НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Кафедра інформаційних систем та технологій

Тема: Online radio station

Курсова робота
З дисципліни «Технології розроблення програмного забезпечення»

Керівник
доц. Амонс О.А.
«Допущений до захисту»
_____
(Особистий підпис керівника)
« » _____2025р.
Захищений з оцінкою
_____
(оцінка)

Виконавець
ст. Козир Я. О.
залікова книжка № ___ – _____
гр. ІА-31
_____
(особистий підпис виконавця)
« »_____2025р.

Члени комісії:

_____ _____
(особистий підпис) (розшифровка підпису)
_____ _____
(особистий підпис) (розшифровка підпису)

Київ – 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
(назва навчального закладу)

Кафедра ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Дисципліна «Технології розроблення програмного забезпечення»

Курс 3 Група ІА-31 Семестр 5

ЗАВДАННЯ

на курсову роботу студента

Козиря Ярослава Олександровича

(прізвище, ім’я, по батькові)

1.Тема роботи: Online radio station

2. Строк здачі студентом закінченої роботи

3. Вихідні дані до роботи:

Додаток повинен служити сервером для радіостанції з можливістю мовлення на радіостанцію (64, 92, 128, 196, 224 kb/s) в потоковому режимі; вести облік підключених користувачів і статистику відвідувань і прослуховувань; налаштувати папки з піснями і можливість вести списки програвання або playlists (не відтворювати всі пісні).

4. Зміст розрахунково – пояснювальної записки (перелік питань, що підлягають розробці)

1. Аналіз предметної області, загальний опис проєкту та огляд існуючих рішень. 2. Формулювання функціональних та нефункціональних вимог до системи. 3. Розробка сценаріїв використання (Use Case) та концептуальної моделі системи. 4. Проєктування архітектури системи, вибір технологічного стеку та баз даних.

Додатки:

Додаток А - проектування паттернів

Додаток Б – конфігурація бази даних та ініціалізація

5.Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Рис. 1.1 — Діаграма варіантів використання; Рис. 1.2–1.5 — Діаграми послідовності; Рис. 1.5 — Діаграма класів репозиторіїв; Рис. 1.6 — Діаграма класів даних; Рис. 1.7 — Діаграма компонентів; Рис. 1.8 — Діаграма розгортання.

6. Дата видачі завдання 17.09.2025

КАЛЕНДАРНИЙ ПЛАН

№, п/п	Назва етапів виконання курсової роботи	Строк виконання етапів роботи	Підписи або примітки
1.	Підбір та вивчення літератури	30.09.2025	
2.	Проектування та написання розділу 1	31.10.2025	
3.	Розробка та написання розділу 2	20.11.2025	
4.	Подання курсової роботи на перевірку	25.11.2025	
5.	Захист курсової роботи	08.12.2025	
6.			
7.			
8.			
9.			
10.			
11.			
12.			
13.			
14.			
15.			
16.			
17.			
18.			

Студент _____ _____Ярослав КОЗИР _____
(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник_____ __Олександр АМОНС__
(підпис) (Ім’я ПРІЗВИЩЕ)

«____»_____ 20___ р.

ЗМІСТ

ВСТУП

Сучасний світ неможливо уявити без цифрових медіа, серед яких онлайн-радіо та аудіострімінг займають значне місце. Популярність інтернет-радіостанцій постійно зростає завдяки їх доступності, різноманітності контенту та можливості персоналізації. Однак створення та підтримка таких сервісів вимагає вирішення низки технічних завдань, пов'язаних із забезпеченням стабільного мовлення, керуванням великими обсягами аудіоконтенту, обліком слухачів та наданням гнучких інструментів для керування ефіром.

Актуальність даної роботи полягає у необхідності розробки надійних, масштабованих та легких у супроводженні програмних рішень для онлайн-радіостанцій. Неефективна архітектура може призвести до проблем з продуктивністю, складнощів у додаванні нових функцій та високих витрат на підтримку.

Метою даного курсового проєкту є проєктування та розробка програмної системи "Online Radio Station". Система має виконувати функції сервера для потокового мовлення аудіо з різною якістю, вести облік підключених користувачів, збирати статистику прослуховувань, а також надавати можливості для керування аудіоконтентом та списками відтворення (плейлистами).

Для досягнення поставленої мети особливу увагу буде приділено архітектурі системи та застосуванню сучасних патернів проєктування. Це дозволить створити гнучку основу для подальшого розвитку функціоналу. Проєкт буде реалізовано з використанням клієнт-серверної архітектури , що є стандартом для подібних веб-орієнтованих систем.

У даній роботі буде проведено аналіз предметної області, розроблено архітектуру системи з використанням UML-діаграм, описано реалізацію ключових функціональних можливостей та продемонстровано застосування таких патернів проєктування, як Ітератор, Адаптер, Factory Method, Facade, Visitor та інших, для вирішення конкретних завдань у межах проєкту.

1 ПРОЄКТУВАННЯ СИСТЕМИ

1.1. Огляд існуючих рішень

Ринок онлайн-радіо та аудіострімінгу сьогодні представлений великою кількістю рішень, від глобальних платформ до нішевих сервісів. Серед найпопулярніших можна виділити:

- Глобальні стрімінгові сервіси: Платформи, такі як Spotify (з функцією Radio), Apple Music, Pandora, пропонують користувачам персоналізовані радіостанції на основі їхніх вподобань, доступ до величезних музичних бібліотек та готові плейлисти. Їхніми сильними сторонами є зручні інтерфейси та рекомендаційні алгоритми.
- Агрегатори інтернет-радіостанцій: Сервіси типу TuneIn Radio збирають тисячі існуючих інтернет-радіостанцій з усього світу, надаючи зручний доступ до різноманітного контенту (музика, новини, подкасти).
- Платформи для створення власних станцій: Технології, такі як SHOUTcast та Icecast, є популярними рішеннями з відкритим кодом, які дозволяють ентузіастам та компаніям створювати власні інтернет-радіостанції. Вони надають базовий функціонал для потокового мовлення, але часто вимагають значних технічних налаштувань та додаткових інструментів для керування контентом і користувачами.

Аналіз існуючих рішень показує високий попит на послуги онлайн-радіо, але також виявляє потребу у гнучких системах, які дозволяють створювати власні станції з розширеними можливостями керування контентом, плейлистами та статистикою, що і є метою даного проєкту.

1.2. Загальний опис проєкту

Проєкт присвячений розробці програмної системи "Online Radio Station", призначеної для організації професійного потокового аудіомовлення в мережі

Інтернет. На відміну від глобальних музичних платформ (як Spotify) або "сирих" серверних технологій (як Icecast), дана система проєктується як гнучке "коробкове" рішення (CMS для радіо) з єдиним веб-інтерфейсом керування та повним циклом обробки медіа-контенту.

Система базується на класичній клієнт-серверній архітектурі.

- Серверна частина (Backend): Реалізована на платформі ASP.NET Core. Вона виступає центральним вузлом, який бере на себе всю бізнес-логіку: автентифікацію користувачів, транскодування аудіофайлів у формат HLS (HTTP Live Streaming), формування динамічних плейлистів, синхронізацію часу ефіру та збір статистики.
- Клієнтська частина (Frontend): Реалізована як "тонкий клієнт" (веб-інтерфейс), що забезпечує взаємодію користувачів із системою через браузер без необхідності встановлення додаткового ПЗ.

Проєкт чітко розмежовує функціонал та права доступу на основі трьох ключових ролей:

1. Адміністратор: Володіє повними правами на керування системою. Його задачі включають створення та налаштування радіостанцій, модерацію користувачів (включно з блокуванням та призначенням ролей), а також керування Глобальною бібліотекою треків. Адміністратор має доступ до аналітичного модуля для перегляду статистики наповнення контентом та активності діджеїв.
2. Діджей (DJ): Є ключовою фігурою ефіру і прив'язаний до конкретної радіостанції. Функціонал Діджея включає:
    - Керування контентом: Завантаження аудіофайлів, які система автоматично конвертує у 5 варіантів якості (64, 92, 128, 196, 224 kb/s).
    - Формування ефіру: Створення черги відтворення (плейлиста) з можливістю зміни порядку треків та їх видалення.
    - Керування трансляцією: Запуск та завершення прямого ефіру (сесії) через "Ефірний пульт".

- Модерація в реальному часі: Використання "Студійного монітора" для контролю поточного треку, а також функції "Shuffle" (випадкове відтворення) та "Skip" (пропуск треку без видалення).

3. Слухач: Кінцевий споживач контенту. Слухачі можуть переглядати вітрину доступних станцій та додавати їх у список "Улюблених". Ключовою особливістю є синхронізоване відтворення: всі слухачі, підключені до однієї станції, чують один і той самий фрагмент треку одночасно (ефект живого радіо). Також реалізовано систему рейтингу (Like/Dislike) для оцінки поточного треку.

Основний сценарій роботи передбачає повний цикл: від завантаження "сирого" MP3-файлу Діджеєм до його автоматичної обробки сервером та доставки Слухачеві у вигляді адаптивного HLS-потоку, стійкого до якості інтернет-з'єднання.

1.3. Вимоги до застосунків системи

1.3.1. Функціональні вимоги до системи

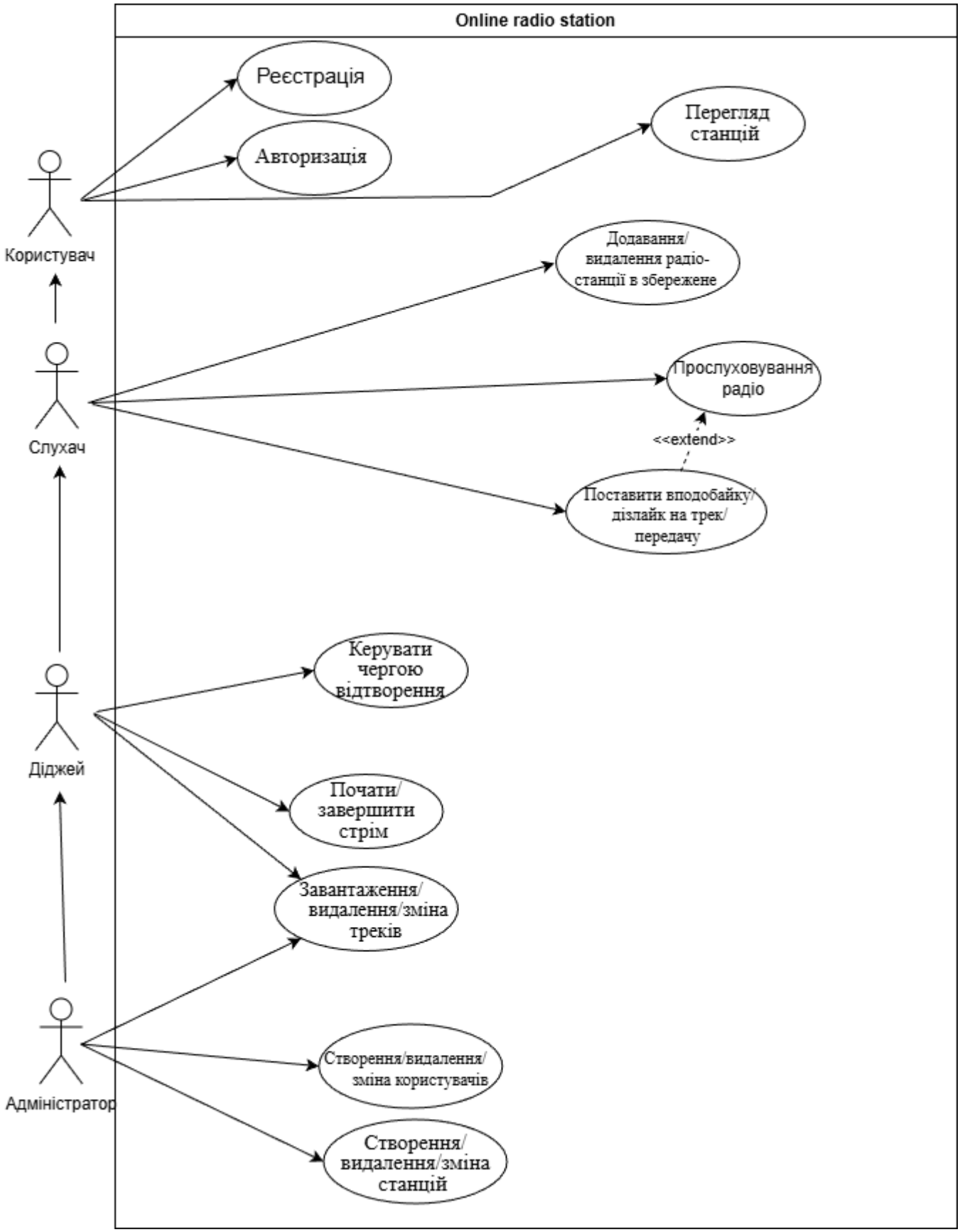

Рис. 1. 1 - Діаграма варіантів використання

№	Актор	Назва вимоги	Короткий опис
1	Користувач	Реєстрація	Можливість створити новий обліковий запис (Слухач, Діджей, Адміністратор) із захищеним збереженням пароля.
2	Користувач	Авторизація	Вхід у систему з використанням логіна та пароля, розподіл прав доступу згідно з роллю.
3	Користувач	Перегляд станцій	Перегляд списку доступних радіостанцій на головній сторінці та в особистому кабінеті.
4	Слухач	Прослуховування радіо	Відтворення потокового аудіо (HLS) обраної станції. Ефір синхронізований для всіх слухачів (імітація живого радіо).
5	Слухач	Оцінка треків (Лайк/Дизлайк)	Можливість оцінити поточний трек. Рейтинг впливає на статистику треку.
6	Слухач	Збереження станцій (Обране)	Додавання станцій до особистого списку "Улюблених" для швидкого доступу.
7	Діджей	Завантаження та обробка треків	Завантаження аудіофайлів (MP3), які автоматично конвертуються системою у формат HLS із різними бітрейтами (64, 92, 128, 196, 224 kb/s).
8	Діджей	Керування плейлистом	Формування черги відтворення зі своєї бібліотеки. Можливість змінювати порядок треків (сортування) та видаляти їх з ефіру.
9	Діджей	Керування ефіром (Пульт)	Запуск та зупинка прямого ефіру (сесії). Перегляд статусу "Онлайн/Офлайн" та таймера тривалості сесії.
10	Діджей	Модерація ефіру в реальному часі	Можливість "м'якого пропуску" (Skip) треку без його видалення та увімкнення

			режиму "Випадкове відтворення" (Shuffle) при старті.
11	Адміністратор	Керування користувачами	Блокування користувачів (Бан), зміна ролей, призначення Діджеїв на конкретні радіостанції.
12	Адміністратор	Керування станціями	Створення, редагування та видалення радіостанцій у системі.
13	Адміністратор	Глобальна бібліотека та Статистика	Перегляд усіх завантажених треків у системі, фізичне очищення файлів ("сміття"), перегляд статистики прослуховувань (Visitor).

