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

«Основи проектування»

Виконав:
студент групи ІА–32
Лось Ярослав

Київ 2025

Мета: Обрати зручну систему побудови UML-діаграм та навчитися будувати діаграми варіантів використання для системи що проєктується, розробляти сценарії варіантів використання та будувати діаграми класів предметної області.

Зміст

Теоретичні відомості

UML (Unified Modeling Language) - це універсальна мова візуального моделювання для специфікації, візуалізації, проєктування та документування програмних систем і бізнес-процесів. Модель системи розглядається на різних рівнях абстракції: концептуальному (початкова модель), логічному (структура та поведінка) і фізичному (реалізація). Діаграма є графічним поданням елементів моделі, що фіксує вимоги та архітектуру системи.

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ (Use-Case)

Призначення: визначити межі системи та функціональні вимоги у вигляді взаємодій користувачів із системою. Це концептуальна модель, яка не описує внутрішній устрій.

Основні елементи:

- Актор (Actor) — зовнішній користувач або система, що взаємодіє із моделлю.
- Варіант використання (Use case) - послуга або поведінка, яку система надає актору; позначається еліпсом із назвою.
- Межа системи - прямокутник, що окреслює, які функції належать системі.
- Відношення:

    * Асоціація - зв'язок між актором і варіантом використання (суцільна лінія).

    * Узагальнення (Generalization) - спадкування між акторами або між варіантами (стрілка з порожнім трикутником до предка).

    * «include» - вказує, що базовий варіант завжди включає поведінку іншого, виокремленого, варіанта використання.

    * «extend» - вказує, що за певних умов до базового варіанта можуть додаватися кроки з іншого (опціональна поведінка).

ДІАГРАМА КЛАСІВ (Class)

Призначення: описати логічну статичну структуру системи: класи, їхні атрибути, операції та відношення між ними.

Основні елементи:

- Клас - зображається прямокутником із трьома секціями: назва; атрибути (із видимістю); операції.

Видимість позначається символами: + (public), - (private), # (protected).

- Асоціація - відношення "користується/посилається на", підтримує множинність (напр., 1, 0..1, 0..*, 1..*).

- Агрегація - відношення «has-a» (слабкий склад), де частини можуть існувати окремо від цілого.

- Композиція - відношення «owns-a» (сильний склад), де життєвий цикл частини залежить від цілого.

- Узагальнення - спадкування атрибутів та операцій, що забезпечує повторне використання й поліморфізм.

БАЗИ ДАНИХ: логічна модель і нормалізація

Логічна модель бази даних описує її структуру у вигляді таблиць, ключів, зв'язків та індексів, незалежно від конкретної СУБД.

Ключові поняття:

- Первинний ключ (Primary Key, PK) - унікальний ідентифікатор кожного рядка в таблиці.

- Зовнішній ключ (Foreign Key, FK) - поле в таблиці, що посилається на первинний ключ в іншій таблиці, забезпечуючи цілісність посилань.

- Індекси - структури, що прискорюють пошук та сортування даних.

Нормалізація — це процес організації таблиць для зменшення логічної надмірності даних та уникнення аномалій при їх оновленні. Основні нормальні форми:

* 1НФ (Перша нормальна форма): Усі атрибути є атомарними, відсутні повторювані групи.

