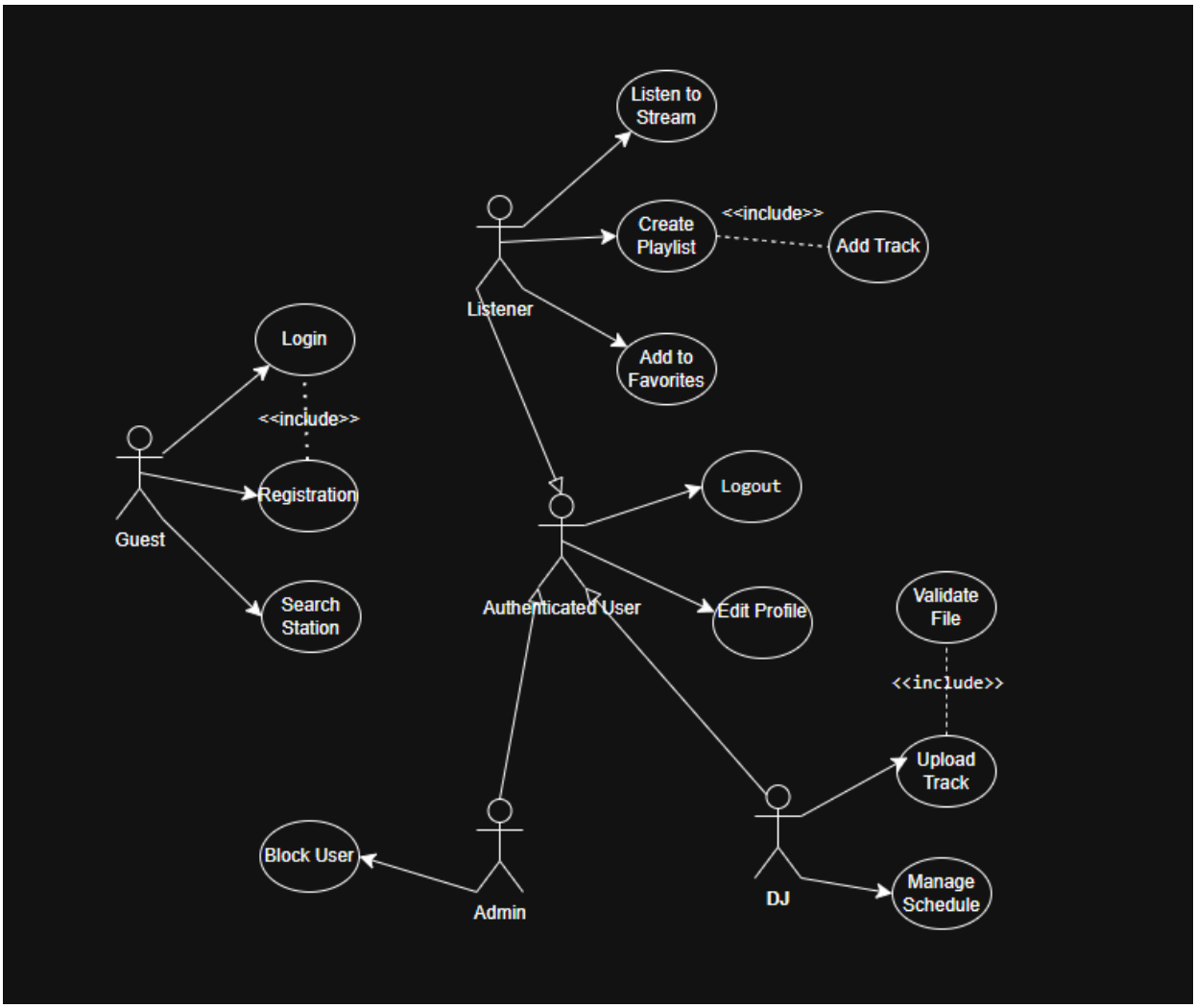

Рис. 1. Рис. 1. Діаграма варіантів використання системи «Online radio station»

1) Розробка сценаріїв варіантів використання

Сценарій №1: Прослуховування радіостанції Передумови: Користувач (Слухач або Гість) знаходиться на головній сторінці сервісу. Постумови:

- Успіх: Встановлено з'єднання з сервером, почалося відтворення аудіо.
- Провал: З'єднання не встановлено, система вивела повідомлення про помилку. Взаємодіючі сторони: Користувач, Система "Radio Service".