Таблиця 1.1. Функціональні вимоги

1.3.2. Нефункціональні вимоги до системи

№	Назва	Опис
1	Продуктивність	Забезпечення стабільного потокового мовлення без значних затримок для N одночасних слухачів; швидке завантаження основних сторінок інтерфейсу.
2	Надійність	Висока доступність сервісу (до 99%); коректна обробка помилок та відновлення після збоїв
3	Масштабованість	Архітектура повинна дозволяти майбутнє збільшення навантаження (кількість слухачів, обсяг контенту) без кардинальної перебудови.
4	Безпека	Захист облікових записів користувачів (хешування паролів); розмежування доступу до функцій відповідно до ролей

5	Зручність використання	Інтуїтивно зрозумілий інтерфейс для всіх категорій користувачів.
6	Супроводжуваність	Добре структурований, читабельний код, що полегшує внесення змін та виправлення помилок
7	Сумісність	Коректна робота веб-інтерфейсу в останніх версіях популярних браузерів

Таблиця 1.2. Нефункціональні вимоги

1.4. Сценарії використання системи

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	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	-

Таблиця 1.3. Сценарій використання «Прослуховування радіо»

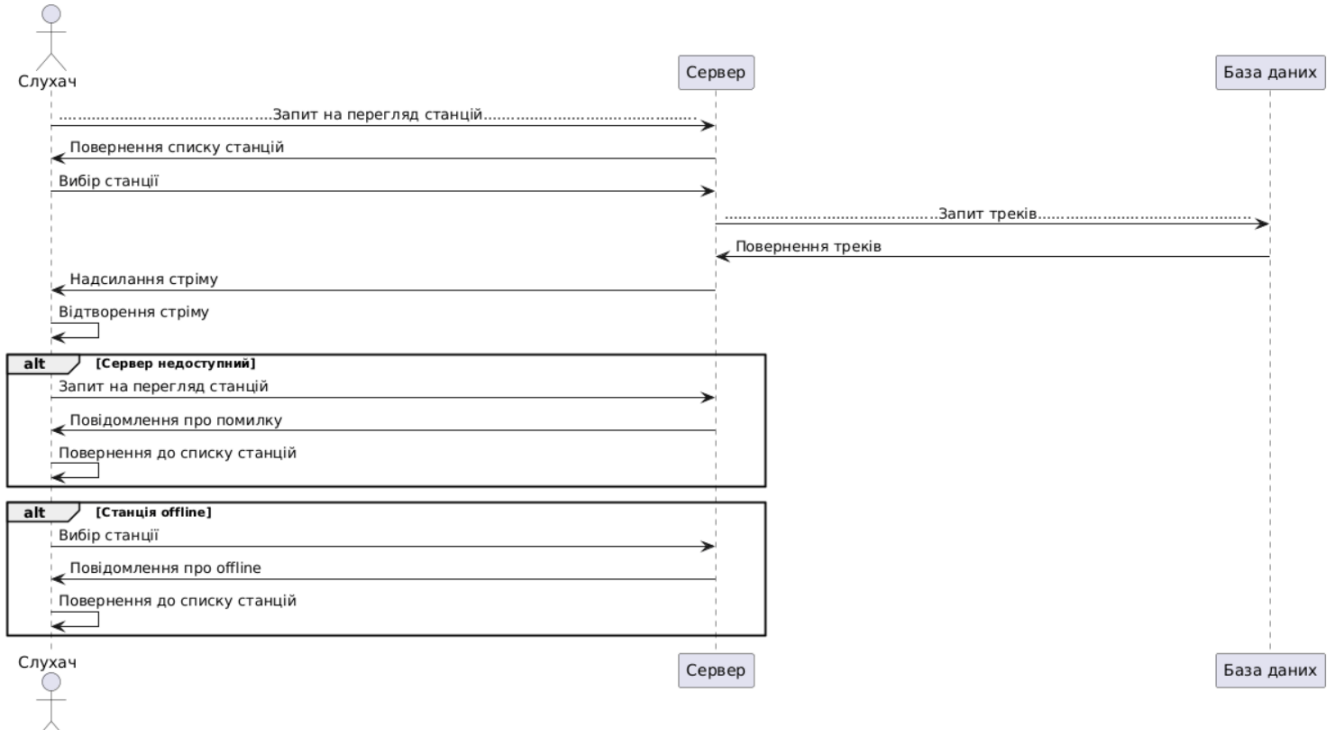

Рис. 1. 2 - Діаграма послідовності «Прослуховування радіо»

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	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	-

Таблиця 1.4. Сценарій використання «Керування чергою відтворення»

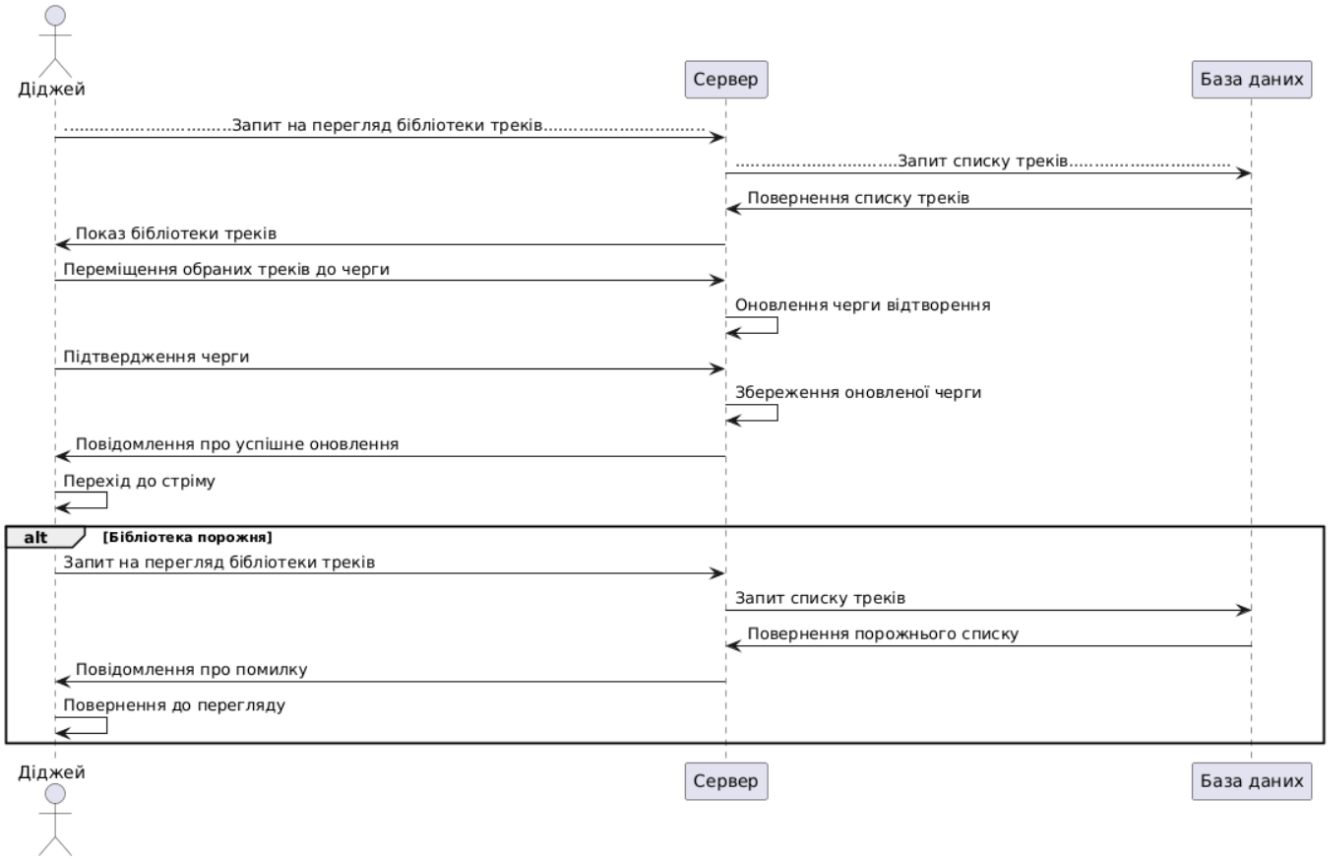

Рис. 1. 3 - Діаграма послідовності «Керування чергою відтворення»

Авторизація користувача	
Передумови	Користувач має зареєстрований акаунт в системі.
Постумови	Користувач увійшов до системи та отримав доступ до функціоналу, що відповідає його ролі (Слухач, DJ, Адміністратор).
Взаємодіючі сторони	Користувач, Система.
Короткий опис	Цей варіант використання визначає процес входу користувача в систему.
Основний потік подій	1. Користувач відкриває сторінку авторизації.
	2. Користувач вводить свій логін (ім'я користувача) та пароль.
	3. Користувач натискає кнопку "Увійти".
	4. Система перевіряє відповідність логіна та пароля у базі даних.
	5. Система ідентифікує роль користувача.
	6. Система перенаправляє користувача на відповідну головну сторінку (панель Слухача, DJ або Адміністратора).
Винятки	Неправильний логін або пароль. Система повідомляє користувача про помилку, вхід не відбувається.

	Акаунт заблоковано. Система повідомляє про блокування акаунту.
Примітки	-

Таблиця 1.5. Сценарій використання «Авторизація користувача»

Почати/завершити стрім	
Передумови	Діджей авторизований у системі. Черга відтворення (плейлист) створена та готова до ефіру.
Постумови	Стан радіостанції змінено на "В ефірі". Слухачі можуть підключитися до потоку.
Взаємодіючі сторони	Діджей, Система, Слухачі (непрямо).
Короткий опис	Визначає процес запуску прямого ефіру на радіостанції.
Основний потік подій	1. Діджей переходить на свою панель керування стрімом.
	2. Система показує поточний статус "Офлайн" та готову чергу відтворення.
	3. Діджей натискає кнопку "Почати стрім".
	4. Система ініціалізує сервер мовлення, підключає аудіопотік та починає відтворення згідно з чергою.
	5. Система змінює статус станції на "В ефірі".
	6. Система починає приймати підключення від слухачів.
Винятки	Помилка сервера мовлення. Система не може почати стрім і повідомляє Діджея про технічну помилку.
	Черга відтворення порожня. Система блокує початок стріму і просить спершу додати треки.
Примітки	Сценарій "Завершити стрім" є зворотним процесом.

Таблиця 1.6. Сценарій використання «Почати/завершити стрім»

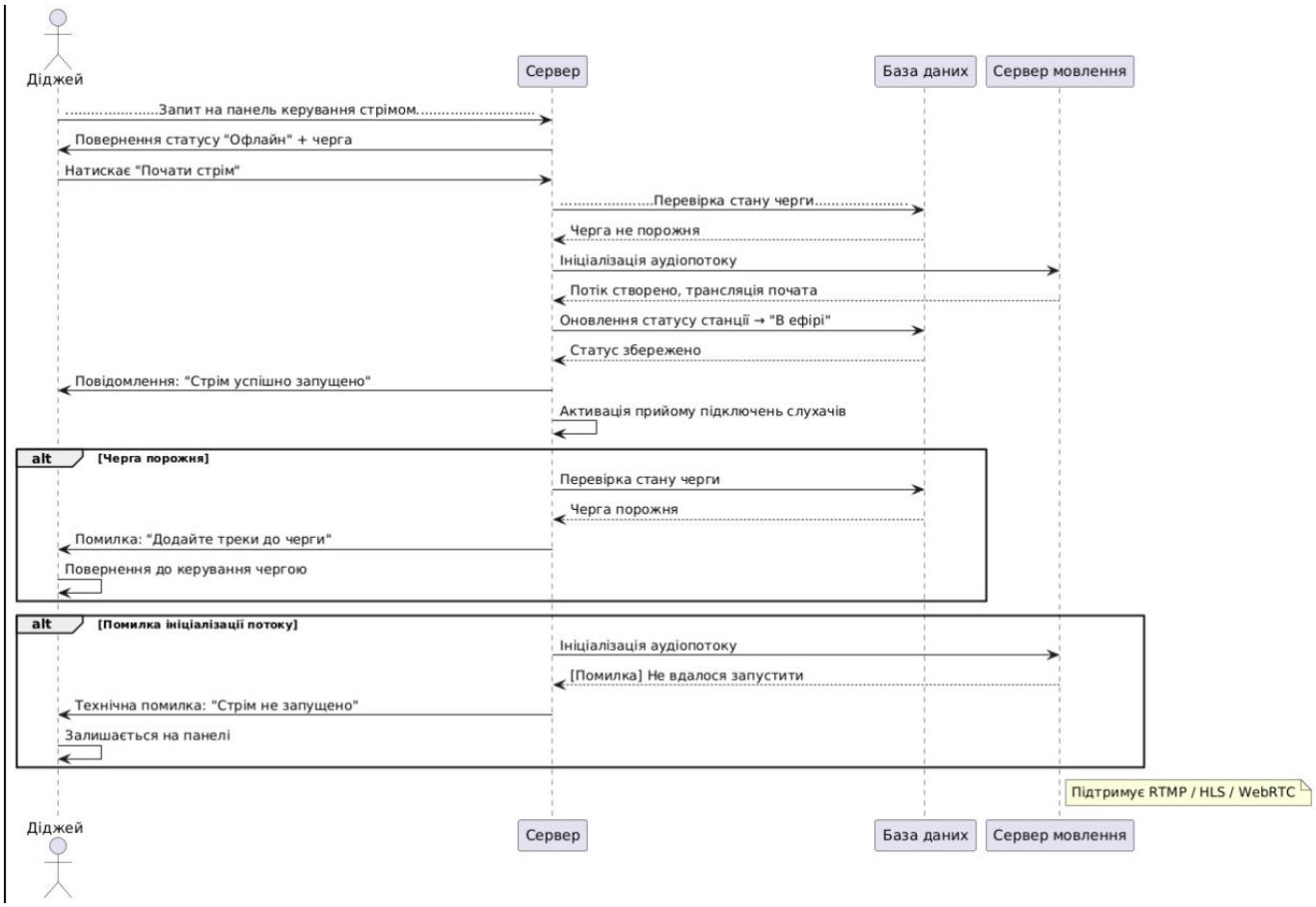

Рис. 1. 4 - Діаграма послідовності «Початок стріму»

Керування користувачами	
Передумови	Адміністратор авторизований у системі та має відповідні права.
Постумови	Обліковий запис користувача змінено (наприклад, змінено роль, заблоковано) або видалено з системи.
Взаємодіючі сторони	Адміністратор, Система.
Короткий опис	Визначає процес керування обліковими записами користувачів з боку адміністратора.
Основний потік подій	1. Адміністратор відкриває панель "Керування користувачами".
	2. Система відображає список користувачів.
	3. Адміністратор знаходить потрібного користувача (наприклад, за допомогою пошуку).
	4. Адміністратор обирає дію "Редагувати" для обраного користувача.
	5. Система відкриває форму редагування профілю.
	6. Адміністратор змінює роль користувача (наприклад, з "Слухач" на "DJ").
	7. Адміністратор зберігає зміни.

