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

Лабораторна робота № 2

із дисципліни «Технології розроблення програмного забезпечення»

Тема: «Основи проектування»

Варіант - 21

Виконав
Студент групи ІА-31:
Козир Я. О.

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

1. **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.

2. **Теоретичні відомості:**

Мова UML є загальноцільовою мовою візуального моделювання, яка розроблена для специфікації, візуалізації, проєктування та документування компонентів програмного забезпечення, бізнес-процесів та інших систем. Мова UML є досить строгим та потужним засобом моделювання, який може бути ефективно використаний для побудови концептуальних, логічних та графічних 18 моделей складних систем різного цільового призначення. Ця мова увібрала в себе найкращі якості та досвід методів програмної інженерії, які з успіхом використовувалися протягом останніх років при моделюванні великих та складних систем.

Діаграма – це графічне уявлення сукупності елементів моделі у формі зв'язкового графа, вершинам і ребрам (дугам) якого приписується певна семантика. Нотація канонічних діаграм є основним засобом розробки моделей мовою UML.

У нотації мови UML визначено такі види діаграм:

- варіантів використання (use case diagram);
- класів (class diagram);
- кооперації (collaboration diagram);
- послідовності (sequence diagram);
- станів (statechart diagram);
- діяльності (activity diagram);
- компонентів (component diagram);
- розгортання (deployment diagram).

Діаграма варіантів використання (Use-Cases Diagram) – це UML діаграма за допомогою якої у графічному вигляді можна зобразити вимоги до системи, що розробляється. Діаграма варіантів використання – це вихідна концептуальна модель проєктованої системи, вона не описує внутрішню побудову системи. Діаграма використання складається з:

- Акторів - будь-які об'єкти, суб'єкти чи системи, що взаємодіють з модельованою бізнес-системою ззовні для досягнення своїх цілей або вирішення певних завдань.
- Варіантів використання - служать для опису служб, які система надає актору.
- Відношень: асоціація, узагальнення, залежність, включення, розширення.

Діаграми класів використовуються при моделюванні програмних систем найчастіше. Вони є однією із форм статичного опису системи з погляду її проєктування, показуючи її структуру. Діаграма класів не відображає динамічної поведінки об'єктів зображених на ній класів. На діаграмах класів показуються класи, інтерфейси та відносини між ними.

Діаграма класів містить у собі класи, їхні методи та атрибути, зв’язки. Методи та атрибути мають 4 модифікатори доступу: public, package, protected, private. Зв’язки налічують у собі асоціацію, агрегацію, композицію, успадкування тощо.

3. **Завдання:**
   - Ознайомитись з короткими теоретичними відомостями.
   - Проаналізувати тему та спроєктувати діаграму варіантів використання відповідно до обраної теми лабораторного циклу.
   - Спроєктувати діаграму класів предметної області.
   - Вибрати 3 варіанти використання та написати за ними сценарії використання.
   - На основі спроєктованої діаграми класів предметної області розробити основні класи та структуру бази даних системи. Класи даних повинні реалізувати шаблон Repository для взаємодії з базою даних.
   - Нарисувати діаграму класів для реалізованої частини системи.
   - Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму варіантів використання відповідно, діаграму
   - класів системи, вихідні коди класів системи, а також зображення структури бази даних

4. **Хід роботи:**

Тема : Online radio station

Діаграма варіантів використання

Актори:

- Користувач
- Слухач
- Діджей
- Адміністратор

Варіанти використання:

- Реєстрація
- Авторизація
- Перегляд станцій
- Прослуховування радіо
- Додавання/видалення радіо-станції в збережене
- Додавання відгуку про конкретний трек/передачу
- Переглянути відгуки про конкретний трек/передачу
- Поставити вподобайку/дізлайк на трек/передачу

- Керувати чергою відтворення
- Почати/завершити стрім
- Створення/видалення/зміна користувачів
- Створення/видалення/зміна станцій
- Створення/видалення/зміна треків

Зв’язки:

- Користувач - Реєстрація
- Користувач - Авторизація
- Користувач - Перегляд станцій
- Слухач - Прослуховування радіо
- Слухач - Додавання/видалення радіо-станції в збережене
- Слухач - Додавання відгуку про конкретний трек/передачу
- Слухач - Переглянути відгуки про конкретний трек/передачу
- Слухач - Поставити вподобайку/дізлайк на трек/передачу
- Діджей - Почати/завершити стрім
- Діджей - Створення/видалення/зміна треків
- Адміністратор - Створення/видалення/зміна користувачів
- Адміністратор - Створення/видалення/зміна станцій
- Адміністратор - Створення/видалення/зміна треків
- Прослуховування радіо - (include) - Перегляд станцій
- Додавання відгуку про конкретний трек/передачу - (include) - Прослуховування радіо
- Переглянути відгуки про конкретний трек/передачу - (include) - Прослуховування радіо
- Поставити вподобайку/дізлайк на трек/передачу - (include) - Прослуховування радіо
- Керувати чергою відтворення - (include) - Почати/завершити стрім
- Додавання/видалення радіо-станції в збережене - (include) - Перегляд станцій
- Створення/видалення/зміна станцій - (include) - Перегляд станцій
- Створення/видалення/зміна треків - (include) - Перегляд треків