Короткий опис: Користувач обирає радіостанцію зі списку доступних і ініціює відтворення потоку.

Основний перебіг подій:

1. Користувач переглядає список станцій і натискає кнопку "Play" на обраній станції.
2. Система надсилає запит на отримання потоку (Stream URL).
3. Система буферизує аудіодані.
4. Починається відтворення звуку, відображається назва поточного треку.

Винятки:

1. Виняток №1: Станція офлайн. Система виводить: «Ефір наразі відсутній».
2. Виняток №2: Втрачено інтернет-з'єднання. Відтворення зупиняється.

Сценарій №2: Завантаження нового треку (Upload Track) Передумови: Користувач авторизований як Діджей (DJ). Постумови:

- Успіх: Файл збережено в GridFS, метадані додано в БД.
- Провал: Файл не завантажено. Короткий опис: Діджей додає нову композицію в базу даних станції.

Основний перебіг подій:

1. Діджей натискає кнопку «Upload Track».
2. Система відкриває вікно вибору файлу.
3. Діджей обирає файл (.mp3) та вводить назву і виконавця.
4. Діджей натискає «Зберегти».
5. Система виконує варіант використання Validate File (перевіряє формат і розмір).
6. Система зберігає файл і виводить повідомлення «Успішно».

Винятки:

- Виняток №1: Обрано файл формату .exe або .doc. Система виводить

помилку валідації.

- Виняток №2: Розмір файлу перевищує 20 МБ. Система відхиляє завантаження.

Сценарій №3: Створення плейлиста Передумови: Користувач авторизований як Слухач (Listener). Постумови:

- Успіх: Створено новий плейлист, до нього додано перший трек. Основний перебіг подій:

1. Слухач натискає «Create Playlist».
2. Система запитує назву плейлиста.
3. Слухач вводить назву і натискає «Створити».
4. Система створює порожній плейлист в базі даних.
5. Слухач виконує дію Add Track (додає трек до новоствореного списку).
6. Система оновлює список треків плейлиста.