	8. Система оновлює дані користувача в базі даних.
Винятки	Адміністратор намагається видалити власний обліковий запис. Система блокує дію та видає попередження.
Примітки	-

Таблиця 1.7. Сценарій використання «Керування користувачами»

Керування радіостанціями	
Передумови	Адміністратор авторизований у системі та має відповідні права.
Постумови	Станцію створено, оновлено або видалено з системи.
Взаємодіючі сторони	Адміністратор, Система.
Короткий опис	Визначає процес керування радіостанціями в системі.
Основний потік подій	1. Адміністратор відкриває панель "Керування користувачами".
	2. Адміністратор натискає "Створити нову станцію".
	3. Система відкриває форму для введення даних (назва станції, опис).
	4. Адміністратор заповнює поля та зберігає зміни.
	5. Система створює новий запис про станцію в базі даних.
	6. Система повертає адміністратора до оновленого списку станцій.
Винятки	Станція з такою назвою вже існує. Система повідомляє про помилку.
	Спроба видалити станцію, на якій зараз йде активний стрім. Система може заблокувати дію та видати попередження.
Примітки	Аналогічні потоки існують для редагування та видалення станцій.

Таблиця 1.8. Сценарій використання «Керування радіостанціями»

Реєстрація користувача	
Передумови	Користувач не авторизований у системі.
Постумови	У базі даних створено новий обліковий запис користувача зі статусом "Слухач". Користувач може увійти до системи, використовуючи ці дані.
Взаємодіючі сторони	Користувач, Система.
Короткий опис	Цей варіант використання визначає процес створення нового облікового запису.

Основний потік подій	1. Користувач відкриває сторінку реєстрації.
	2. Користувач вводить необхідні дані (наприклад, ім'я користувача, email, пароль).
	3. Користувач натискає кнопку "Зареєструватися".
	4. Система перевіряє дані на валідність (наприклад, чи не зайнятий email або ім'я користувача).
	5. Система створює новий запис користувача в базі даних із роллю "Слухач".
	6. Система повідомляє про успішну реєстрацію та перенаправляє на сторінку входу.
Винятки	Ім'я користувача або email вже зайняті. Система повідомляє про помилку, реєстрація не відбувається.
	Пароль не відповідає вимогам безпеки. Система повідомляє про помилку.
Примітки	-

Таблиця 1.9. Сценарій використання «Реєстрація користувача»

1.5. Концептуальна модель системи

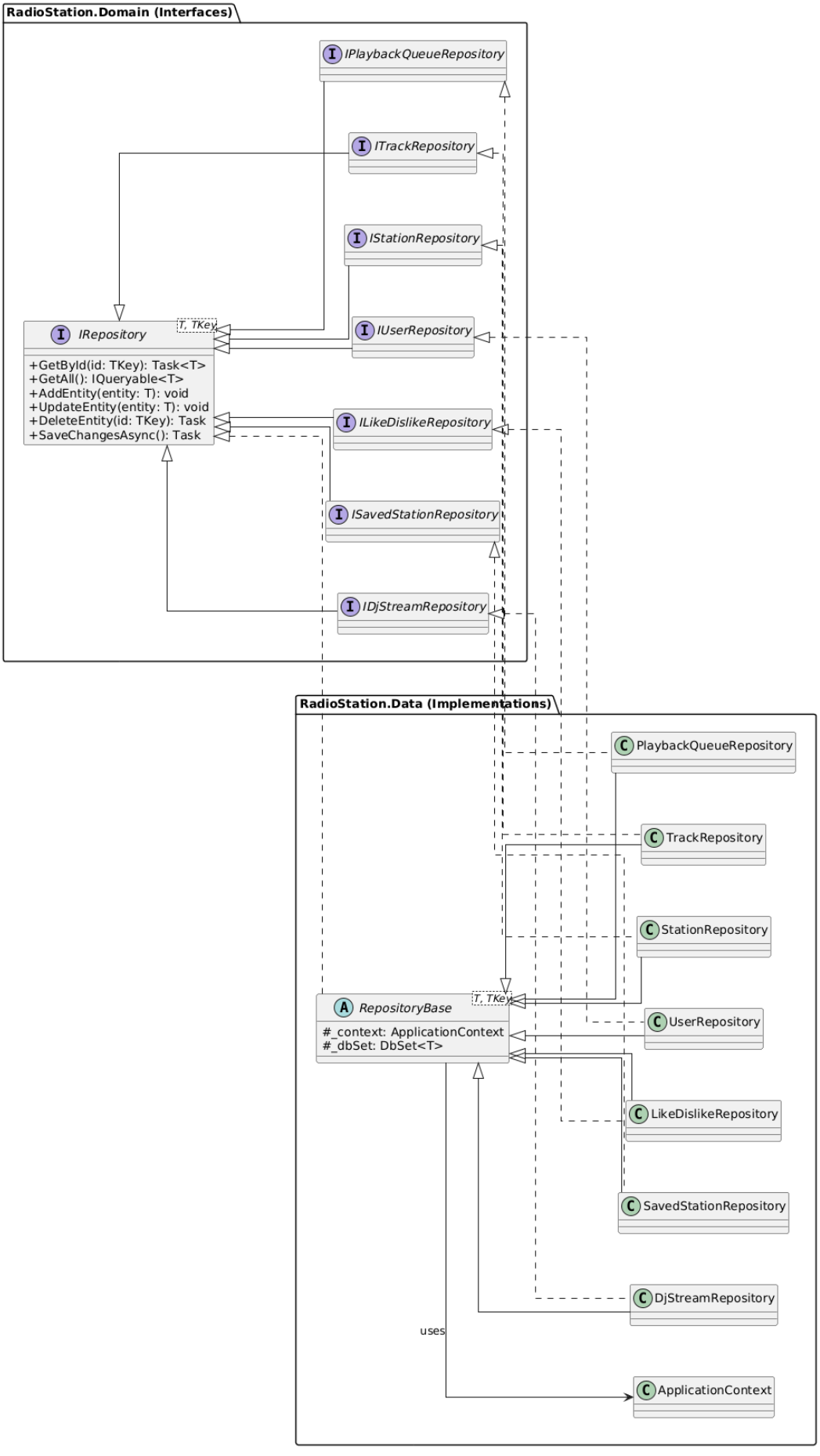
RadioStation.Domain (Interfaces)
IPlaybackQueueRepository
ITrackRepository
IStationRepository
IRepository
T, TKey
+GetById(id: TKey): Task<T>
+GetAll(): IQueryable<T>
+AddEntity(entity: T): void
+UpdateEntity(entity: T): void
+DeleteEntity(id: TKey): Task
+SaveChangesAsync(): Task
IUserRepository
ILikeDislikeRepository
ISavedStationRepository
IDjStreamRepository
RadioStation.Data (Implementations)
PlaybackQueueRepository
TrackRepository
StationRepository
RepositoryBase
T, TKey
#_context: ApplicationContext
#_dbSet: DbSet<T>
UserRepository
LikeDislikeRepository
SavedStationRepository
DjStreamRepository
uses
ApplicationContext

Рис. 1. 5 - Діаграма класів репозиторіїв

На діаграмі класів (Рис. 1.6) представлена реалізація патерну "Репозиторій" (Repository), який використовується у проєкті для ізоляції логіки доступу до даних від решти бізнес-логіки. Архітектура складається з двох основних частин: узагальнених (generic) компонентів та їхніх специфічних реалізацій.

1. Узагальнені (Generic) компоненти Ці компоненти створюють єдиний "контракт" та спільну реалізацію для всіх репозиторіїв, що дозволяє уникнути дублювання коду.

Інтерфейс IRepository<T, TKey> Призначення: Універсальний "контракт", який описує базовий набір CRUD-операцій (Create, Read, Update, Delete), що має виконувати будь-який репозиторій. Параметри:

- T: Тип сутності (наприклад, User або RadioStationEntity).
- TKey: Тип первинного ключа сутності (наприклад, Guid або int).

Методи:

- GetById(TKey): Повертає задачу (Task), результатом якої є об'єкт типу T за його ключем.
- GetAll(): Повертає запит IQueryable<T> для отримання всіх записів.
- AddEntity(T): Додає новий об'єкт T до контексту (void).
- UpdateEntity(T): Позначає існуючий об'єкт T як змінений (void).
- DeleteEntity(TKey): Асинхронно видаляє об'єкт за його ключем (Task).
- SaveChangesAsync(): Асинхронно зберігає всі зміни в базі даних (Task).

Абстрактний клас RepositoryBase<T, TKey> Призначення: Спільна реалізація інтерфейсу IRepository. Усі конкретні репозиторії успадковують цей клас, щоб автоматично отримати всю базову логіку. Атрибути:

- _context: ApplicationContext: Захищене поле, що зберігає посилання на контекст бази даних (DbContext з Entity Framework).
- _dbSet: DbSet<T>: Захищене поле, що зберігає посилання на конкретну таблицю (DbSet) у базі даних, яка відповідає сутності T. Зв'язки: Реалізує інтерфейс IRepository<T, TKey>.

2. Специфічні реалізації для сутностей Для кожної сутності (наприклад, User, Track, DjStream) створюється власний інтерфейс та клас-реалізація. Це дозволяє за

потреби додавати унікальні методи (наприклад, GetUserByUsernameAsync), водночас успадковуючи всю базову логіку.

Сутність	Специфічний Інтерфейс	Клас-Реалізація
User	IUserRepository	UserRepository
Station	IStationRepository	StationRepository
LikeDislike	ILikeDislikeRepository	LikeDislikeRepository
DjStream	IDjStreamRepository	DjStreamRepository
SavedStation	ISavedStationRepository	SavedStationRepository
Track	ITrackRepository	TrackRepository
PlaybackQueue	IPlaybackQueueRepository	PlaybackQueueRepository

. Таблиця 1.10. Реалізація специфічних реалізацій для сутностей системи

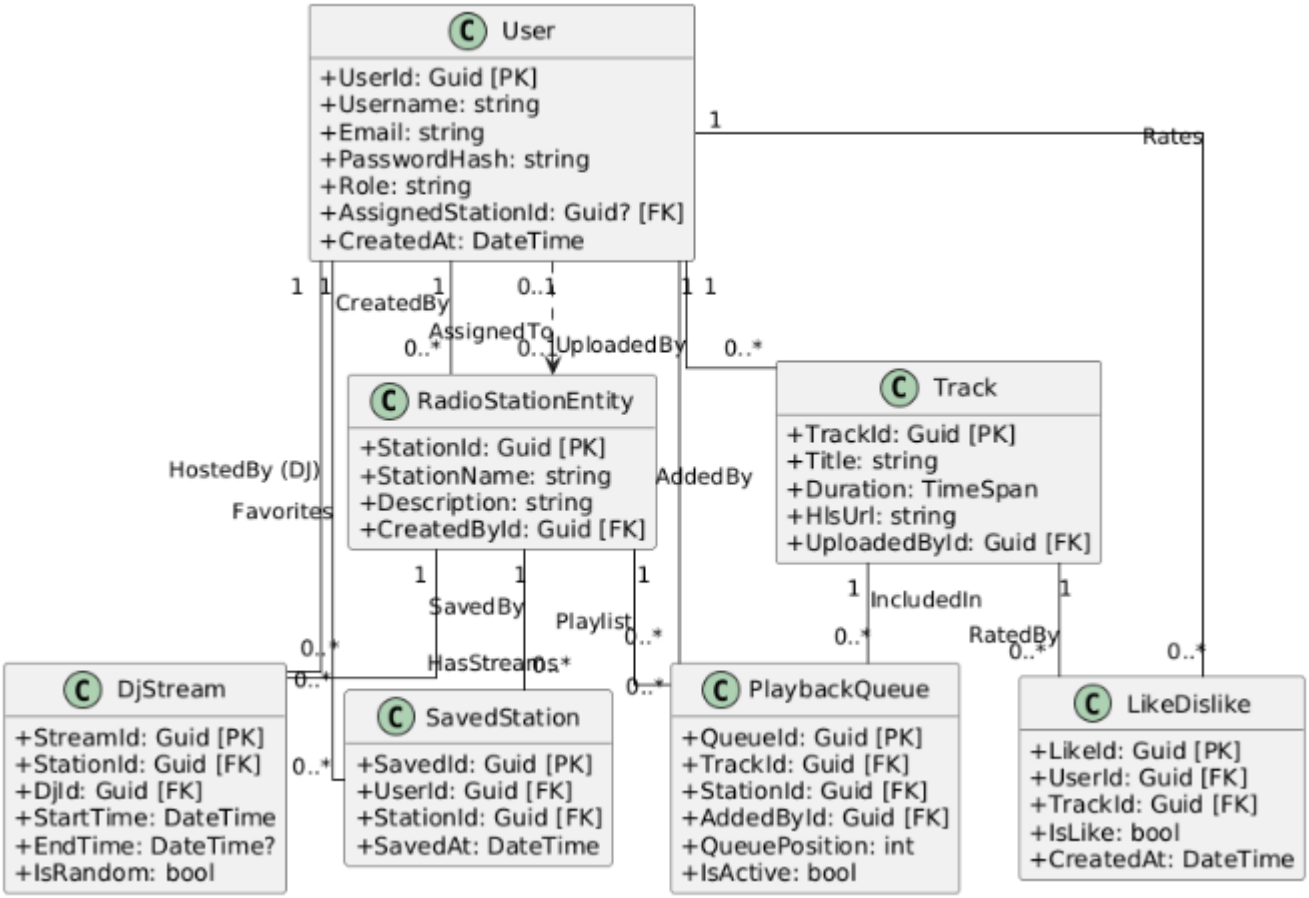

Рис. 1. 6 - Діаграма класів даних

Концептуальна модель системи представлена на діаграмі класів (Рис. 1.6), яка описує ключові сутності (класи), їхні атрибути та зв’язки між ними.