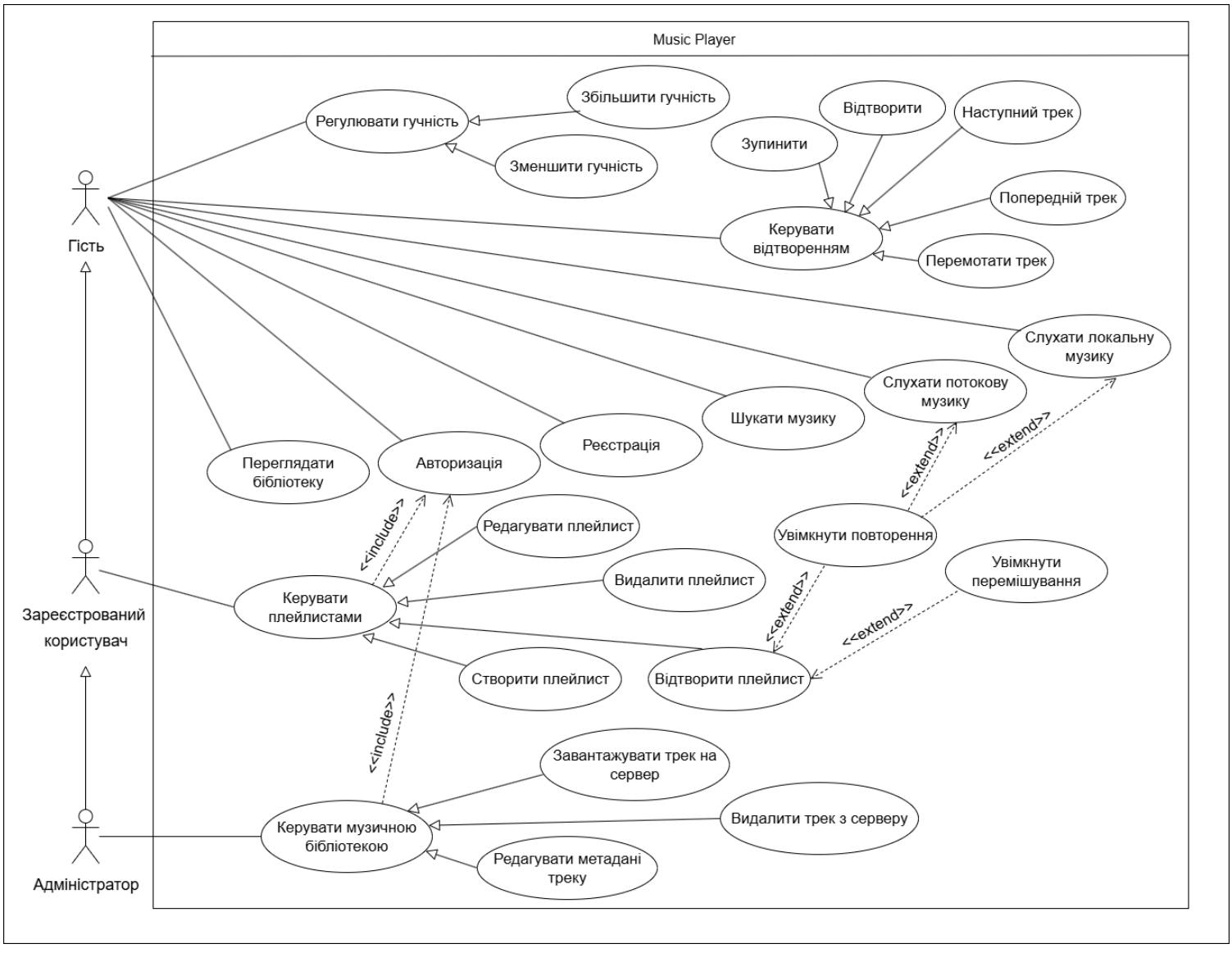
* 2НФ (Друга нормальна форма): Таблиця знаходиться в 1НФ, і всі неключові атрибути повністю функціонально залежать від усього складеного первинного ключа.

* 3НФ (Третя нормальна форма): Таблиця знаходиться в 2НФ, і в ній відсутні транзитивні залежності (коли неключовий атрибут залежить від іншого неключового атрибута).

Хід роботи

1. **Музичний програвач** (iterator, command, memento, facade, visitor, client-server)

1. Побудуємо Use-case діаграму



Діаграма використання для Music Player описує взаємодію користувачів із музичним програвачем та його основні функції.

Актори:

Гість – взаємодіє з системою через базовий функціонал.

Зареєстрований користувач – має розширений функціонал, прив’язаний до облікового запису.

Адміністратор – має доступ до функціоналу управління та налаштування системи.

Основні варіанти використання:

Програвання музичних файлів або відтворення потокової музики.

Можливість створення, запам’ятовування і редагування списків програвання.

Перемішування/повторення, розпізнавання різних аудіо форматів, еквалайзер.

Керування доступними музичними файлами на сервері.

2. Оберемо 3 варіанти використання та напишемо за ними сценарії використання

UC-01: Авторизація користувача

Актор: Гість.