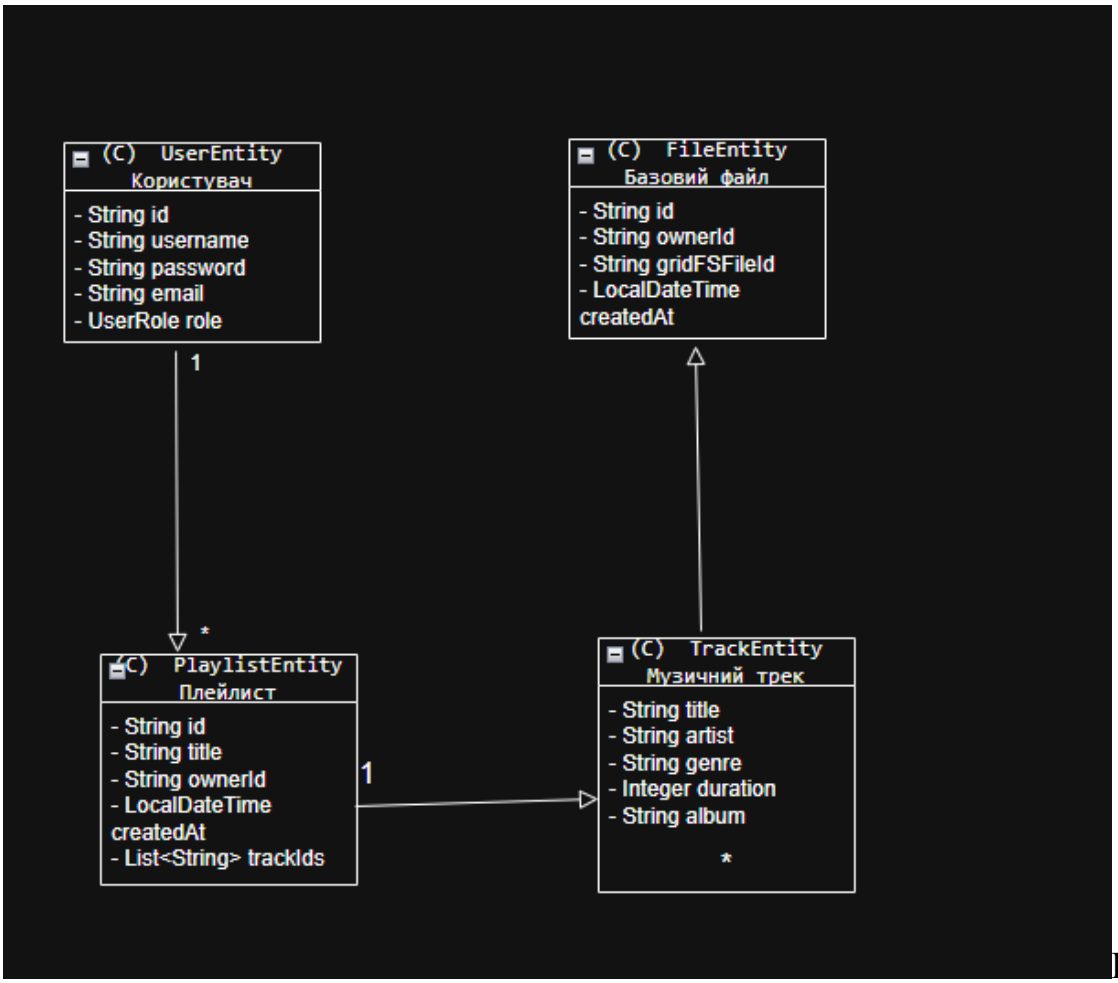

Рис. 2. Діаграма класів предметної області

4) Проєктування структури бази даних та реалізація патерну Repository

Для зберігання метаданих про радіостанції, користувачів, музичні треки та плейлисти було спроєктовано базу даних. Її структура складається з кількох основних колекцій (таблиць), що відображають сутності, визначені в моделі предметної області:

- User: Зберігає інформацію про всіх користувачів системи (ім'я, email, пароль, роль).
- Track: Містить деталі про кожен музичний трек (назва, виконавець, жанр, тривалість, посилання на аудіофайл).
- Playlist: Фіксує створені користувачами списки відтворення, посилаючись на відповідні треки та власника плейлиста.

- File: Використовується для зберігання метаданих про бінарні файли (аудіо), завантажені в систему через GridFS.

Між цими таблицями встановлено відношення типу «один-до-багатьох» (one-to-many), що забезпечує цілісність даних. Наприклад, один користувач може мати багато плейлистів, а один плейлист містить посилання на багато треків.

Для взаємодії з цією базою даних було реалізовано патерн проєктування Repository. Він дозволяє ізолювати бізнес-логіку застосунку від конкретних деталей роботи з БД, надаючи зручні інтерфейси для пошуку та збереження даних.