Клас (Сутність)	Атрибути (Властивості)	Короткий опис
User	userId, username, passwordHash, role, createdAt, assignedStationId	Зберігає облікові дані, роль та прив'язку Діджея до конкретної станції.
RadioStation	stationId, stationName, description, createdBy	Описує радіостанцію в системі.
Track	trackId, title, duration, hlsUrl, uploadedBy	Медіа-файл. Містить HlsUrl — шлях до потокового маніфесту .m3u8.
DjStream	streamId, stationId, djId, startTime, endTime, isRandom	Сесія ефіру. IsRandom визначає, чи увімкнено режим Shuffle.
PlaybackQueue	queueId, trackId, stationId, isActive, queuePosition	Елемент черги. IsActive дозволяє тимчасово пропускати трек без видалення.
SavedStation	savedId, userId, stationId, savedAt	Зв'язок для списку "Улюблені станції".
LikeDislike	likeId, userId, trackId, isLike, createdAt	Рейтинг треків (Лайк/Дизлайк).

Таблиця 1.11. Основні класи

2. Зв’язки між класами (Multiplicity)

Зв'язки (асоціації) на діаграмі показують, як сутності взаємодіють між собою:

- User (1) до RadioStation (0..*): Один користувач (DJ/Admin) може створити багато радіостанцій.

- User (1) до Track (0..*): Один користувач (DJ) може завантажити багато треків.
- User (1) до Stream (0..*): Один користувач (DJ) може провести багато стрімів (сесій ефіру).
- User (1) до PlaybackQueue (0..*): Один користувач (DJ) може додати багато треків до черги.
- User (1) до SavedStation (0..*): Один користувач (Слухач) може зберегти багато станцій ("Обране").
- User (1) до LikeDislike (0..*): Один користувач (Слухач) може поставити багато лайків/дизлайків.
- RadioStation (1) до Stream (0..*): Одна радіостанція може мати багато стрімів (історію трансляцій).
- RadioStation (1) до PlaybackQueue (0..*): Одна радіостанція може мати багато треків у своїй черзі відтворення.
- RadioStation (1) до SavedStation (0..*): Одну радіостанцію можуть зберегти багато користувачів.
- Track (1) до PlaybackQueue (0..*): Один і той самий трек може бути доданий до черги відтворення багато разів (або на різних станціях).
- Track (1) до LikeDislike (0..*): Один трек може мати багато лайків/дизлайків.

1.6. Вибір бази даних

Для коректного функціонування інформаційної системи необхідно забезпечити надійне та стабільне збереження даних. У межах проєкту база даних використовується для зберігання ключових сутностей системи, зокрема: користувачів (Users), радіостанцій (Stations), треків (Tracks), плейлистів, черги відтворення (PlaybackQueue), даних про стріми (DjStream) та улюблених станцій користувачів (SavedStations).

На початковому етапі розробки було проведено аналіз кількох поширених СУБД: PostgreSQL, MySQL, Microsoft SQL Server та SQLite. PostgreSQL виявилася надмірно складною для потреб поточного застосунку, а хоча MySQL є швидкою та

популярною, її інтеграція з .NET дещо поступається нативним можливостям Microsoft SQL Server.

Початковий вибір було зроблено на користь Microsoft SQL Server, оскільки ця СУБД має низку значних переваг:

Нативна інтеграція з екосистемою .NET – забезпечує стабільність, продуктивність та простоту налаштувань.

Потужні безкоштовні версії – Developer та Express Edition покривають усі потреби навчального та невеликого виробничого проєкту.

Зручні професійні інструменти адміністрування – SSMS значно спрощує розробку та тестування.

Масштабованість і надійність – SQL Server є придатним як для невеликих додатків, так і для корпоративних систем.

Широка документація та спільнота – сприяють швидкому вирішенню технічних питань.

Однак у процесі подальшого структурування даних та підготовки до презентації проєкту було прийнято рішення змінити підхід. Щоб викладач та будь-який користувач могли легко запустити систему без необхідності встановлення та налаштування повноцінного серверного оточення, було обрано SQLite як СУБД для фінальної демонстрації.

Це рішення обґрунтоване такими перевагами:

Повна портативність та zero-configuration — база зберігається у вигляді одного файлу (RadioStation.db), що робить розгортання максимально простим.

Повна сумісність з Entity Framework Core та Code-First підходом — структура таблиць автоматично створюється при запуску.

Достатність функціоналу для завдань курсового проєкту — SQLite забезпечує транзакційність, реляційні зв’язки та потрібні типи даних.

Єдиним недоліком цього рішення є те, що SQLite поступається SQL Server за рівнем безпеки, масштабованості та стійкості до високих навантажень. Однак з

огляду на те, що проєкт є навчальним та виконується студентом 3 курсу, такий компроміс є цілком виправданим і дозволяє спростити оцінювання та запуск проєкту.

1.7. Вибір мови програмування та середовища розробки

Для реалізації серверної частини веб-застосунку «Online Radio Station» було обрано мову програмування C# та фреймворк ASP.NET Core.

Мова програмування C# є сучасною, об’єктно-орієнтованою мовою зі строгою типізацією. Вона ідеально підходить для створення складних та надійних систем. Для даного проєкту C# було обрано завдяки її глибокій інтеграції з фреймворком ASP.NET Core, потужній стандартній бібліотеці та першокласній підтримці роботи з базами даних через Entity Framework Core.

Фреймворк ASP.NET Core використовується для створення бекенду веб-застосунку. Його ключові переваги:

- Висока продуктивність: Один з найшвидших сучасних веб-фреймворків.
- Кросплатформеність: Додатки можуть запускатися на Windows, Linux та macOS.
- Архітектура MVC (Model-View-Controller): Фреймворк підтримує цей шаблон, що забезпечує чітке розділення логіки програми (Model), користувацького інтерфейсу (View) та керування запитами (Controller). Це робить систему гнучкою та зручною для підтримки.
- Вбудоване впровадження залежностей (DI): Спрощує реалізацію складних архітектурних патернів, таких як "Репозиторій" та "Сервіси", які використовуються у цьому проєкті.

Як середовище розробки було обрано Visual Studio Code (VS Code). Це легкий, швидкий та кросплатформений редактор коду. Завдяки потужним розширенням для C# та .NET, він надає всі необхідні можливості для повноцінної розробки. Важливою перевагою є наявність інтегрованого терміналу, який активно використовувався для створення структури проєкту за допомогою dotnet CLI та керування системою контролю версій Git.

1.8. Проєктування розгортання системи

Для повного розуміння архітектури системи необхідно розглянути її з двох точок зору: логічної та фізичної. Логічна архітектура визначає, з яких програмних частин складається додаток, тоді як фізична архітектура показує, як вони розміщуються на реальному обладнанні.

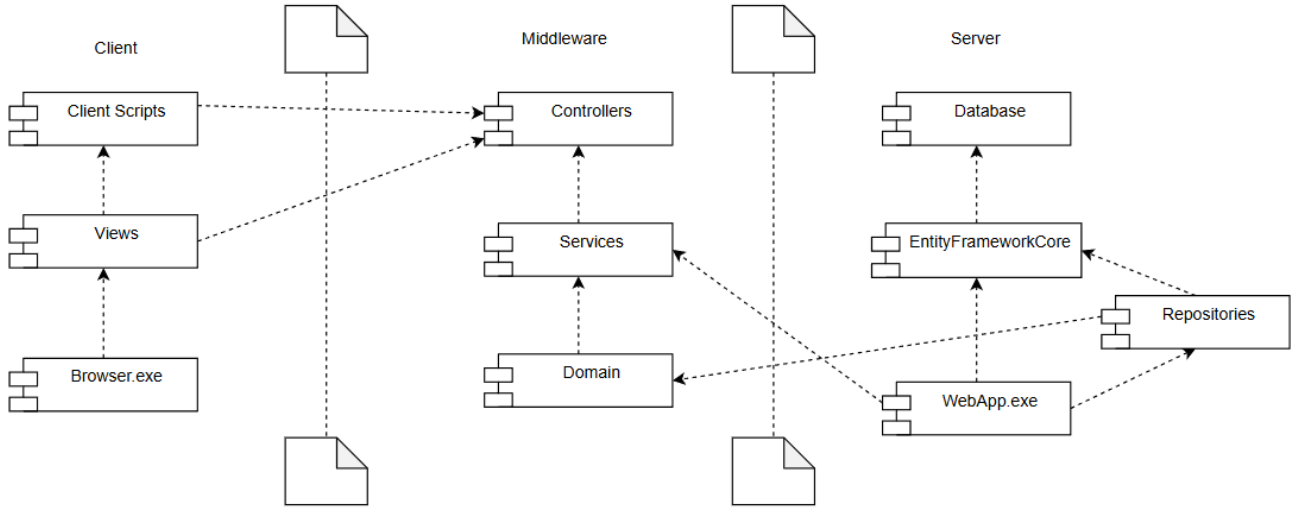

Рис. 1. 7 - Діаграма компонентів

На діаграмі компонентів зображено логічну архітектуру системи, поділену на три основні шари:

- Шар представлення (Client): Представлені компоненти, що відповідають за інтерфейс користувача та взаємодію з ним у браузері (Browser.exe, Views, Client Scripts).
- Шар бізнес-логіки (Middleware): Тут розташовані модулі, що реалізують основну логіку додатку. Вони містять контролери API (Controllers), сервіси (Services) та загальні моделі даних (Domain).
- Шар інфраструктури та доступу до даних (Server/Infrastructure): У цій частині показано компоненти, що відповідають за запуск додатку (WebApp.exe) та

технологічний фреймворк для доступу до даних (EntityFrameworkCore, Repositories).

Компонент Database показаний окремо, і з ним взаємодіє шар доступу до даних (Repositories).

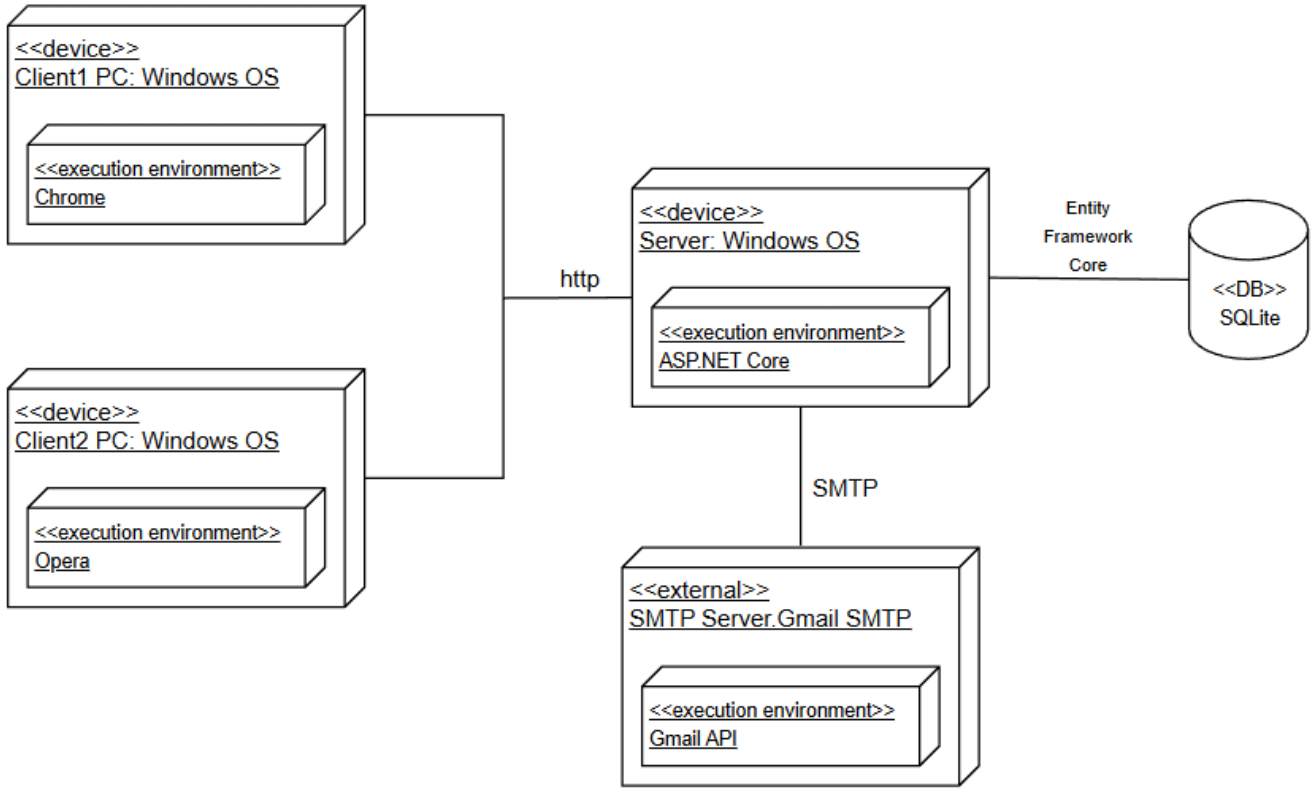

Рис. 1. 8 - Діаграма розгортання

На діаграмі розгортання зображено фізичну архітектуру веб-застосунку "Online radio station". Система побудована за клієнт-серверною моделлю та включає такі вузли:

- Клієнтські вузли (Client PC): Представлені як персональні комп'ютери під керуванням Windows OS. Вони взаємодіють із системою через веб-браузери (Chrome, Opera), які виступають середовищем виконання для клієнтської частини.
- Серверний вузол (Server): Це центральний вузол, що працює на Windows OS та виконує основну логіку додатку у середовищі ASP.NET Core. Він відповідає за обробку запитів від клієнтів та бізнес-логіку.