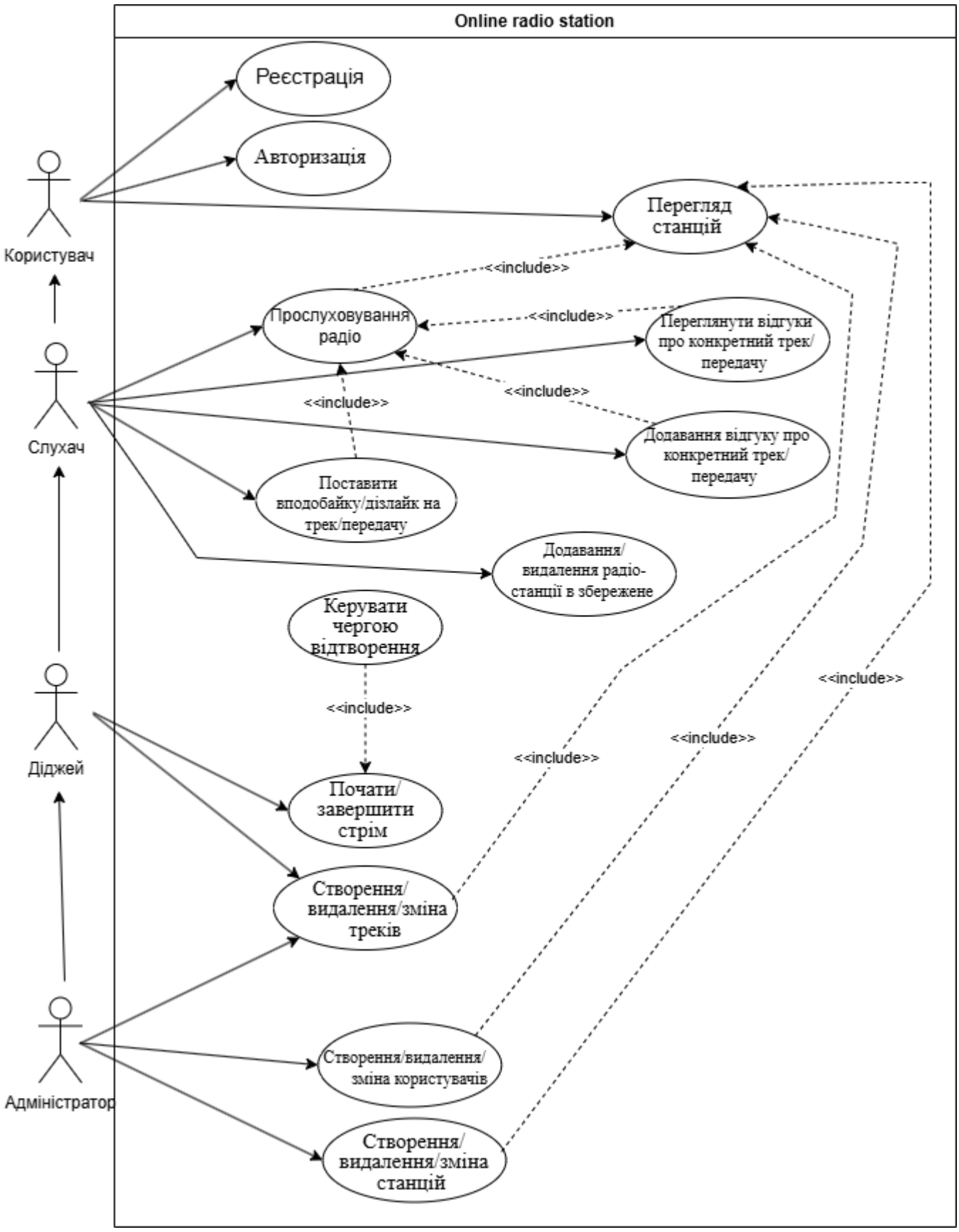

Рис. 1 - Діаграма використання

Сценарії використання:

Прослуховування радіо	
Передумови	Слухач авторизований у системі, радіостанція активна
Постумови	Слухач слухає обрану радіостанцію.
Взаємодіючі сторони	Слухач, Система онлайн-радіостанції.
Короткий опис	Цей варіант використання визначає процес прослуховування радіостанції.
Основний потік подій	Цей варіант використання запускається, коли слухач хоче прослухати радіо.
	1. Слухач переглядає доступні станції.
	2. Слухач обирає станцію.
	3. Система відтворює ефір станції.
Винятки	Станція недоступна. Система повідомляє про помилку, і слухач повертається до перегляду станцій.
Примітки	-

Керувати чергою відтворення	
Передумови	Діджей авторизований у системі, бібліотека треків заповнена
Постумови	Черга відтворення оновлена для стріму.
Взаємодіючі сторони	Діджей, Система онлайн-радіостанції.
Короткий опис	Цей варіант використання визначає процес управління чергою відтворення треків.
Основний потік подій	Цей варіант використання запускається, коли діджей готує стрім.
	1. Діджей переглядає бібліотеку треків.
	2. Діджей переміщує обрані треки до черги відтворення.
	3. Діджей підтверджує чергу.
Винятки	Бібліотека порожня. Система повідомляє про помилку, і діджей повертається до перегляду.
Примітки	-

Додавання відгуку про конкретний трек/передачу	
Передумови	Слухач прослухав трек/передачу, авторизований у системі
Постумови	Створюється відгук про трек/передачу.
Взаємодіючі сторони	Слухач, Система онлайн-радіостанції.
Короткий опис	Цей варіант використання визначає процес додавання відгуку.
Основний потік подій	Цей варіант використання запускається, коли слухач хоче залишити відгук.
	1. Слухач обирає трек/передачу під час прослуховування.
	2. Слухач натискає кнопку додавання відгуку.
	3. Слухач вводить текст відгуку.
	4. Слухач підтверджує дію.

Винятки	Трек/передача не прослухана. Система повідомляє про неможливість додати відгук.
Примітки	-

Структура БД:

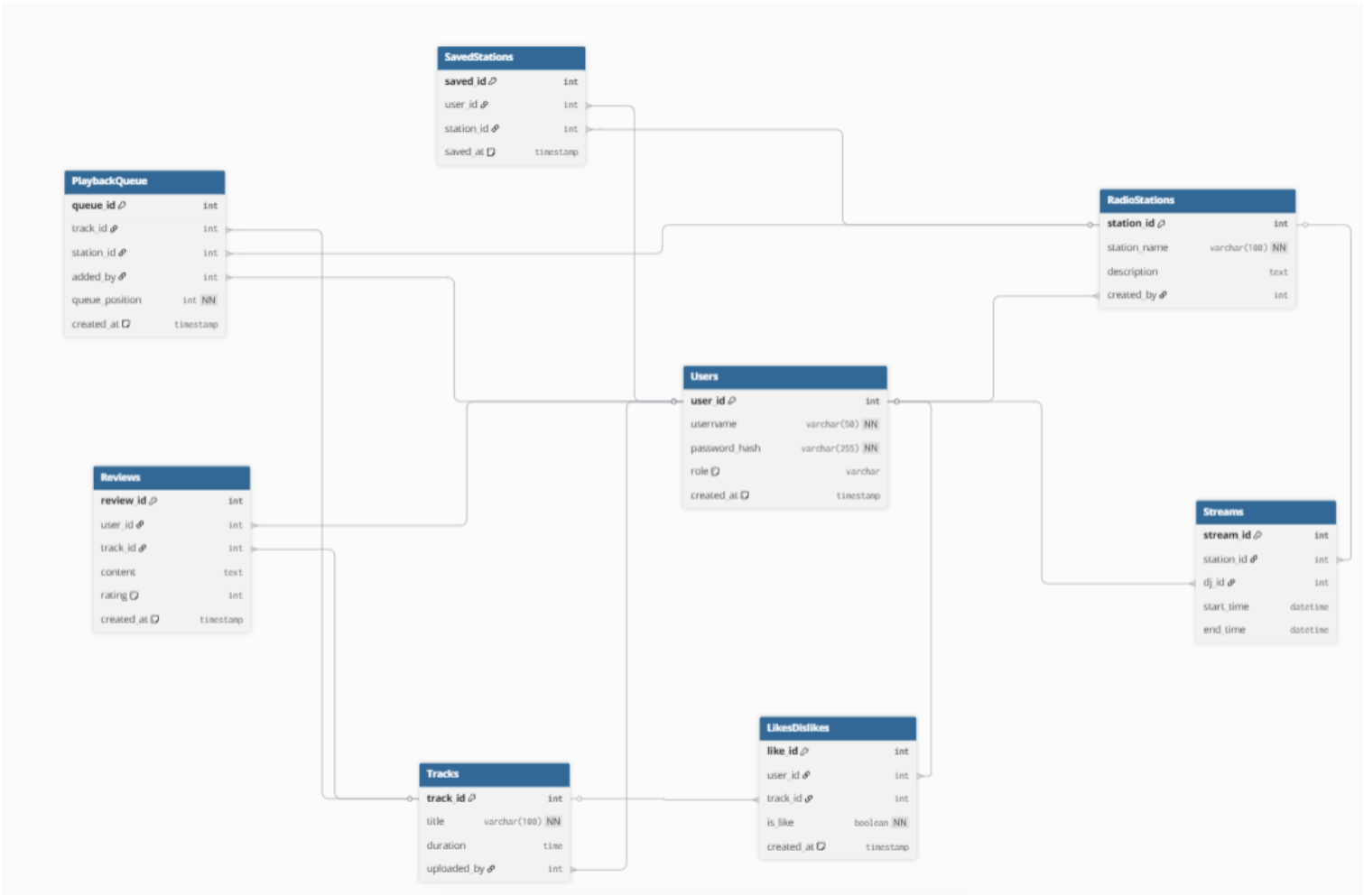

Рис. 2 - Структура БД

Діаграми класів:

Репозиторії:

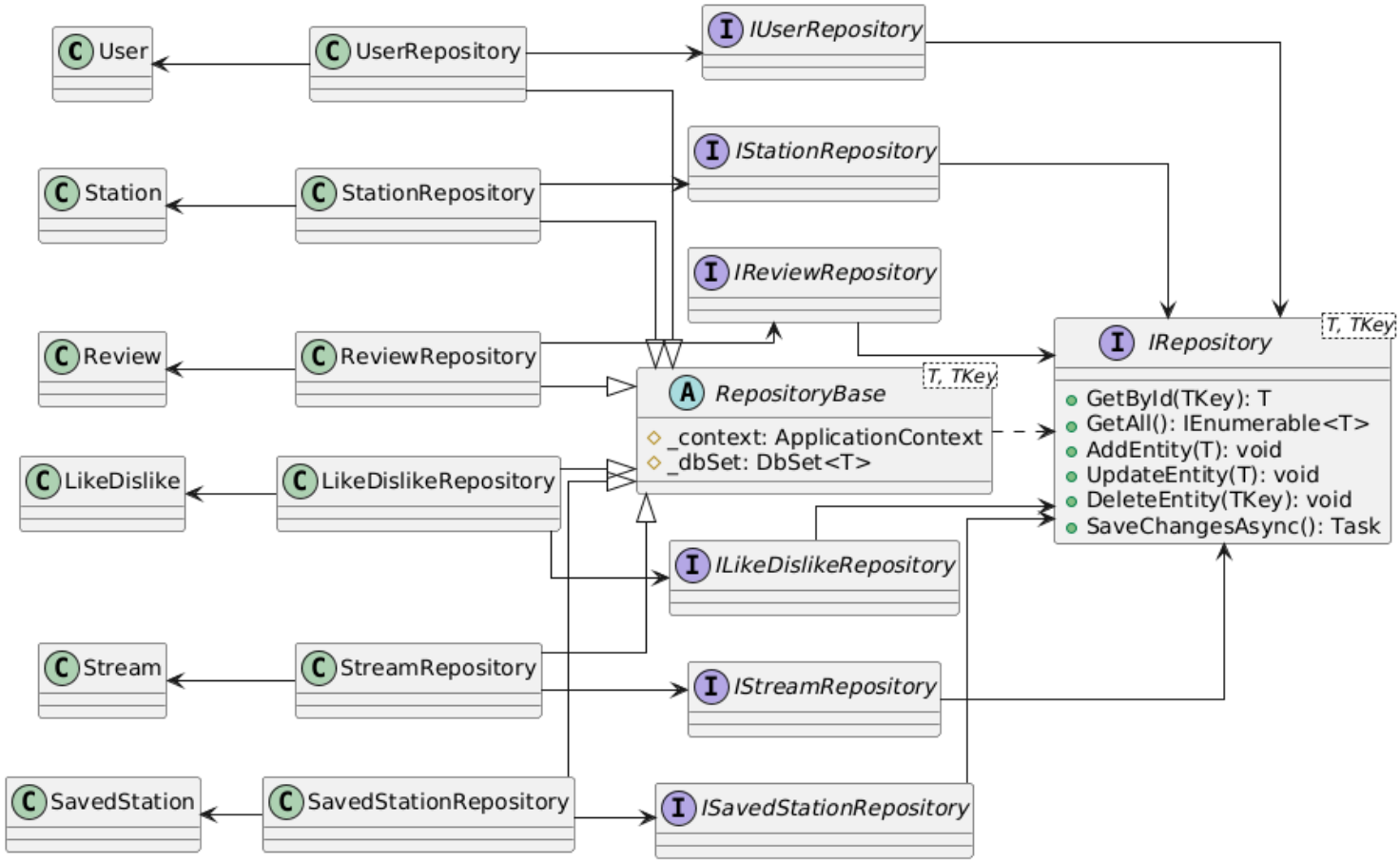

Рис. 3 - Діаграма класів репозиторіїв

1. Узагальнені (Generic) компоненти

- Інтерфейс IRepository<T, TKey>
  - Призначення: Універсальний "контракт", який описує базові операції, що має виконувати будь-який репозиторій.
  - Параметри:
    - T: Тип сутності (наприклад, User або Station).
    - TKey: Тип первинного ключа (наприклад, int або Guid).
  - Методи:
    - GetById(TKey): Повертає об'єкт типу T.
    - GetAll(): Повертає колекцію IEnumerable<T>.
    - AddEntity(T): Додає новий об'єкт. Повертає void.

        - UpdateEntity(T): Оновлює існуючий об'єкт. Повертає void.
        - DeleteEntity(TKey): Видаляє об'єкт за ключем. Повертає void.
        - SaveChangesAsync(): Зберігає зміни в базі даних асинхронно. Повертає Task.
- Абстрактний клас RepositoryBase<T, TKey>
    - Призначення: Спільна реалізація інтерфейсу IRepository. Усі конкретні репозиторії успадковують цей клас, щоб не дублювати код для базових операцій.
    - Атрибути:
        - _context: ApplicationContext: Посилання на контекст бази даних (наприклад, з Entity Framework).
        - _dbSet: DbSet<T>: Посилання на таблицю в базі даних, що відповідає сутності T.
    - Зв'язки:
        - Реалізує інтерфейс IRepository<T, TKey>.

2. Специфічні реалізації для сутностей

- User
    - Інтерфейс: IUserRepository, успадковує IRepository<User, TKey>.
    - Клас: UserRepository, реалізує IUserRepository та успадковує RepositoryBase<User, TKey>. Асоційований із сутністю User.
- Station
    - Інтерфейс: IStationRepository, успадковує IRepository<Station, TKey>.
    - Клас: StationRepository, реалізує IStationRepository та успадковує RepositoryBase<Station, TKey>. Асоційований із сутністю Station.
- Review
    - Інтерфейс: IReviewRepository, успадковує IRepository<Review, TKey>.
    - Клас: ReviewRepository, реалізує IReviewRepository та успадковує RepositoryBase<Review, TKey>. Асоційований із сутністю Review.
- LikeDislike
    - Інтерфейс: ILikeDislikeRepository, успадковує IRepository<LikeDislike, TKey>.

- Клас: LikeDislikeRepository, реалізує ILikeDislikeRepository та успадковує RepositoryBase<LikeDislike, TKey>. Асоційований із сутністю LikeDislike.

- Stream
  - Інтерфейс: IStreamRepository, успадковує IRepository<Stream, TKey>.
  - Клас: StreamRepository, реалізує IStreamRepository та успадковує RepositoryBase<Stream, TKey>. Асоційований із сутністю Stream.
- SavedStation
  - Інтерфейс: ISavedStationRepository, успадковує IRepository<SavedStation, TKey>.
  - Клас: SavedStationRepository, реалізує ISavedStationRepository та успадковує RepositoryBase<SavedStation, TKey>. Асоційований із сутністю SavedStation.

Класи даних:

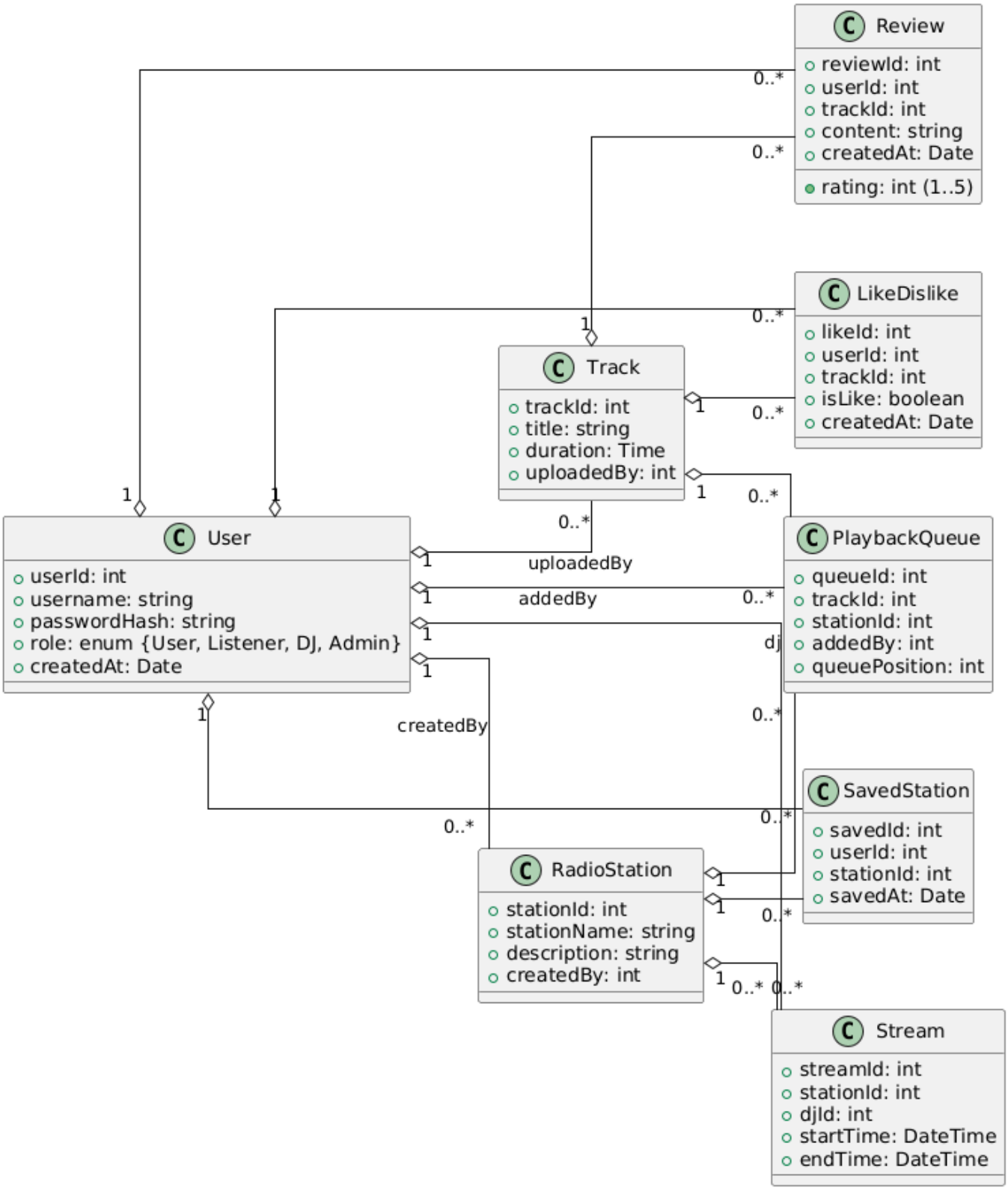

Рис. 4 - Діаграма класів даних

Класи даних (сутності)

- **User**

- **Attributes:**
  - userId: int (PK)
  - username: string
  - passwordHash: string
  - role: enum {User, Listener, DJ, Admin}
  - createdAt: Date
- **Methods:**
  - (Методи не показано на діаграмі)

- **RadioStation**
  - **Attributes:**
    - stationId: int (PK)
    - stationName: string
    - description: string
    - createdBy: int (FK to User)
  - **Methods:**
    - (Методи не показано на діаграмі)
- **Track**
  - **Attributes:**
    - trackId: int (PK)
    - title: string
    - duration: Time
    - uploadedBy: int (FK to User)
  - **Methods:**
    - (Методи не показано на діаграмі)
- **PlaybackQueue**
  - **Attributes:**
    - queueId: int (PK)
    - trackId: int (FK to Track)
    - stationId: int (FK to RadioStation)

        - addedBy: int (FK to User)
        - queuePosition: int
    - **Methods:**
        - (Методи не показано на діаграмі)
- **SavedStation**
    - **Attributes:**
        - savedId: int (PK)
        - userId: int (FK to User)
        - stationId: int (FK to RadioStation)
        - savedAt: Date
    - **Methods:**
        - (Методи не показано на діаграмі)
- **Review**
    - **Attributes:**
        - reviewId: int (PK)
        - userId: int (FK to User)
        - trackId: int (FK to Track)
        - content: string
        - rating: int (1..5)
        - createdAt: Date
    - **Methods:**
        - (Методи не показано на діаграмі)
- **LikeDislike**
    - **Attributes:**
        - likeId: int (PK)
        - userId: int (FK to User)
        - trackId: int (FK to Track)
        - isLike: boolean
        - createdAt: Date