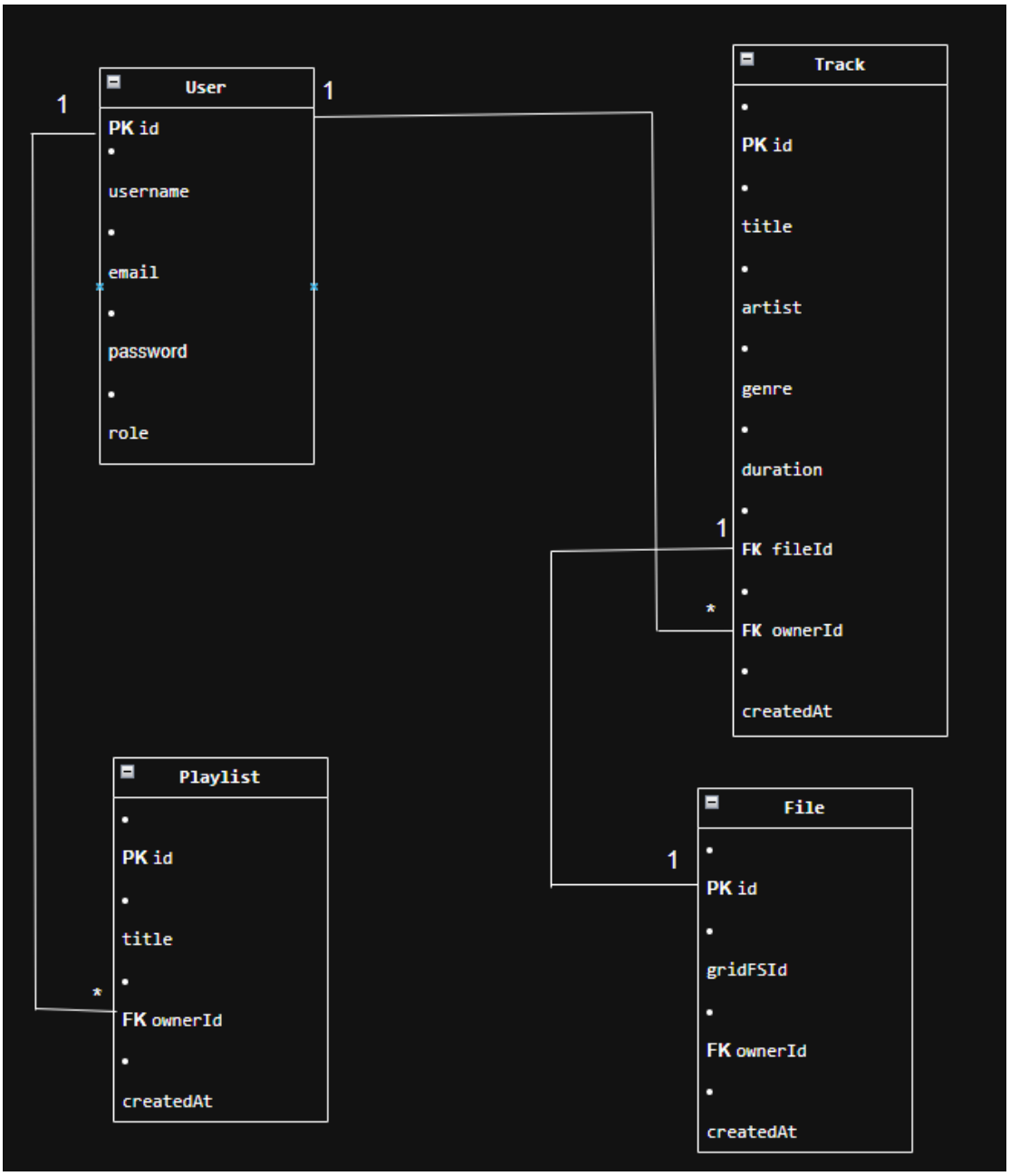

Рис. 3. Схема даних, реалізована в draw.io

Для взаємодії з цією базою даних було реалізовано патерн Repository. Він ізолює бізнес-логіку від деталей роботи з БД, надаючи зручні інтерфейси для маніпуляції даними (пошук, збереження, видалення) без необхідності написання прямих запитів до бази.

3) Проєктування діаграми класів реалізованої системи

Фінальна діаграма класів відображає повну архітектуру розробленої частини системи веб-сервісу «Онлайн-радіостанція». Вона об'єднує класи предметної області, моделі даних, репозиторії та застосовані патерни проєктування, забезпечуючи надійну структуру застосунку.