- Вузол бази даних (Database): Фізично відокремлений сервер (або локальний файл у випадку SQLite), який забезпечує надійне зберігання даних системи.
- Зовнішній поштовий сервер (Google SMTP): Сторонній вузол, з яким взаємодіє серверна частина для відправки транзакційних електронних листів (підтвердження реєстрації, сповіщення про зміну статусу) через протокол SMTP.

2 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ

2.1. Структура бази даних

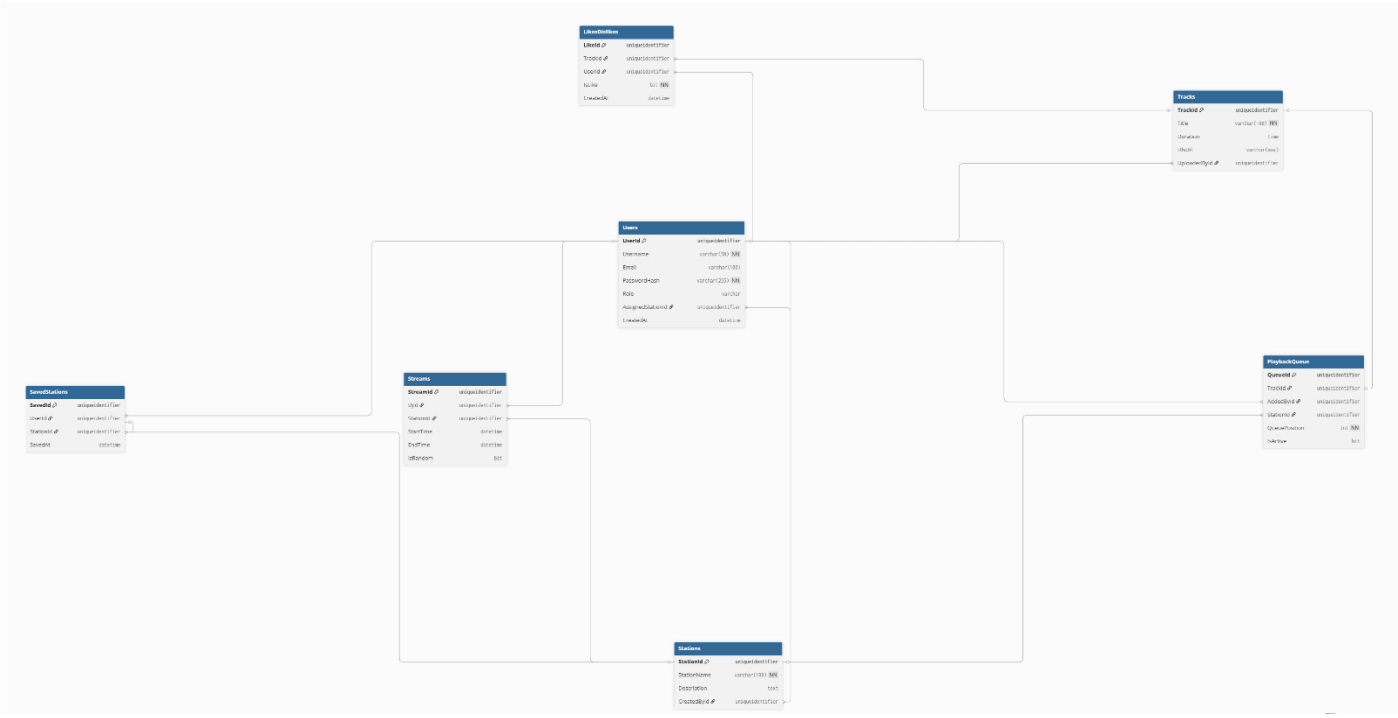

Рис. 2.1 – Проектування бази даних

1. User (Користувачі) Зберігає облікові дані всіх учасників системи.
   - UserId (PK, Guid): Унікальний ідентифікатор.
   - Username (string): Логін користувача.
   - Email (string): Електронна пошта.
   - PasswordHash (string): Хеш пароля (забезпечує безпеку).
   - Role (string): Роль користувача (Admin, Dj, User, Banned).
   - AssignedStationId (FK, Guid, Nullable): Ідентифікатор станції, до якої прив'язаний Діджей (якщо роль DJ).
   - CreatedAt (DateTime): Дата реєстрації.
2. RadioStation (Радіостанції) Інформація про станції, на яких ведеться мовлення.
   - StationId (PK, Guid): Унікальний ідентифікатор.
   - StationName (string): Назва станції.
   - Description (string): Опис формату або жанру.

- CreatedById (FK, Guid): Посилання на творця (Адміна).

3. Track (Музичні треки) Глобальна бібліотека аудіофайлів.

- TrackId (PK, Guid): Унікальний ідентифікатор.
- Title (string): Назва композиції.
- Duration (TimeSpan): Тривалість треку.
- UploadedById (FK, Guid): Діджей або Адмін, який завантажив файл.
- HlsUrl (string): Веб-посилання на master.m3u8 файл для потокового відтворення (результат конвертації).

4. PlaybackQueue (Черга відтворення) Визначає, які треки і в якому порядку грають на конкретній станції.

- QueueId (PK, Guid): Унікальний ідентифікатор запису в черзі.
- TrackId (FK, Guid): Посилання на трек.
- StationId (FK, Guid): Посилання на станцію.
- QueuePosition (int): Порядковий номер треку в плейлисті.
- IsActive (bool): Статус треку (true — грає в ефірі, false — тимчасово пропущений/скіпнутий).
- AddedById (FK, Guid): Хто додав трек у чергу.

5. Stream (Сесії мовлення) Журнал активності діджеїв (історія ефірів).

- StreamId (PK, Guid): Унікальний ідентифікатор сесії.
- StationId (FK, Guid): На якій станції йде ефір.
- DjId (FK, Guid): Який діджей веде ефір.
- StartTime (DateTime): Час початку.
- EndTime (DateTime, Nullable): Час завершення (якщо NULL — ефір триває).
- IsRandom (bool): Чи був увімкнений режим випадкового відтворення (Shuffle) під час цієї сесії.

6. SavedStation (Збережені станції) Реалізує функціонал "Улюблене" для слухачів.

- SavedId (PK, Guid): Ідентифікатор.
- UserId (FK, Guid): Слухач.
- StationId (FK, Guid): Обрана станція.

- SavedAt (DateTime): Дата додавання.

7. LikeDislike (Оцінки) Реалізує рейтинг треків.

- LikeId (PK, Guid): Ідентифікатор.
- UserId (FK, Guid): Хто оцінив.
- TrackId (FK, Guid): Який трек оцінено.
- IsLike (bool): Тип оцінки (true — Лайк, false — Дизлайк).
- CreatedAt (DateTime): Час оцінки.

2.2. Архітектура системи

2.2.1. Специфікація системи

Розроблена система "Online Radio Station" є веб-орієнтованим програмним комплексом, побудованим на базі платформи .NET (версія 9.0). Архітектура системи спроєктована за принципом багатошарової архітектури (Layered Architecture) з чітким розділенням відповідальності, що забезпечує гнучкість, тестованість та легкість супроводу коду. Для реалізації системи було використано наступний набір технологій та інструментів:

Платформа	.NET 9.
Мова програмування	C# 12.
Веб-фреймворк	ASP.NET Core MVC (Model-View-Controller).
Доступ до даних (ORM)	Entity Framework Core.
База даних	Microsoft SQL Server (з можливістю використання SQLite для локальної розробки та перенесення).
Обробка медіа	FFmpeg (зовнішня утиліта для транскодування аудіо у формат HLS).
Клієнтська частина	Razor Views (.cshtml), HTML5, CSS3 (Bootstrap 5), JavaScript.
Відтворення аудіо	Бібліотека hls.js для підтримки протоколу HTTP Live Streaming у браузері.
Сповіщення	SMTP-протокол (через System.Net.Mail) для відправки електронних листів.

Таблиця 2.1. Технологічний стек

Структура рішення

Програмний код організовано у вигляді рішення (Solution), що складається з чотирьох логічно ізольованих проєктів:

1. RadioStation.Domain (Ядро системи):
    - Містить основні сутності (User, Track, RadioStationEntity, PlaybackQueue).
    - Визначає інтерфейси (контракти) для патернів та репозиторіїв (IRepository, IAudioProcessor, IStreamFactory), забезпечуючи принцип інверсії залежностей (DIP).
    - Містить реалізацію бізнес-логіки патернів, що не залежать від зовнішньої інфраструктури (наприклад, BitrateStreamFactory).
2. RadioStation.Data (Шар даних):
    - Відповідає за взаємодію з базою даних.
    - Містить контекст бази даних ApplicationContext (наслідується від DbContext).
    - Реалізує інтерфейси репозиторіїв (TrackRepository, UserRepository), використовуючи Entity Framework Core для виконання SQL-запитів.
    - Включає механізм ініціалізації бази даних (DbInitializer) для автоматичного розгортання.
3. RadioStation.Services (Сервісний шар):
    - Виступає проміжною ланкою між контролерами та даними.
    - Містить бізнес-логіку, яка координує роботу репозиторіїв (наприклад, StationService для розрахунку часу ефіру, UserService для логіки бану та розсилки).
4. RadioStationSolution.WebApp (Шар представлення / Presentation Layer):
    - Головний виконуваний проєкт (ASP.NET Core Web Application).
    - Містить Контролери (DjController, AdminController), які приймають HTTP-запити, обробляють вхідні дані та викликають сервіси.

- Містить Представлення (Views), які відповідають за генерацію HTML-сторінок для користувача.
- Зберігає статичні файли (wwwroot) — стилі, скрипти та згенеровані HLS-потоки.

2.2.2. Вибір та обґрунтування патернів реалізації

Iterator (Ітератор)

Проблема: Необхідність послідовного обходу треків у плейлисті станції без розкриття внутрішньої структури колекції (яка може змінюватися).

Рішення: Патерн реалізовано для керування чергою відтворення на стороні клієнта (сторінка прослуховування).

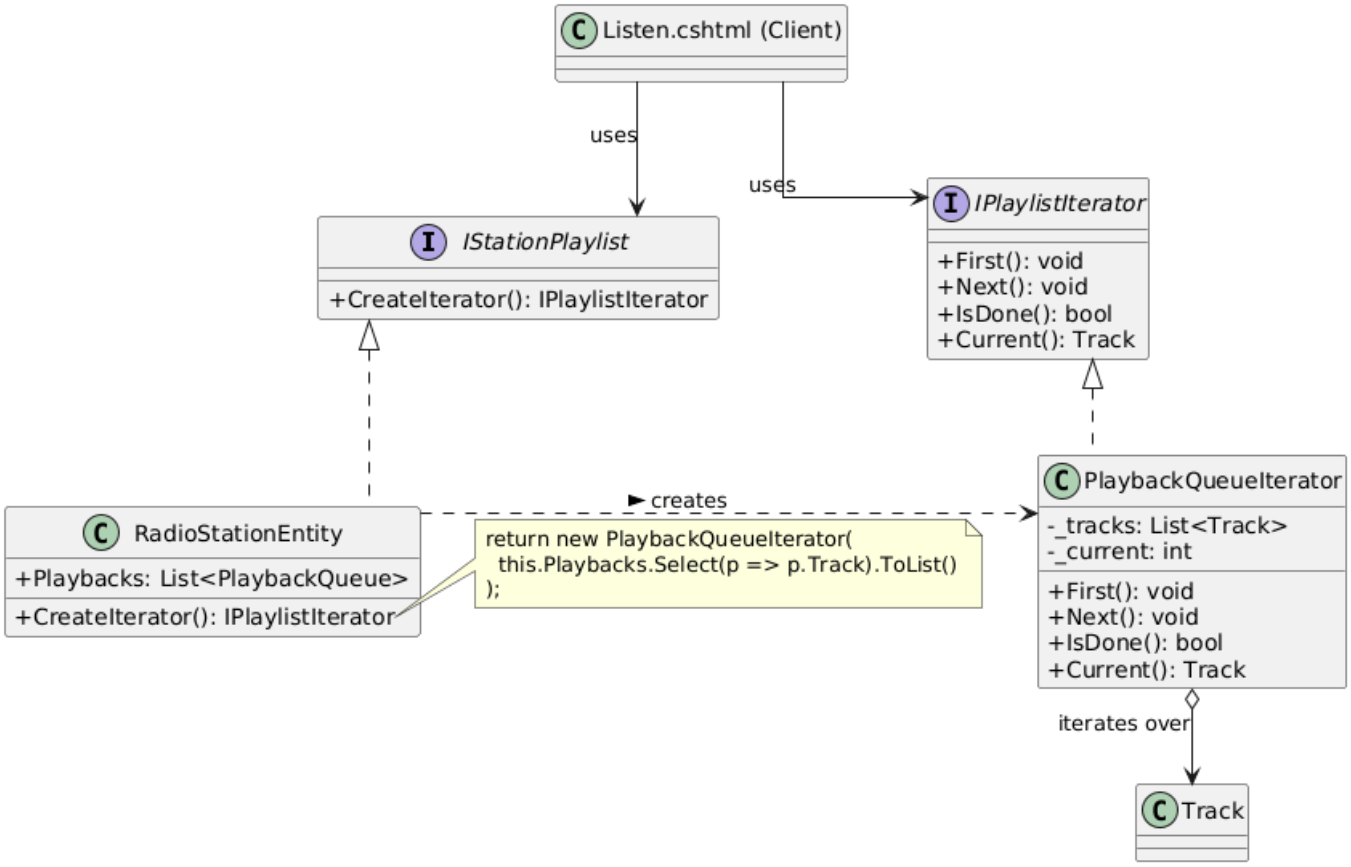

Рис. 2.2 - Структура патерну Iterator

IStationPlaylist — інтерфейс колекції, що декларує метод створення ітератора. Реалізується класом RadioStationEntity.

IPlaylistIterator — інтерфейс, що визначає методи обходу (First, Next, IsDone, Current).

PlaybackQueueIterator — конкретна реалізація, яка забезпечує прохід по списку треків. Перевага: Це дозволяє відокремити алгоритм обходу (наприклад, з урахуванням пропущених треків) від самої структури даних плейлиста.