Мета: Увійти до системи для доступу до персоналізованих функцій (плейлисти, історія прослуховувань).

Передумови: Користувач має існуючий обліковий запис.

Тригер: Користувач натискає кнопку "Увійти" та переходить на сторінку авторизації.

Основний сценарій:

1. Система відображає форму входу з полями "Ім'я користувача" та "Пароль".
2. Користувач вводить свої облікові дані.
3. Користувач натискає кнопку "Підтвердити".

4. Система валідує дані: перевіряє існування користувача та відповідність пароля.

5. Валідація успішна. Система створює авторизовану сесію.

6. Система перенаправляє користувача на головну сторінку, відображаючи його персоналізований контент.

Альтернативи / винятки:

4a. Облікові дані невірні. Система виводить повідомлення "Неправильне ім'я користувача або пароль". Користувач залишається на сторінці входу.

Постумови: Користувач успішно увійшов до системи та ідентифікований як "Зареєстрований користувач".

UC-02: Створити та наповнити плейлист

Актор: Зареєстрований користувач.

Мета: Створити новий персональний список відтворення та додати до нього треки з бібліотеки.

Передумови: Користувач авторизований. Бібліотека сервера містить треки для додавання.

Тригер: Користувач натискає кнопку "Створити плейлист".

Основний сценарій:

1. Система запитує назву нового плейлиста.

2. Користувач вводить назву (напр., "Вечірній джаз") і підтверджує.

3. Система створює порожній плейлист та відображає його у списку плейлистів користувача.

4. Користувач переглядає загальну бібліотеку треків.

5. Навпроти бажаного треку користувач натискає опцію "Додати до плейлиста".

6. Система відображає список плейлистів користувача.

7. Користувач обирає щойно створений плейлист "Вечірній джаз".

8. Система додає трек до обраного плейлиста.

Альтернативи / винятки:

2а. Користувач вводить порожню назву. Система виводить помилку "Назва не може бути порожньою".

8а. Обраний трек вже існує в цьому плейлисті. Система виводить інформаційне повідомлення "Цей трек вже є у плейлисті".

Постумови: Створено новий плейлист, що містить один або кілька обраних треків.

UC-03: Завантажити трек на сервер

Актор: Адміністратор.

Мета: Розширити музичну бібліотеку, додавши новий аудіофайл.

Передумови: Адміністратор авторизований та має відповідні права доступу.

Тригер: Адміністратор переходить до панелі адміністрування та натискає "Завантажити трек".

Основний сценарій:

1. Система відображає форму для завантаження файлу та введення метаданих (назва, виконавець).
2. Адміністратор обирає аудіофайл зі свого комп'ютера.
3. Адміністратор заповнює обов'язкові поля метаданих.
4. Адміністратор натискає кнопку "Завантажити".
5. Система валідує формат файлу та повноту метаданих.
6. Система завантажує файл на сервер у визначене сховище.
7. Система зберігає метадані та шлях до файлу в базу даних.
8. Система відображає повідомлення про успішне додавання треку.