Основні елементи фінальної діаграми:

- Сутності предметної області (Entities): UserEntity, TrackEntity, PlaylistEntity, FileEntity. Ці класи містять поля даних, гетери, сетери та конструктори (реалізовані через бібліотеку Lombok).
- Репозиторії (Repositories): UserRepository, TrackRepository, PlaylistRepository. Це інтерфейси, що успадковують MongoRepository і ізолюють логіку доступу до бази даних від бізнес-логіки, надаючи готові методи для пошуку (наприклад, findByUsername або findAllByGenre).

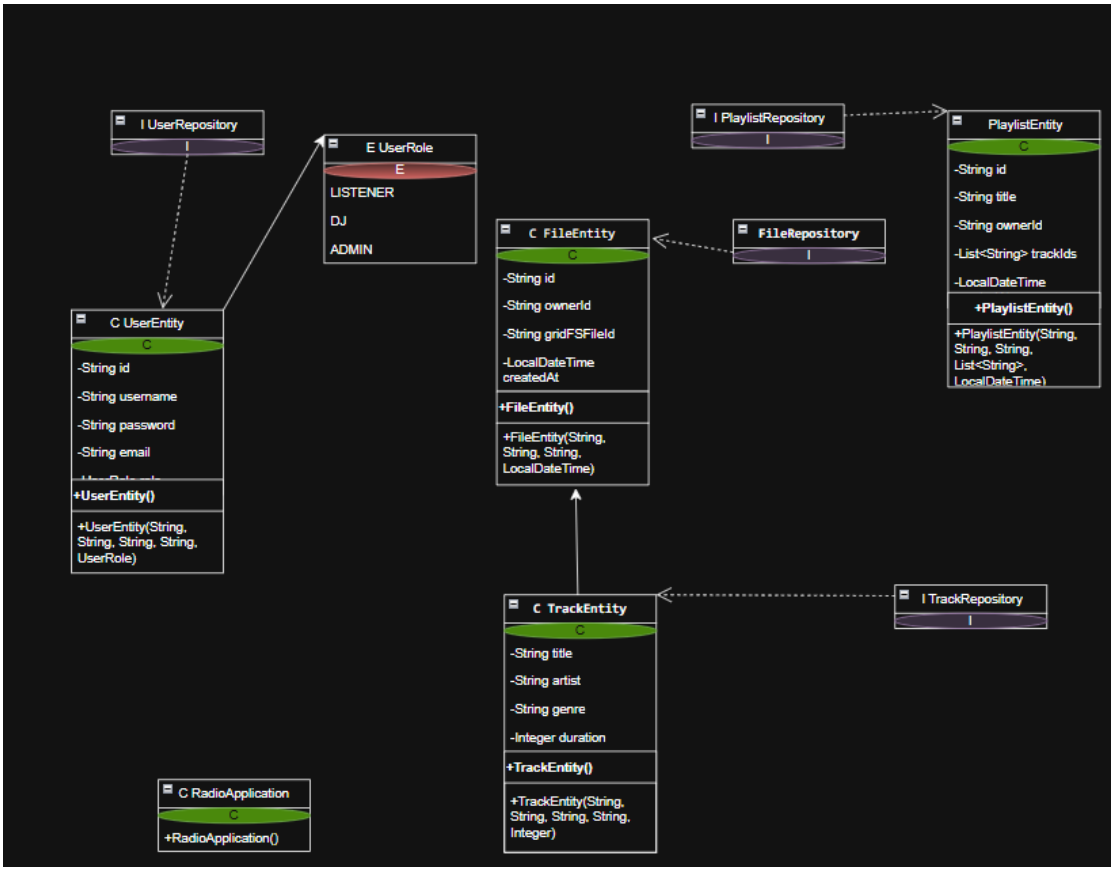

Рис. 4. Діаграма класів реалізованої системи

https://github.com/wildeseed/trpz-1-lab1-2

4) Лістинг коду реалізованих класів системи

Нижче наведено програмний код основних сутностей та репозиторіїв системи, розробленої на мові Java з використанням фреймворку Spring Boot. Код демонструє використання анотацій Lombok для скорочення шаблонного коду та анотацій Spring Data MongoDB для мапінгу класів на колекції бази даних.

```java
package com.example.radio.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import java.time.LocalDateTime;

@Data
@Document("files")
@NoArgsConstructor
@AllArgsConstructor
public abstract class FileEntity {
    @Id
    private String id;
    private String ownerId;
    private String gridFSFileId;
    @CreatedDate
    private LocalDateTime createdAt;
}
```

```java
package com.example.radio.model;

import lombok.*;
import org.springframework.data.annotation.TypeAlias;
import org.springframework.data.mongodb.core.mapping.Document;

@Data
@Builder
@Document("tracks")
@NoArgsConstructor
@AllArgsConstructor
@TypeAlias("track")
@EqualsAndHashCode(callSuper = true)
public class TrackEntity extends FileEntity {
    private String title;
    private String artist;
    private String genre;
    private Integer duration;
}
```

```java
package com.example.radio.model;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import org.springframework.data.annotation.CreatedDate;
import java.time.LocalDateTime;
import java.util.List;

@Data
@Builder
@Document("playlists")
@NoArgsConstructor
@AllArgsConstructor
public class PlaylistEntity {
    @Id
    private String id;
    private String title;
    private String ownerId;
    private List<String> trackIds;
    @CreatedDate
    private LocalDateTime createdAt;
}
```

```java
package com.example.radio.model;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document

@Data
@Builder
@Document("users")
@NoArgsConstructor
@AllArgsConstructor
public class UserEntity {
    @Id
    private String id;
    private String username;
    private String password;