Adapter (Адаптер)

Проблема: Необхідність використання зовнішньої консольної утиліти FFmpeg для транскодування аудіо, інтерфейс якої несумісний з об’єктно-орієнтованим кодом C#.

Рішення: Реалізовано клас FFmpegAdapter, який імплементує цільовий інтерфейс системи IAudioConverter.

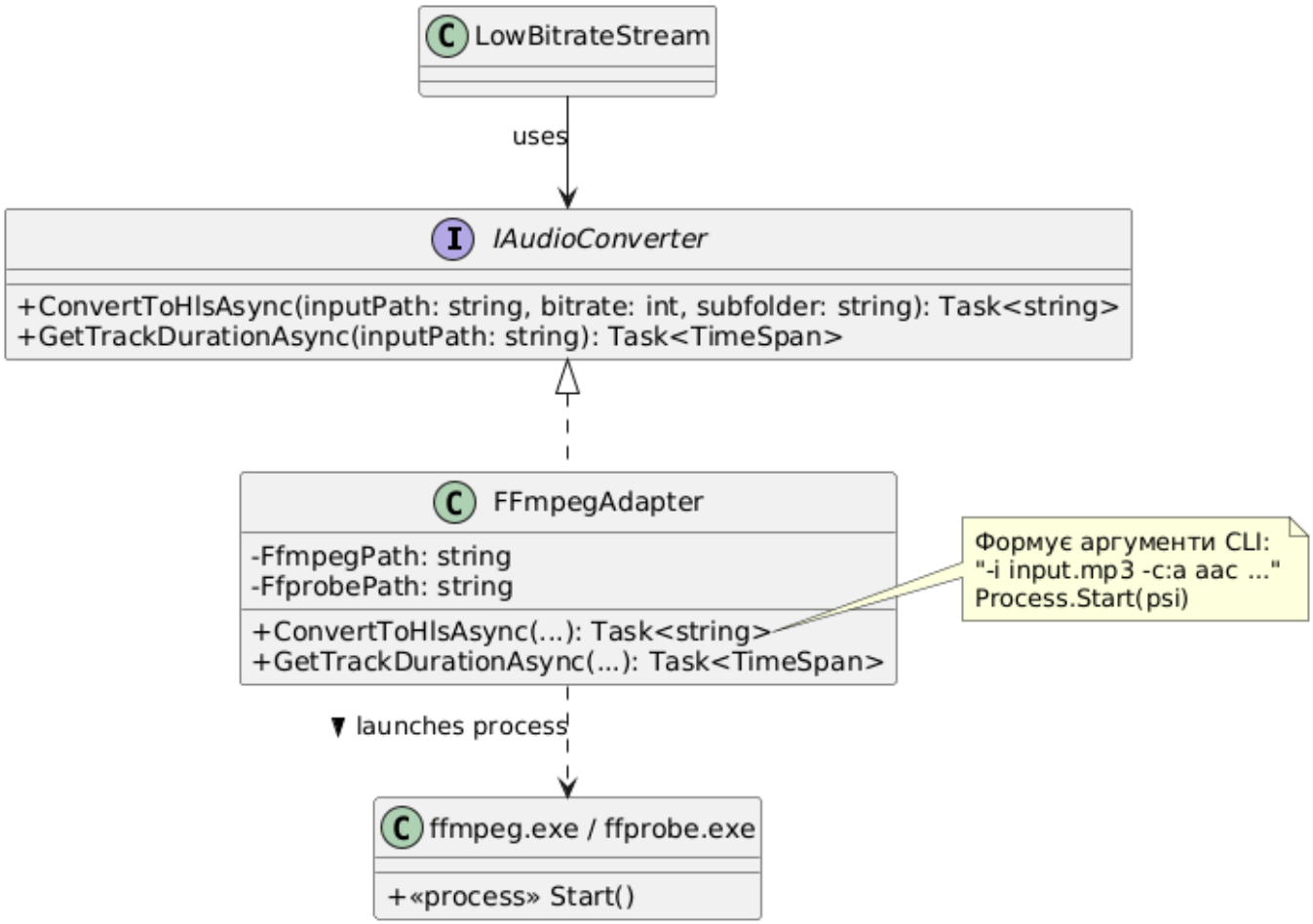

Рис. 2.3 - Структура патерну Adapter

У даному застосунку реалізовано структурний шаблон проектування Adapter, який дозволяє узгодити інтерфейс зовнішнього інструменту з вимогами внутрішнього коду.

Інтерфейс IAudioConverter виступає контрактом для перетворення аудіо — він визначає метод ConvertToHlsAsync(...), який має повертати шлях до згенерованого HLS-плейлиста.

Клас FFmpegAdapter реалізує цей інтерфейс і адаптує консольну утиліту ffmpeg.exe до внутрішнього контракту. Адаптер формує та запускає процес ffmpeg із потрібними параметрами, очікуючи завершення конвертації, після чого повертає результат.

Клас StreamingService виступає клієнтом, який працює лише з інтерфейсом IAudioConverter і не знає про існування ffmpeg.exe чи особливості його виклику.

Такий підхід забезпечує гнучкість і розширюваність системи — у майбутньому можна легко замінити FFmpegAdapter на іншу реалізацію (наприклад, LibAvAdapter), не змінюючи бізнес-логіку клієнта.

Factory Method (Фабричний метод)

Проблема: Система повинна підтримувати створення потоків різної якості (бітрейту), і логіка створення відповідного обробника може змінюватися.

Рішення: Використано для інкапсуляції логіки створення об'єктів стрімінгу.

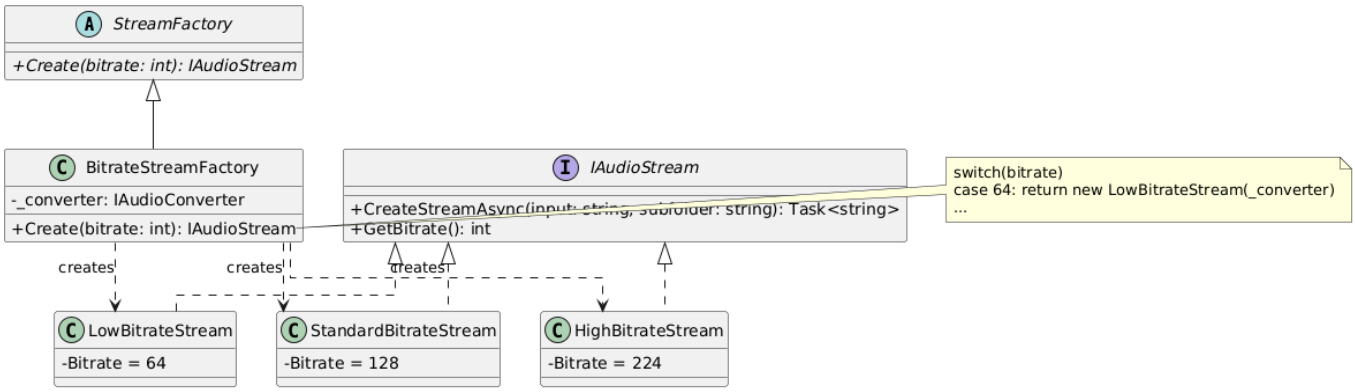

Рис. 2.4 - Структура патерну Factory Method

StreamFactory — абстрактний клас, що оголошує метод Create(int bitrate).

BitrateStreamFactory — конкретна реалізація, яка на основі вхідного бітрейту (64, 128, 224 kb/s) вирішує, який екземпляр класу створити: LowBitrateStream, StandardBitrateStream або HighBitrateStream. Перевага: Дозволяє легко додавати нові формати якості без зміни основного коду обробки.

Facade (Фасад)

Проблема: Процес завантаження нового треку є складним і вимагає взаємодії багатьох компонентів: конвертації файлу, збереження метаданих, розрахунку позиції

в черзі та запису в базу даних. Рішення: Реалізовано клас AudioProcessingFacade, який надає простий інтерфейс (ProcessNewTrackAsync) для контролера.

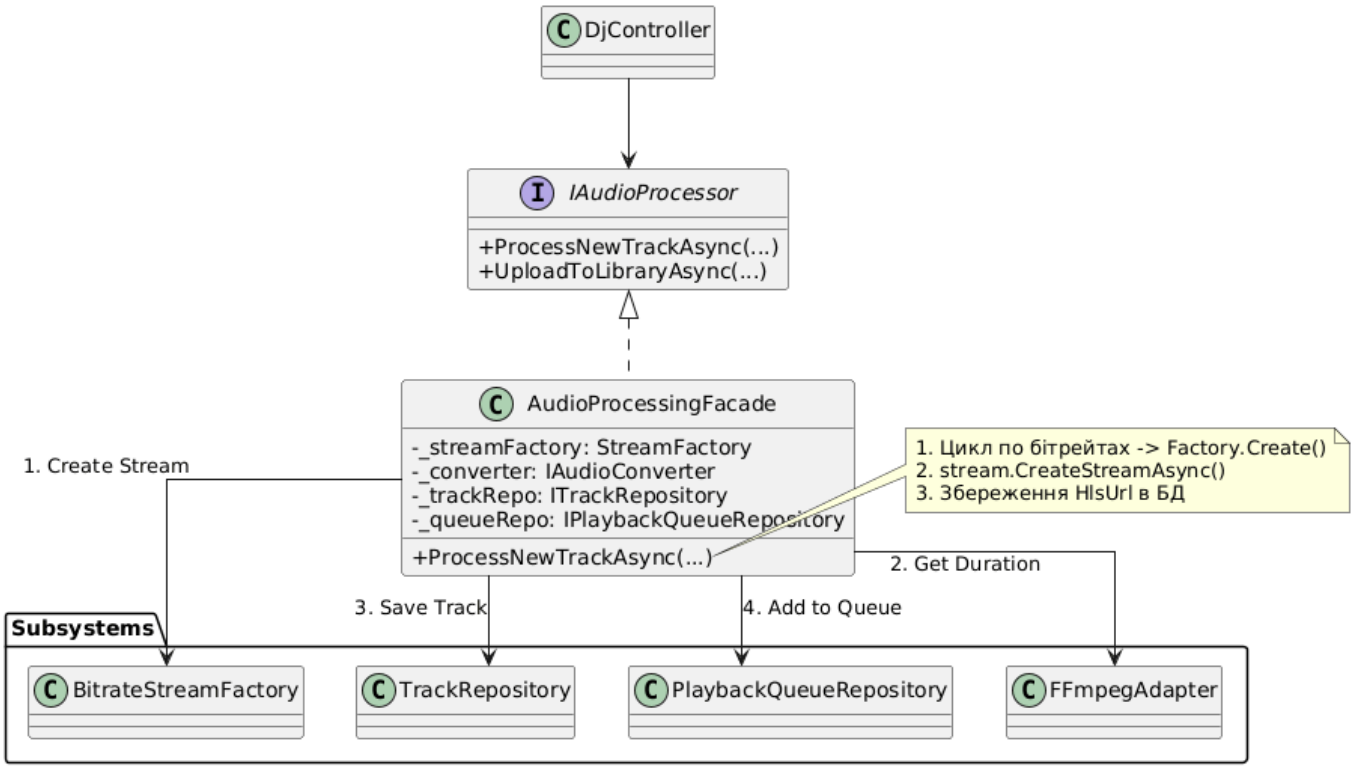

Рис. 2.5 - Структура патерну Facade

Фасад приховує складну взаємодію між Фабрикою (створення стратегії конвертації), Адаптером (фізична конвертація) та Репозиторіями (збереження сутностей Track та PlaybackQueue).

DjController викликає лише один метод фасаду, не вдаючись у деталі реалізації бізнес-процесу.

Visitor (Відвідувач)

Проблема: Необхідність виконання операцій аналізу та збору статистики над об'єктами різнорідної структури (Треки, Стріми, Елементи черги) без зміни класів цих об'єктів.

Рішення: Використано для генерації звітів на сторінці адміністратора.

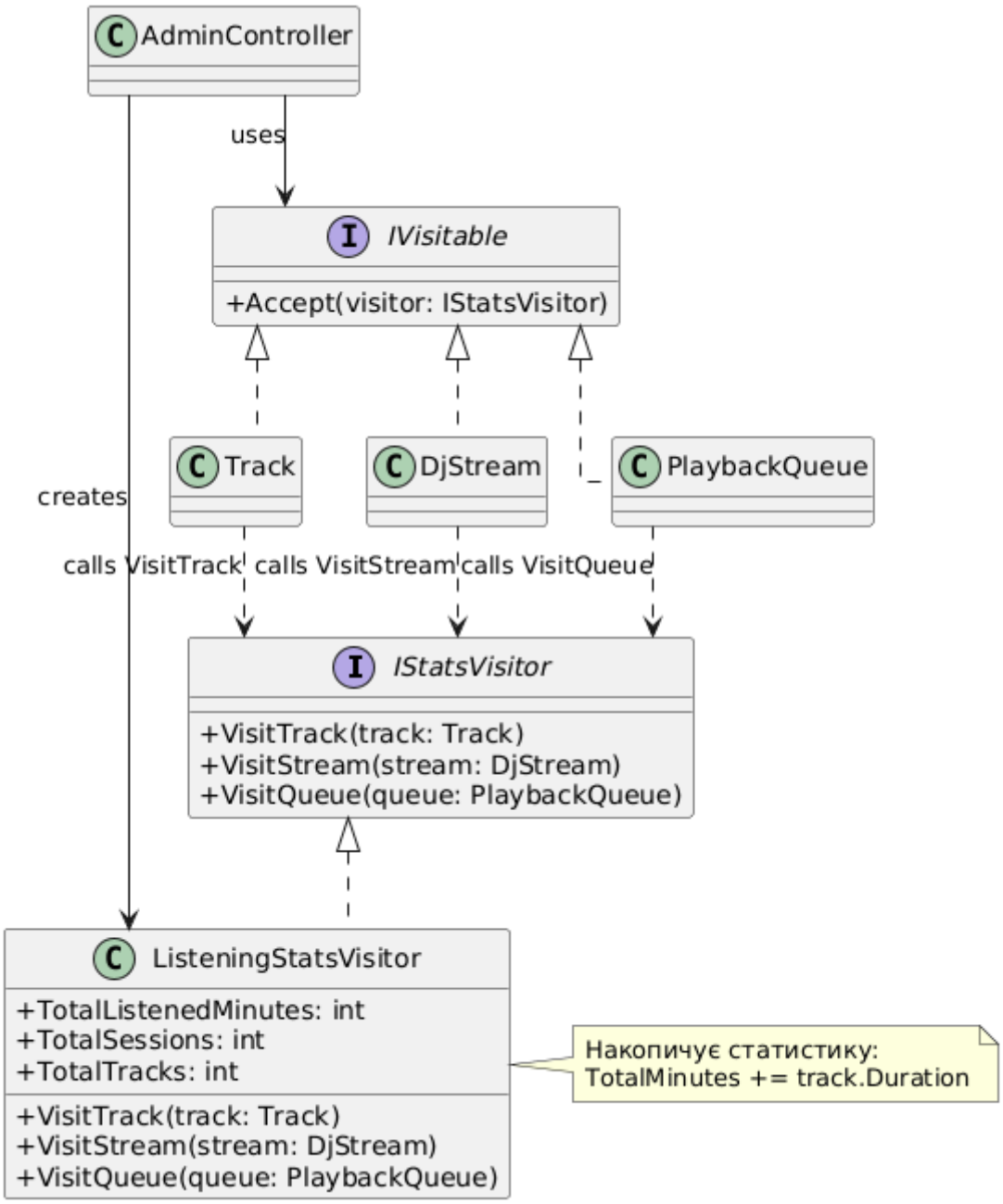

Рис. 2.6 - Структура патерну Відвідувач у системі "Online Radio Station"

- IStatsVisitor — інтерфейс відвідувача, що оголошує методи VisitTrack, VisitStream тощо.
- ListeningStatsVisitor — конкретна реалізація, яка містить алгоритм підрахунку загальної тривалості ефіру та кількості сесій.
- Сутності (Track, DjStream) реалізують метод Accept, передаючи себе відвідувачу. Перевага: Дозволяє додавати нові види звітів (наприклад, експорт у XML), просто створюючи нового Відвідувача, не змінюючи код сутностей.