- **Methods:**
  - (Методи не показано на діаграмі)

- **Stream**
  - **Attributes:**
    - streamId: int (PK)
    - stationId: int (FK to RadioStation)
    - djId: int (FK to User)
    - startTime: DateTime
    - endTime: DateTime
  - **Methods:**
    - (Методи не показано на діаграмі)

Зв’язки між класами (Multiplicity)

- User -- (1:N) --> RadioStation (один користувач може створити багато станцій)
- User -- (1:N) --> Track (один користувач може завантажити багато треків)
- User -- (1:N) --> PlaybackQueue (один користувач може додати багато треків у черги)
- User -- (1:N) --> SavedStation (один користувач може зберегти багато станцій)
- User -- (1:N) --> Review (один користувач може залишити багато відгуків)
- User -- (1:N) --> LikeDislike (один користувач може поставити багато лайків/дизлайків)
- User -- (1:N) --> Stream (один користувач (DJ) може провести багато стрімів)
- RadioStation -- (1:N) --> Stream (одна станція може мати багато стрімів)
- RadioStation -- (1:N) --> SavedStation (одну станцію можуть зберегти багато користувачів)
- RadioStation -- (1:N) --> PlaybackQueue (одна станція може мати багато треків у черзі)
- Track -- (1:N) --> LikeDislike (один трек може мати багато лайків/дизлайків)
- Track -- (1:N) --> Review (один трек може мати багато відгуків)
- Track -- (1:N) --> PlaybackQueue (один трек може бути у багатьох чергах)

Вихідний код класів:

```
namespace RadioStation.Data
{
    public interface IRepository<T, TKey> where T : class
    {
        Task<T> GetById(TKey id);
        IQueryable<T> GetAll();
        void AddEntity(T entity);
        void UpdateEntity(T entity);
        void DeleteEntity(TKey id);
        Task SaveChangesAsync();
    }
}
```

Рис. 5 - Код інтерфейсу IRepository<T, TKey>

```
namespace RadioStation.Data
{
    public abstract class RepositoryBase<T, TKey> : IRepository<T, TKey> where T : class
    {
        protected ApplicationContext _context;
        protected DbSet<T> _dbSet;
        public RepositoryBase(ApplicationContext context)
        {
            _context = context;
            _dbSet = context.Set<T>();
        }
        public Task<T> GetById(TKey id)
        {
            return _dbSet.FindAsync(id).AsTask();
        }
        public IQueryable<T> GetAll()
        {
            return _dbSet.AsQueryable();
        }
        public void AddEntity(T entity)
        {
            _dbSet.Add(entity);
        }
        public void UpdateEntity(T entity)
        {
            _dbSet.Update(entity);
        }
        public void DeleteEntity(TKey id)
        {
            var entity = _dbSet.Find(id);
            if (entity != null)
            {
                _dbSet.Remove(entity);
            }
        }
        public async Task SaveChangesAsync()
        {
```

Рис. 6 - Код класу RepositoryBase<T, TKey>

```
namespace RadioStation.Data
{
    public class User
    {
        [Key]
        public Guid UserId { get; set; }
        public string Username { get; set; }
        public string PasswordHash { get; set; }
        public string Role { get; set; }
        public DateTime CreatedAt { get; set; }

        public ICollection<RadioStation> CreatedStations { get; set; }
        public ICollection<Track> UploadedTracks { get; set; }
        public ICollection<PlaybackQueue> AddedQueues { get; set; }
        public ICollection<SavedStation> SavedStations { get; set; }
        public ICollection<Review> Reviews { get; set; }
        public ICollection<LikeDislike> LikesDislikes { get; set; }
        public ICollection<Stream> DjStreams { get; set; }
    }
}
```

Рис. 7 - Код класу User

```
namespace RadioStation.Data
{
    public class RadioStation
    {
        [Key]
        public Guid StationId { get; set; }
        public string StationName { get; set; }
        public string Description { get; set; }
        public Guid CreatedById { get; set; }

        public User CreatedBy { get; set; }
        public ICollection<PlaybackQueue> Playbacks { get; set; }
        public ICollection<SavedStation> SavedByUsers { get; set; }
        public ICollection<Stream> Streams { get; set; }
    }
}
```