    private String email;
    private UserRole role;
}

enum UserRole {
    LISTENER, DJ, ADMIN
}
```

```java
package com.example.radio.repository;

import com.example.radio.model.TrackEntity;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;
import java.util.List;

@Repository
public interface TrackRepository extends MongoRepository<TrackEntity, String> {
    List<TrackEntity> findAllByGenre(String genre);
    List<TrackEntity> findAllByArtist(String artist);
    List<TrackEntity> findAllByOwnerId(String ownerId);
}
```

```java
package com.example.radio.repository;

import com.example.radio.model.PlaylistEntity;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;
import java.util.List;

@Repository
public interface PlaylistRepository extends MongoRepository<PlaylistEntity, String>
    List<PlaylistEntity> findAllByOwnerId(String ownerId);
}
```

```java
package com.example.radio.repository;

import com.example.radio.model.UserEntity;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;
import java.util.Optional;

@Repository
public interface UserRepository extends MongoRepository<UserEntity, String> {
    Optional<UserEntity> findByUsername(String username);
    boolean existsByEmail(String email);
}
```

Висновок

У ході виконання даної лабораторної роботи були успішно опановані та застосовані на практиці ключові інструменти та методології об'єктно-орієнтованого аналізу та проєктування (ООАП) на прикладі розробки архітектури веб-сервісу «Онлайн-радіостанція».

На початковому етапі було проведено детальний аналіз предметної області та побудовано діаграму варіантів використання (Use Case Diagram). Це дозволило чітко визначити ролі в системі (Гість, Слухач, DJ, Адмін) та функціонал, який їм доступний.

На основі функціональних вимог було розроблено модель предметної області та діаграму класів. Було виділено ключові сутності (User, Track, Playlist) та встановлено зв'язки між ними. Особливу увагу було приділено проєктуванню схеми бази даних (ER-діаграма), де для зберігання медіа-контенту було обрано документо-орієнтований підхід (MongoDB) з використанням механізму GridFS.

Фінальним етапом стала розробка діаграми класів реалізованої системи та написання програмного коду мовою Java з використанням фреймворку Spring Boot. Для взаємодії з даними було успішно реалізовано патерн Repository, що забезпечило гнучкість архітектури та ізоляцію бізнес-логіки від деталей зберігання даних.

Результатом роботи є спроєктована та частково реалізована архітектура веб-сервісу, яка готова до подальшого масштабування та додавання нових функцій.

.