2.3. Інструкція користувача

Інструкція описує порядок роботи з веб-застосунком "Online Radio Station" для трьох основних ролей користувачів: Слухача, Діджея та Адміністратора.

2.3.1. Початок роботи (для всіх користувачів)

1. Головна сторінка: При вході на сайт користувач потрапляє на головну сторінку-вітрину, де відображається перелік доступних радіостанцій та загальна інформація про сервіс. Для доступу до функціоналу необхідно авторизуватися.
2. Реєстрація та Авторизація:
    - Натисніть кнопку "Реєстрація", якщо у вас немає акаунту. Заповніть поля: Логін, Email, Пароль. Система автоматично хешує пароль для безпеки. Після успішної реєстрації на вказану електронну пошту автоматично надсилається вітальний лист із підтвердженням створення облікового запису.
    - Якщо акаунт вже існує, натисніть "Вхід", введіть облікові дані.
    - Для зручності вводу пароля доступна кнопка "Показати пароль" (іконка ока).
3. Особистий кабінет (Dashboard): Після успішного входу система автоматично перенаправляє користувача на відповідну панель керування залежно від його ролі. Доступ до кабінету завжди можна отримати через кнопку "Кабінет" у верхньому меню.

2.3.2. Інструкція Слухача

Основна мета слухача — пошук музики та прослуховування ефірів.

1. Вибір станції: На сторінці "Моя сторінка" (UserDashboard) відображаються картки всіх радіостанцій.
    - Обране: Натисніть іконку "Сердечко" на картці станції, щоб додати її до списку улюблених. Улюблені станції автоматично закріплюються у верхній частині списку.
2. Прослуховування ефіру: Натисніть кнопку "Слухати" на картці станції. Відкриється сторінка плеєра.

- Статус ефіру: Якщо діджей не веде трансляцію, з'явиться повідомлення "Ефір завершено". Якщо ефір активний, з'явиться кнопка "LIVE" або почнеться автоматичне відтворення.
- Плеєр: Інтерфейс відображає назву поточного треку та статус. Прогрес-бар відсутній, оскільки відтворення відбувається в режимі реального часу (синхронізовано з сервером).
- Оцінка треків: Використовуйте кнопки "Лайк" (палець вгору) або "Дизлайк" (палець вниз), щоб оцінити поточну композицію.

2.3.3. Інструкція Діджея

Діджей відповідає за контент та проведення прямих ефірів.

Важливо: Перед початком роботи Діджей повинен бути прив'язаний до конкретної станції Адміністратором.

1. Панель Діджея (DJ Dashboard): Містить доступ до трьох основних інструментів: "Ефірний пульт", "Плейлист" та "Бібліотека".
2. Керування бібліотекою та плейлистом:
   - Перейдіть у розділ "Бібліотека", щоб переглянути всі доступні треки в системі. Натисніть кнопку "+ Додати", щоб включити трек у свій ефір.
   - Перейдіть у розділ "Мій плейлист". Тут можна завантажити нові файли (кнопка "Завантажити новий трек"). При завантаженні система автоматично конвертує MP3 у формат HLS (мульти-бітрейт).
   - У плейлисті можна видаляти треки з ефіру (кнопка "Прибрати"). Це не видаляє файл фізично, а лише прибирає його з черги.
3. Проведення ефіру (Стрім): Перейдіть у розділ "Управління ефіром".
   - Старт: Оберіть режим відтворення (звичайний або "Shuffle" — випадковий) та натисніть "ПОЧАТИ ЕФІР". Статус зміниться на "ON AIR", запуститься таймер сесії.
   - Моніторинг: Панель "Студійний монітор" показує, який трек зараз чують слухачі.

- Керування чергою (на льоту): Під час ефіру ви можете використовувати кнопки "Скіпнути" (м'який пропуск треку) або "Відновити". Також доступна зміна порядку треків (вгору/вниз), якщо це не суперечить логіці поточного відтворення.
- Завершення: Натисніть "ЗАВЕРШИТИ ЕФІР", щоб зупинити мовлення для всіх слухачів.

2.3.4. Інструкція Адміністратора

Адміністратор має повний контроль над системою.

1. Керування станціями: У розділі "Керування станціями" можна створювати нові радіостанції, редагувати їх назву та опис або видаляти їх.
2. Керування користувачами:
   - Перейдіть у розділ "Керування користувачами".
   - Щоб призначити Діджея: натисніть "Редагувати" на користувачеві, змініть роль на "Dj" та оберіть станцію зі списку.
   - Щоб заблокувати порушника: змініть роль на "Banned". Заблокований користувач не зможе увійти в систему.
   - При зміні ролі користувача або його блокуванні система автоматично надсилає йому відповідне сповіщення на електронну пошту.
3. Глобальна бібліотека: Дозволяє переглядати всі завантажені треки в системі, бачити їх авторів та рейтинг (кількість лайків/дизлайків). Тут доступна функція повного видалення треку з бази даних та всіх плейлистів.
4. Статистика та обслуговування: Розділ "Статистика" відображає аналітичні дані (загальна тривалість контенту, кількість треків, історія сесій). Кнопка "Очистити файлове сміття" дозволяє видалити з диска фізичні файли, які більше не використовуються в базі даних.

ВИСНОВКИ

У ході виконання курсової роботи було успішно спроєктовано та розроблено програмну систему «Online Radio Station» — веб-орієнтований комплекс для організації професійного потокового аудіомовлення. На основі проведеного аналізу предметної області та існуючих аналогів було сформульовано функціональні вимоги та обрано клієнт-серверну архітектуру як основу системи. Реалізація серверної частини на базі платформи .NET 9 (ASP.NET Core MVC) забезпечила високу продуктивність та надійність обробки запитів, а використання Entity Framework Core дозволило організувати гнучкий шар доступу до даних, сумісний як з Microsoft SQL Server, так і з SQLite.

Ключовим досягненням роботи стала практична реалізація та інтеграція класичних патернів проєктування (GoF), що дозволило вирішити складні архітектурні задачі. Зокрема, патерн Adapter забезпечив безшовну інтеграцію зовнішньої утиліти FFmpeg, що уможливило реалізацію професійного стандарту мовлення HLS (HTTP Live Streaming). Використання Factory Method надало гнучкий механізм створення потоків із різним бітрейтом, забезпечуючи адаптивність мовлення, тоді як Facade значно спростив взаємодію контролерів зі складною підсистемою обробки медіа-контенту. Для коректного послідовного обходу треків під час відтворення було застосовано патерн Iterator, а патерн Visitor забезпечив можливість розширення функціоналу збору статистики без зміни структури класів даних.

Розроблена система надає повний набір інструментів для всіх категорій користувачів. Слухачі отримали зручний інтерфейс для прослуховування синхронізованого ефіру, можливість формувати список улюблених станцій та оцінювати треки. Діджеї забезпечені потужним «Ефірним пультом» для керування трансляцією в реальному часі, інструментами для завантаження контенту та керування чергою відтворення. Адміністратори мають повний контроль над користувачами, станціями та глобальною бібліотекою контенту.

Особливу увагу в роботі було приділено надійності та безпеці: реалізовано механізм хешування паролів, систему рольового доступу, захист від «фантомних» файлів та автоматичне сповіщення користувачів через Email. Таким чином, мета курсової роботи досягнута в повному обсязі. Отриманий програмний продукт є повнофункціональною, архітектурно досконалою системою, яка готова до подальшого масштабування та впровадження.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Visual Studio Code Documentation. Documentation for Visual Studio Code [Електронний ресурс]. – Режим доступу: https://code.visualstudio.com/docs.
2. SQL Server technical documentation. SQL Server Documentation [Електронний ресурс] // Microsoft Learn. – Режим доступу: https://learn.microsoft.com/en-us/sql/sql-server/.
3. ASP.NET Core Documentation. Documentation for ASP.NET Core [Електронний ресурс] // Microsoft Learn. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/.
4. Entity Framework Core Documentation. Documentation for Entity Framework Core [Електронний ресурс] // Microsoft Learn. – Режим доступу: https://learn.microsoft.com/en-us/ef/core/.
5. SQLite Documentation. Official Documentation for SQLite [Електронний ресурс]. – Режим доступу: https://www.sqlite.org/docs.html.
6. Troelsen A., Japikse P. C# 7.0 and .NET Core 2.0 – with Visual Studio 2017. – Apress, 2017. – 1372 p. [Електронний ресурс]. – Режим доступу: https://dl.ebooksworld.ir/motoman/Apress.Pro.Csharp.7.With.NET.and.NET.Core.www.EBooksWorld.ir.pdf .
7. Refactoring.Guru. Design Patterns [Електронний ресурс]. – Режим доступу: https://refactoring.guru/uk/design-patterns .
8. Fowler M. Patterns of Enterprise Application Architecture. – Addison-Wesley Professional, 2002. – 442 p. [Електронний ресурс]. – Режим доступу: https://dl.ebooksworld.ir/motoman/Patterns%20of%20Enterprise%20Application%20Architecture.pdf.
9. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – Addison-Wesley Professional, 1994. – 395 p. [Електронний ресурс]. – Режим доступу: https://www.javier8a.com/itc/bd1/articulo.pdf .

10. FFmpeg Documentation. Official Documentation for FFmpeg [Електронний ресурс]. – Режим доступу: https://ffmpeg.org/documentation.html.
11. HTTP Live Streaming (HLS) Authoring Specification for Apple Devices. [Електронний ресурс] // Apple Developer. – Режим доступу: https://developer.apple.com/documentation/http_live_streaming.

ДОДАТКИ

Додаток А

Проектування паттернів

1. Програмна реалізація патерну Adapter (демонструє адаптацію FFmpeg)

Інтерфейс IAudioConverter.cs:

```
using System.Threading.Tasks;

namespace OnlineRadioStation.Domain
{
    public interface IAudioConverter
    {
        Task<string> ConvertToHlsAsync(string inputPath, int bitrate, string
subfolder);
        Task<TimeSpan> GetTrackDurationAsync(string inputPath);

    }
}
```

Клас FFmpegAdapter.cs:

```
using System;
using System.Diagnostics;
using System.Globalization;
using System.IO;
using System.Text;
using System.Threading.Tasks;

namespace OnlineRadioStation.Domain
{
    public class FFmpegAdapter : IAudioConverter
    {
        private const string FffmpegPath = "wwwroot/ffmpeg/ffmpeg.exe";
        private const string FfprobePath = "wwwroot/ffmpeg/ffprobe.exe";

        public async Task<string> ConvertToHlsAsync(string inputPath, int bitrate,
string subfolder)
        {
            var baseFolder = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot",
"streams", Path.GetFileNameWithoutExtension(inputPath));
            var outputFolder = Path.Combine(baseFolder, subfolder);
            Directory.CreateDirectory(outputFolder);

            var playlistPath = Path.Combine(outputFolder, "index.m3u8");
            var segmentPath = Path.Combine(outputFolder, "seg%03d.ts");

            var args = $"-i \"{inputPath}\" " +
```

```
                    $"-c:a aac -b:a {bitrate}k " +
                    $"-f hls -hls_time 10 -hls_list_size 0 " +
                    $"-hls_segment_filename \"{segmentPath}\" " +
                    $"\"{playlistPath}\"";

            var psi = new ProcessStartInfo
            {
                FileName = FffmpegPath,
                Arguments = args,
                UseShellExecute = false,
                RedirectStandardOutput = true,
                RedirectStandardError = true,
                CreateNoWindow = true
            };

            var outputBuilder = new StringBuilder();
            var errorBuilder = new StringBuilder();

            using var process = new Process { StartInfo = psi };

            process.OutputDataReceived += (sender, e) => { if (e.Data != null)
outputBuilder.AppendLine(e.Data); };
            process.ErrorDataReceived += (sender, e) => { if (e.Data != null)
errorBuilder.AppendLine(e.Data); };

            process.Start();
            process.BeginOutputReadLine();
            process.BeginErrorReadLine();
            await process.WaitForExitAsync();

            if (process.ExitCode != 0)
            {
                throw new Exception($"FFmpeg failed: {errorBuilder}");
            }

            var webRootPath = Path.Combine(Directory.GetCurrentDirectory(),
"wwwroot");
            return "/" + Path.GetRelativePath(webRootPath,
playlistPath).Replace("\\", "/");
        }

        public async Task<TimeSpan> GetTrackDurationAsync(string inputPath)
{
    if (!File.Exists(FfprobePath))
    {
        Console.WriteLine($"[Adapter ERROR] ffprobe.exe НЕ ЗНАЙДЕНО за шляхом:
{FfprobePath}");
        return TimeSpan.Zero;
    }
```

```
    var args = $"-v error -show_entries format=duration -of
default=noprint_wrappers=1:nokey=1 \"{inputPath}\"";

    var psi = new ProcessStartInfo
    {
        FileName = FfprobePath,
        Arguments = args,
        UseShellExecute = false,
        RedirectStandardOutput = true,
        RedirectStandardError = true,
        CreateNoWindow = true
    };

    using var process = Process.Start(psi);
    if (process == null) return TimeSpan.Zero;

    var output = await process.StandardOutput.ReadToEndAsync();
    var error = await process.StandardError.ReadToEndAsync();
    await process.WaitForExitAsync();

    Console.WriteLine($"[Adapter] FFprobe Raw Output: '{output.Trim()}'");

    if (!string.IsNullOrEmpty(error))
    {
        Console.WriteLine($"[Adapter ERROR] FFprobe Error: {error}");
    }

    if (double.TryParse(output.Trim(), System.Globalization.NumberStyles.Any,
System.Globalization.CultureInfo.InvariantCulture, out double seconds))
    {
        var duration = TimeSpan.FromSeconds(seconds);
        Console.WriteLine($"[Adapter] Duration parsed: {duration}");
        return duration;
    }

    Console.WriteLine("[Adapter ERROR] Не вдалося розпарсити тривалість.");
    return TimeSpan.Zero;
}
    }
}
```