Рис. 8 - Код класу RadioStation

```
namespace RadioStation.Data
{
    public class Track
    {
        [Key]
        public Guid TrackId { get; set; }
        public string Title { get; set; }
        public TimeSpan Duration { get; set; }
        public Guid UploadedById { get; set; }

        public User UploadedBy { get; set; }
        public ICollection<PlaybackQueue> Queues { get; set; }
        public ICollection<Review> Reviews { get; set; }
        public ICollection<LikeDislike> LikesDislikes { get; set; }
    }
}
```

Рис. 9 - Код класу Track

```
namespace RadioStation.Data
{
    public class PlaybackQueue
    {
        [Key]
        public Guid QueueId { get; set; }
        public Guid TrackId { get; set; }
        public Guid StationId { get; set; }
        public Guid AddedById { get; set; }
        public int QueuePosition { get; set; }

        public Track Track { get; set; }
        public RadioStation Station { get; set; }
        public User AddedBy { get; set; }
    }
}
```

Рис. 10 - Код класу PlaybackQueue

```
namespace RadioStation.Data
{
    public class SavedStation
    {
        [Key]
        public Guid SavedId { get; set; }
        public Guid UserId { get; set; }
        public Guid StationId { get; set; }
        public DateTime SavedAt { get; set; }

        public User User { get; set; }
        public RadioStation Station { get; set; }
    }
}
```

Рис. 11 - Код класу SavedStation

```
namespace RadioStation.Data
{
    public class Review
    {
        [Key]
        public Guid ReviewId { get; set; }
        public Guid UserId { get; set; }
        public Guid TrackId { get; set; }
        public string Content { get; set; }
        public int Rating { get; set; }
        public DateTime CreatedAt { get; set; }

        public User User { get; set; }
        public Track Track { get; set; }
    }
}
```

Рис. 12 - Код класу Review

```
namespace RadioStation.Data
{
    public class LikeDislike
    {
        [Key]
        public Guid LikeId { get; set; }
        public Guid UserId { get; set; }
        public Guid TrackId { get; set; }
        public bool IsLike { get; set; }
        public DateTime CreatedAt { get; set; }

        public User User { get; set; }
        public Track Track { get; set; }
    }
}
```

Рис. 13 - Код класу LikeDislike

```
namespace RadioStation.Data
{
    public class Stream
    {
        [Key]
        public Guid StreamId { get; set; }
        public Guid StationId { get; set; }
        public Guid DjId { get; set; }
        public DateTime StartTime { get; set; }
        public DateTime? EndTime { get; set; }

        public RadioStation Station { get; set; }
        public User Dj { get; set; }
    }
}
```

Рис. 14 - Код класу Stream

5. Питання до лабораторної роботи:

1. Що таке UML? – універсальна мова для опису та візуалізації систем.
2. Що таке діаграма класів UML? – схема з класами, їхніми полями, методами та зв’язками.
3. Які діаграми UML називають канонічними? – стандартні: класів, об’єктів, варіантів використання, послідовностей тощо.

4. Що таке діаграма варіантів використання? – показує акторів і функції, які вони виконують у системі.
5. Що таке варіант використання? – дія або функція, яку виконує користувач.
6. Які відношення можуть бути відображені на діаграмі використання? – асоціація, include, extend, узагальнення.
7. Що таке сценарій? – опис кроків взаємодії користувача з системою.
8. Що таке діаграма класів? – схема структури системи через класи та їх зв’язки.
9. Які зв’язки між класами ви знаєте? – асоціація, агрегація, композиція, наслідування, залежність.
10. Чим відрізняється композиція від агрегації? – у композиції частини не існують без цілого, в агрегації можуть.
11. Чим відрізняється зв'язки типу агрегації від зв'язків композиції на діаграмах класів? – агрегація має порожній ромб, композиція – зафарбований.
12. Що являють собою нормальні форми баз даних? – правила організації таблиць для уникнення дублювання.
13. Що таке фізична модель бази даних? Логічна? – фізична: як реально зберігається; логічна: концептуальна схема.
14. Який взаємозв'язок між таблицями БД та програмними класами? – таблиця ≈ клас, рядок ≈ об’єкт, стовпчик ≈ поле.

6. Висновок

У процесі виконання лабораторної роботи з теми "Online radio station" було здійснено комплексний аналіз вимог, розроблено моделі системи за допомогою UML, включаючи діаграми варіантів використання, класів та структури даних, а також створено базову архітектуру коду. Використано сучасні підходи до проектування, такі як патерни Repository та асинхронне програмування, для забезпечення гнучкості та масштабовності. Результатом є теоретична основа для реалізації функціональної системи, що відповідає поставленим завданням і може бути розширена в майбутньому.