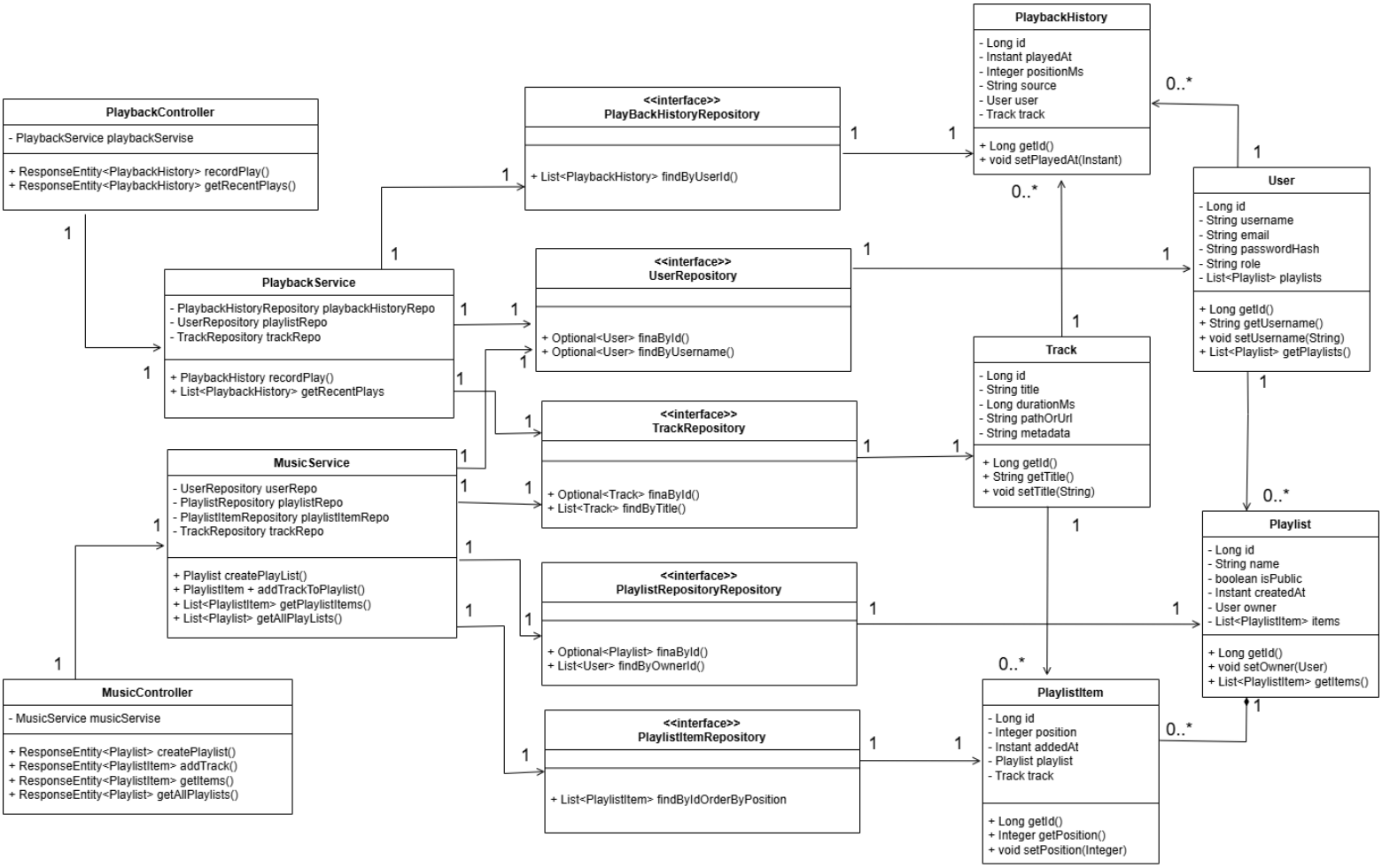
Альтернативи / винятки:

5а. Формат файлу не підтримується. Система виводить помилку "Непідтримуваний формат файлу". Завантаження скасовується.

5б. Не заповнені обов'язкові метадані. Система підсвічує порожні поля та виводить помилку "Заповніть обов'язкові поля".

Постумови: Новий трек фізично знаходиться на сервері та доступний для всіх користувачів через бібліотеку.

3. Зобразимо діаграму класів



MusicController

Контролер для роботи з користувачами, плейлистами та треками. Приймає HTTP-запити, викликає методи MusicService, повертає відповіді.

PlaybackController

Контролер для відтворення треків і перегляду історії прослуховувань.

MusicService

Сервіс бізнес-логіки для керування музичним контентом. Виконує операції над сутностями користувачів, плейлистів і треків через відповідні репозиторії.

PlaybackService

Сервіс для обробки операцій відтворення. Додає записи до історії прослуховувань, керує отриманням треків для користувача.

User

Сутність користувача. Містить інформацію про логін, email, пароль, роль.

Playlist

Сутність музичного плейлиста. Містить назву, дату створення, публічність, власника і список елементів.

PlaylistItem

Сутність елемента плейлиста. Зберігає позицію, дату додавання, пов’язаний Track та Playlist.