2. Програмна реалізація патерну Factory Method (демонструє створення стрімів різної якості)

   Абстрактна фабрика StreamFactory.cs:

```
namespace OnlineRadioStation.Domain
{
```

```
    public abstract class StreamFactory
    {
        public abstract IAudioStream Create(int bitrate);
    }
}
```

Конкретна фабрика BitrateStreamFactory.cs:

```
namespace OnlineRadioStation.Domain
{
    public class BitrateStreamFactory : StreamFactory
    {
        private readonly IAudioConverter _converter;

        public BitrateStreamFactory(IAudioConverter converter)
        {
            _converter = converter;
        }

        public override IAudioStream Create(int bitrate)
        {
            return bitrate switch
            {
                <= 64 => new LowBitrateStream(_converter),
                <= 128 => new StandardBitrateStream(_converter),
                _ => new HighBitrateStream(_converter)
            };
        }
    }
}

using System.Threading.Tasks;
```

Приклад продукту LowBitrateStream.cs:

```
namespace OnlineRadioStation.Domain
{
    public class LowBitrateStream : IAudioStream
    {
        private readonly IAudioConverter _converter;
        private const int Bitrate = 64;

        public LowBitrateStream(IAudioConverter converter)
        {
            _converter = converter;
        }

        public int GetBitrate() => Bitrate;

        public async Task<string> CreateStreamAsync(string inputAudioPath, string
subfolder)
```

```
        {
            return await _converter.ConvertToHlsAsync(inputAudioPath, Bitrate,
subfolder);
        }
    }
}
```

3. Програмна реалізація патерну Facade (демонструє головний мозок обробки файлів).

Інтерфейс IAudioProcessor.cs:

```
using System;
using System.Threading.Tasks;

namespace OnlineRadioStation.Domain
{
    public interface IAudioProcessor
    {
        Task ProcessNewTrackAsync(string tempFilePath, string title, Guid stationId,
Guid djId);
        Task UploadToLibraryAsync(string tempFilePath, string title, Guid adminId);
    }
}
```

Клас AudioProcessingFacade.cs:

```
using System;
using System.IO;
using System.Linq;
using System.Threading.Tasks;

namespace OnlineRadioStation.Domain
{
    public class AudioProcessingFacade : IAudioProcessor
    {
        private readonly IAudioConverter _converter;
        private readonly ITrackRepository _trackRepository;
        private readonly IPlaybackQueueRepository _queueRepository;
        private readonly StreamFactory _streamFactory;

        public AudioProcessingFacade(
            IAudioConverter converter,
            ITrackRepository trackRepository,
            IPlaybackQueueRepository queueRepository,
            StreamFactory streamFactory)
        {
            _converter = converter;
            _trackRepository = trackRepository;
            _queueRepository = queueRepository;
            _streamFactory = streamFactory;
```

```
        }

        private async Task<string> ConvertFileInternal(string tempFilePath)
        {
            int[] targetBitrates = { 64, 92, 128, 196, 224 };
            var masterPlaylistContent = "#EXTM3U\n#EXT-X-VERSION:3\n";
            var baseFileName = Path.GetFileNameWithoutExtension(tempFilePath);
            var baseFolder = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot",
"streams", baseFileName);
            Directory.CreateDirectory(baseFolder);

            foreach (var bitrate in targetBitrates)
            {
                var streamProduct = _streamFactory.Create(bitrate);
                await streamProduct.CreateStreamAsync(tempFilePath,
bitrate.ToString());
                masterPlaylistContent += $"#EXT-X-STREAM-INF:BANDWIDTH={bitrate *
1000},CODECS=\"mp4a.40.2\"\n";
                masterPlaylistContent += $"{bitrate}/index.m3u8\n";
            }

            var masterPath = Path.Combine(baseFolder, "master.m3u8");
            await File.WriteAllTextAsync(masterPath, masterPlaylistContent);
            return $"/streams/{baseFileName}/master.m3u8";
        }

        public async Task ProcessNewTrackAsync(string tempFilePath, string title,
Guid stationId, Guid djId)
        {
            var finalHlsUrl = await ConvertFileInternal(tempFilePath);
            var realDuration = await _converter.GetTrackDurationAsync(tempFilePath);

            var newTrack = new Track
            {
                TrackId = Guid.NewGuid(),
                Title = title,
                Duration = realDuration,
                UploadedById = djId,
                HlsUrl = finalHlsUrl
            };

            _trackRepository.AddEntity(newTrack);

            var nextPosition = _queueRepository.GetAll().Count(q => q.StationId ==
stationId) + 1;
            var newQueueEntry = new PlaybackQueue
            {
                QueueId = Guid.NewGuid(),
                TrackId = newTrack.TrackId,
```

```
                StationId = stationId,
                AddedById = djId,
                QueuePosition = nextPosition
            };

            _queueRepository.AddEntity(newQueueEntry);
            await _trackRepository.SaveChangesAsync();
        }

        public async Task UploadToLibraryAsync(string tempFilePath, string title,
Guid adminId)
        {
            var finalHlsUrl = await ConvertFileInternal(tempFilePath);
            var realDuration = await _converter.GetTrackDurationAsync(tempFilePath);

            var newTrack = new Track
            {
                TrackId = Guid.NewGuid(),
                Title = title,
                Duration = realDuration,
                UploadedById = adminId,
                HlsUrl = finalHlsUrl
            };

            _trackRepository.AddEntity(newTrack);
            await _trackRepository.SaveChangesAsync();
        }
    }
}
```

4. Програмна реалізація патерну Iterator (демонструє обхід плейлиста).

Інтерфейс IStationPlaylist.cs

```
namespace OnlineRadioStation.Domain
{
    public interface IStationPlaylist
    {
        IPlaylistIterator CreateIterator();
    }
}
```

Інтерфейс IPlaylistIterator.cs

```
namespace OnlineRadioStation.Domain
{
    public interface IPlaylistIterator
    {
        void First();
        void Next();
        bool IsDone();
        Track Current();
```

```
    }
}
```

Реалізація в сутності RadioStationEntity.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;

namespace OnlineRadioStation.Domain
{
    public class RadioStationEntity : IStationPlaylist
    {
        [Key]
        public Guid StationId { get; set; }
        public string StationName { get; set; } = string.Empty;
        public string Description { get; set; } = string.Empty;
        public Guid CreatedById { get; set; }
        public User CreatedBy { get; set; } = null!;
        public ICollection<PlaybackQueue> Playbacks { get; set; } = new
List<PlaybackQueue>();
        public ICollection<SavedStation> SavedByUsers { get; set; } = new
List<SavedStation>();
        public ICollection<DjStream> DjStreams { get; set; } = new List<DjStream>();

        public IPlaylistIterator CreateIterator()
        {
            var tracks = this.Playbacks
                .OrderBy(p => p.QueuePosition)
                .Select(p => p.Track)
                .ToList();
            return new PlaybackQueueIterator(tracks);
        }
    }
}
```

Конкретний ітератор PlaybackQueueIterator.cs

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace OnlineRadioStation.Domain
{
    public class PlaybackQueueIterator : IPlaylistIterator
    {
        private readonly List<Track> _tracks;
        private int _currentIndex = 0;

        public PlaybackQueueIterator(List<Track> tracks)
```

```
        {
            _tracks = tracks;
        }

        public void First() => _currentIndex = 0;

        public void Next() => _currentIndex++;

        public bool IsDone() => _currentIndex >= _tracks.Count;

        public Track Current()
        {
            if (IsDone())
                throw new InvalidOperationException("Кінець плейлиста");
            return _tracks[_currentIndex];
        }
    }
}
```

5. Програмна реалізація патерну Visitor (демонструє збір статистики).

Інтерфейс IStatsVisitor.cs

```
{
    public interface IStatsVisitor
    {
        void VisitTrack(Track track);
        void VisitStream(DjStream stream);
        void VisitQueue(PlaybackQueue queue);
    }
}
```

Клас ListeningStatsVisitor.cs

```
namespace OnlineRadioStation.Domain
{
    public class ListeningStatsVisitor : IStatsVisitor
    {
        public int TotalListenedMinutes { get; private set; } = 0;
        public int TotalSessions { get; private set; } = 0;
        public int TotalTracks { get; private set; } = 0;

        public void VisitTrack(Track track)
        {
            TotalListenedMinutes += (int)track.Duration.TotalMinutes;
            Console.WriteLine($"[Stats] Тривалість треку {track.Title}:
{track.Duration.TotalMinutes} хв");
        }

        public void VisitStream(DjStream stream)
        {
            TotalSessions += 1;
```

```
            Console.WriteLine($"[Stats] Сесія стріму {stream.StreamId}:
{stream.EndTime - stream.StartTime}");
        }

        public void VisitQueue(PlaybackQueue queue)
        {
            TotalTracks += 1;
            Console.WriteLine($"[Stats] Черга {queue.QueueId}: {queue.QueuePosition}
позиція");
        }
    }
}
```

Метод ShowStats де створюється visitor і викликається Accept у контролері AdminController.cs

```
[HttpGet]
        public async Task<IActionResult> ShowStats()
        {
            var visitor = new ListeningStatsVisitor();
            var allTracks = await _trackRepo.GetAll().ToListAsync();
            var allStreams = await _streamRepo.GetAll().ToListAsync();
            var allQueueItems = await _queueRepo.GetAll().ToListAsync();
            foreach (var track in allTracks) track.Accept(visitor);
            foreach (var stream in allStreams) stream.Accept(visitor);
            foreach (var item in allQueueItems) item.Accept(visitor);
            return View(visitor);
        }
```

Додаток Б

Конфігурація бази даних та ініціалізація

1. Контекст даних ApplicationContext.cs

```
using Microsoft.EntityFrameworkCore;
using OnlineRadioStation.Domain;

namespace OnlineRadioStation.Data
{
    public class ApplicationContext : DbContext
    {
        public DbSet<User> Users { get; set; }
        public DbSet<RadioStationEntity> Stations { get; set; }
        public DbSet<Track> Tracks { get; set; }
        public DbSet<LikeDislike> LikesDislikes { get; set; }
        public DbSet<DjStream> Streams { get; set; }
        public DbSet<SavedStation> SavedStations { get; set; }
        public DbSet<PlaybackQueue> PlaybackQueue { get; set; }

        public ApplicationContext(DbContextOptions<ApplicationContext> options)
            : base(options)
```

```
        {
        }
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            modelBuilder.Entity<User>().ToTable("Users");
            modelBuilder.Entity<RadioStationEntity>().ToTable("Stations");
            modelBuilder.Entity<Track>().ToTable("Tracks");
            modelBuilder.Entity<LikeDislike>().ToTable("LikesDislikes");
            modelBuilder.Entity<DjStream>().ToTable("Streams");
            modelBuilder.Entity<SavedStation>().ToTable("SavedStations");
            modelBuilder.Entity<PlaybackQueue>().ToTable("PlaybackQueue");
        }
    }
}
```

2. Ініціалізатор DbInitializer.cs (код, де створюються дефолтні адмін та станції).

```
using BCrypt.Net;
using OnlineRadioStation.Domain;
using System;
using System.Linq;

namespace OnlineRadioStation.Data
{
    public static class DbInitializer
    {
        public static void Initialize(ApplicationContext context)
        {

            context.Database.EnsureCreated();

            if (context.Users.Any())
            {
                return;
            }
            var admin = new User
            {
                UserId = Guid.NewGuid(),
                Username = "admin",
                Email = "admin@radio.com",
                PasswordHash = BCrypt.Net.BCrypt.HashPassword("admin"),
                Role = "Admin",
                CreatedAt = DateTime.UtcNow
            };
            context.Users.Add(admin);
```

```
            var dj = new User
            {
                UserId = Guid.NewGuid(),
                Username = "dj",
                Email = "dj@radio.com",
                PasswordHash = BCrypt.Net.BCrypt.HashPassword("dj"),
                Role = "Dj",
                CreatedAt = DateTime.UtcNow
            };
            context.Users.Add(dj);

            var user = new User
            {
                UserId = Guid.NewGuid(),
                Username = "user",
                Email = "user@radio.com",
                PasswordHash = BCrypt.Net.BCrypt.HashPassword("user"),
                Role = "User",
                CreatedAt = DateTime.UtcNow
            };
            context.Users.Add(user);

            var roks = new RadioStationEntity
            {
                StationId = Guid.NewGuid(),
                StationName = "Radio ROKS",
                Description = "Тільки рок! Найкращі хіти.",
                CreatedById = admin.UserId
            };
            context.Stations.Add(roks);

            var hitfm = new RadioStationEntity
            {
                StationId = Guid.NewGuid(),
                StationName = "Hit FM",
                Description = "Тільки хіти 90-х та сучасності.",
                CreatedById = admin.UserId
            };
            context.Stations.Add(hitfm);

            dj.AssignedStationId = roks.StationId;

            context.SaveChanges();
        }
    }
}
```