Track

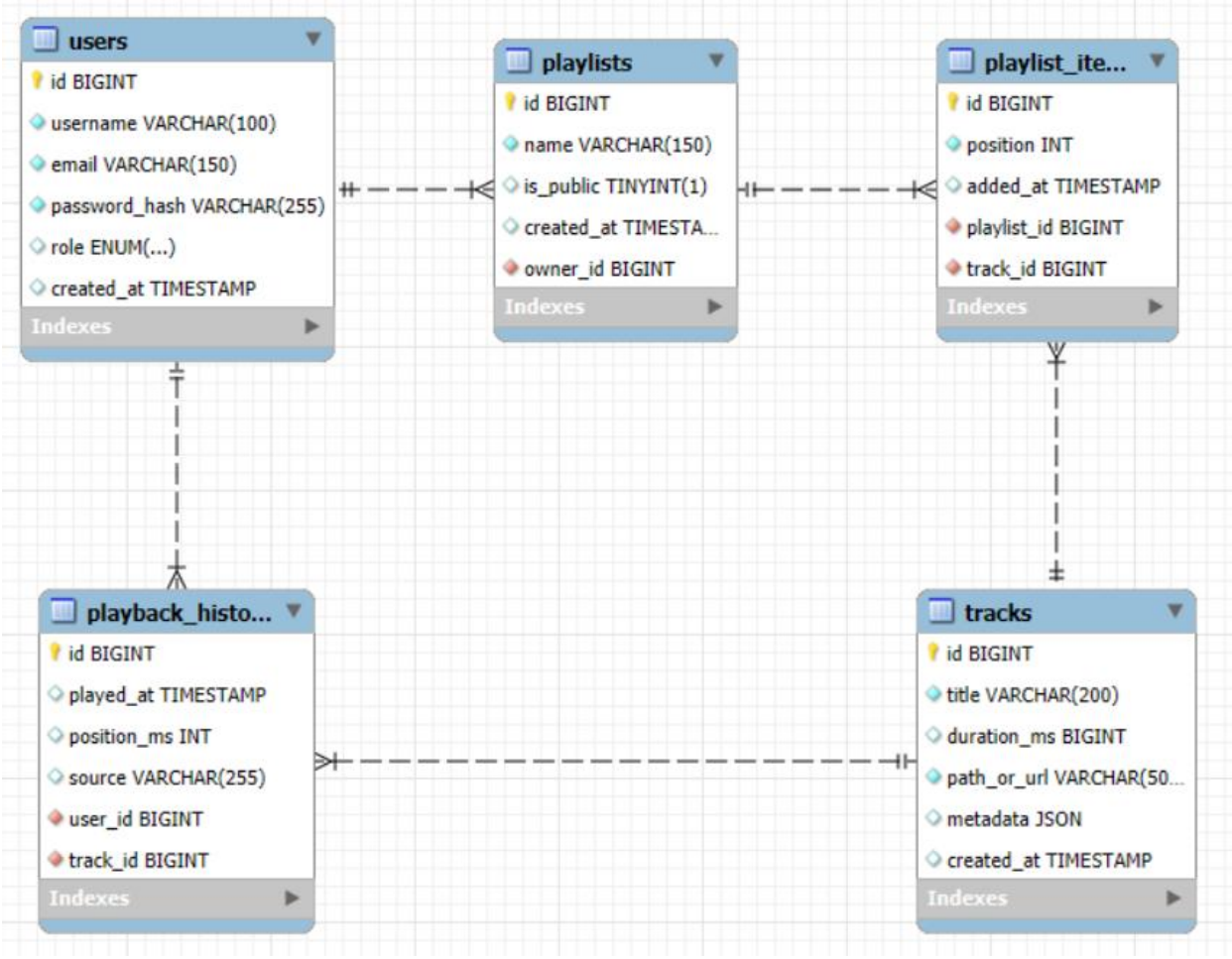
Сутність музичного треку. Містить назву, тривалість, шлях до файлу або URL, метадані.

PlaybackHistory

Сутність історії відтворень. Зберігає інформацію про час відтворення, трек і користувача.

PlaylistRepository, PlaylistItemRepository, TrackRepository, UserRepository, PlaybackHistoryRepository – інтерфейси, що реалізують патерн Repository. Призначені для роботи з даними (CRUD-операції) для конкретної сутності, приховуючи деталі взаємодії з базою даних.

4. Спроєктуємо БД



Модель бази даних спроєктована для забезпечення надійного зберігання всієї інформації, необхідної для функціонування музичного веб-програвача. Вона враховує потреби зберігання даних про користувачів, музичну бібліотеку, персональні плейлисти та історію прослуховувань.

Вона складається з кількох взаємопов'язаних таблиць:

users - Центральна таблиця, що ідентифікує унікального користувача системи. Кожен запис у цій таблиці представляє один обліковий запис, до якого прив'язані всі персоналізовані дані, такі як плейлисти та історія. Містить поля для автентифікації (username, password_hash) та розмежування прав (role).

tracks - Довідкова таблиця, що зберігає метадані про кожен музичний трек, завантажений на сервер. Містить такі поля, як назва (title), виконавець (artist), тривалість (duration_ms) та шлях до файлу на сервері (path_or_url).

playlists - Зберігає інформацію про персональні плейлисти, створені користувачами. Кожен запис містить назву плейлиста (name) та посилання на його власника (owner_id).

playlist_items - Асоціативна таблиця, що реалізує зв'язок "багато-до-багатьох" між плейлистами та треками. Вона дозволяє одному треку бути в кількох плейлистах, а одному плейлисту містити багато треків. Ключове поле position забезпечує збереження порядку треків у кожному плейлисті.

playback_history - Зберігає історію прослуховувань треків користувачами. Кожен запис фіксує, який користувач (user_id) прослухав який трек (track_id) та коли це відбулося (played_at).

Ключові зв'язки:

users - playlists (1:N, Один-до-Багатьох): Один користувач може створити багато плейлистів. Зв'язок реалізовано через зовнішній ключ owner_id у таблиці playlists.

playlists - playlist_items (1:N) та tracks - playlist_items (1:N): Ці два зв'язки разом утворюють відношення "багато-до-багатьох". Вони дозволяють гнучко наповнювати плейлисти треками.

users - playback_history (1:N): Один користувач може мати багато записів в історії прослуховувань.

tracks - playback_history (1:N): Один трек може бути прослуханий багато разів різними користувачами.

База даних спроєктована відповідно до принципів нормалізації (3НФ), що дозволяє уникнути надмірності даних, запобігти аномаліям при їх оновленні та забезпечити логічну узгодженість інформації.

Контрольні запитання

1. Що таке UML?

UML (Unified Modeling Language) - це уніфікована мова візуального моделювання, яка використовується для специфікації, проєктування, документування та візуалізації програмних систем і бізнес-процесів.

2. Що собою становить діаграма класів UML?

Діаграма класів - це статичне подання системи, яке показує класи, їхні атрибути та операції, а також відношення між класами (асоціації, узагальнення, агрегації, композиції).

3. Які діаграми UML називають канонічними?

До канонічних UML-діаграм належать: діаграма варіантів використання, класів, послідовності, кооперації, станів, діяльності, компонентів і розгортання.

4. Що собою становить діаграма варіантів використання?

Діаграма варіантів використання (Use Case) — це концептуальна модель, що показує функціональні вимоги до системи через взаємодію акторів (користувачів чи зовнішніх систем) із варіантами використання (сценаріями роботи).

5. Чим відрізняються зв'язок типу агрегації від зв'язків композиції на діаграмі класів?

Агрегація («has-a») - слабкий зв'язок «ціле–частина»: частина може існувати окремо від цілого. Композиція («owns-a») - сильний зв'язок: життєвий цикл частини залежить від цілого, при знищенні цілого знищуються всі його частини.

6. Що собою становлять нормальні форми баз даних?

Нормальні форми — це правила організації таблиць БД, що усувають надмірність і аномалії при оновленні даних. Основні: 1НФ - атрибути атомарні; 2НФ - усі неключові атрибути залежать від усього ключа; 3НФ - відсутні транзитивні залежності від ключа; BCNF - посилена 3НФ.

7. Що таке фізична модель баз даних? Логічна?

Логічна модель бази даних - це абстрактний опис структури даних, який включає таблиці, поля, ключі, зв'язки, індекси та обмеження цілісності. Вона показує, які сутності існують у системі, як вони пов'язані між собою, але не залежить від

конкретної СУБД. Фізична модель бази даних - це конкретна реалізація логічної моделі на рівні обраної СУБД. Вона описує зберігання даних на носіях, використання файлів і сторінок, структуру індексів, механізми доступу та оптимізацію продуктивності.

8. Який взаємозв'язок між таблицями БД та програмними класами?

Таблиці бази даних і програмні класи відображають одні й ті самі сутності предметної області, але на різних рівнях: Класи описують логіку та поведінку об'єктів у програмі (атрибути, методи). Таблиці зберігають ці ж об'єкти у вигляді рядків і стовпців. Між ними встановлюється відображення (mapping): «Один клас — одна таблиця» (найпоширеніший варіант). «Один клас — кілька таблиць» (якщо дані рознесені). «Кілька класів — одна таблиця» (якщо реалізується спадкування або спільне зберігання).

Висновки

У ході виконання лабораторної роботи я провів проєктування програмної системи «Музичний програвач» з використанням мови візуального моделювання UML. Протягом роботи я розробив діаграму варіантів використання (Use Case), яка дозволила чітко визначити функціональні вимоги до системи, її межі, а також основні ролі користувачів та їхні можливі дії; також написав детальні сценарії для трьох ключових варіантів використання, що формалізували основні та альтернативні потоки подій, усунувши неоднозначності у вимогах; спроєктував діаграму класів предметної області, яка змоделювала основні бізнес-сутності та логічні зв'язки між ними; на основі предметної області розробив діаграму класів а також створив логічну модель бази даних з дотриманням принципів нормалізації (3НФ)

Додаток А

Вихідні коди класів системи

```
public class User {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable=false, length=100)
    private String username;

    @Column
    private String email;

    @Column(name="password_hash")
    private String passwordHash;

    @Column(nullable=false)
    private String role;

    @OneToMany(mappedBy = "owner", cascade = CascadeType.ALL,
orphanRemoval = true)
    private List<Playlist> playlists;
}

public class Track {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable=false)
    private String title;
```

```
    @Column(name = "duration_ms")
    private Long durationMs;

    @Column(name = "path_or_url", columnDefinition = "TEXT")
    private String pathOrUrl;

    @Column(columnDefinition = "JSON")
    private String metadata;
}

public class Playlist {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable=false)
    private String name;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "owner_id", nullable = false)
    private User owner;

    @Column(name = "is_public")
    private boolean isPublic = false;

    @Column(name = "created_at")
    private Instant createdAt = Instant.now();

    @OneToMany(mappedBy = "playlist", cascade = CascadeType.ALL,
orphanRemoval = true)
    @OrderBy("position ASC")
```

```
    private List<PlaylistItem> items = new ArrayList<>();
}

public class PlaylistItem {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "playlist_id", nullable = false)
    private Playlist playlist;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "track_id", nullable = false)
    private Track track;

    @Column(nullable = false)
    private Integer position;

    @Column(name = "added_at")
    private Instant addedAt = Instant.now();
}

public class PlaybackHistory {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id")
    private User user;

    @ManyToOne(fetch = FetchType.LAZY)
```

```
    @JoinColumn(name = "track_id")
    private Track track;

    @Column(name = "played_at")
    private Instant playedAt = Instant.now();

    @Column(name = "position_ms")
    private Integer positionMs;

    @Column(length = 50)
    private String source;
}

public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);
}

public interface TrackRepository extends JpaRepository<Track, Long> {
    List<Track> findByTitleContainingIgnoreCase(String q);
}

public interface PlaylistRepository extends JpaRepository<Playlist, Long> {
    List<Playlist> findByOwnerId(Long ownerId);
}

public interface PlaylistItemRepository extends JpaRepository<PlaylistItem,
Long> {
    List<PlaylistItem> findByPlaylistIdOrderByPosition(Long playlistId);
}
```

```
public interface PlaybackHistoryRepository extends
JpaRepository<PlaybackHistory, Long> {
    List<PlaybackHistory> findTop100ByUserIdOrderByPlayedAtDesc(Long userId);
}
```