Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА МУЗИЧНОГО ВЕБЗАСТОСУНКУ, ЩО ДАЄ МОЖЛИВІСТЬ ВИВЧЕННЯ ІНОЗЕМНИХ МОВ ЗА ДОПОМОГОЮ ШТУЧНОГО ІНТЕЛЕКТУ НА ОСНОВІ ТЕКСТІВ ПІСЕНЬ
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-21-2
Юдаков Н. М.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Руденко Д. О.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Юдакову Нікіті Михайловичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка музичного вебзастосунку, що дає можливість вивчення іноземних мов за допомогою штучного інтелекту на основі текстів пісень

затверджена наказом університету від 19 травня 2025 року № 381Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 28 травня 2025 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, фреймворк з відкритим кодом FastAPI.

4. Перелік питань, що потрібно опрацювати в роботі
1. Теоритичні основи створення музичного вебзастосунку.
2. Проєктування музичного вебзастосунку.
3. Практична реалізація застосунку.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми взаємодії користувача з музичним і текстовим контентом з метою вивчення мови через інтерфейс застосунку, постановка задачі, тестові зображення.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	07.04.2025	
2	Аналіз завдання, підбір літератури	08.04.25-10.04.25	
3	Аналіз літератури з досліджуваної проблеми	11.04.25-14.04.25	
4	Аналіз технічних засобів	15.04.25-20.04.25	
5	Розробка методу	21.04.25-27.04.25	
6	Програмна реалізація	28.04.25-11.05.25	
7	Оформлення пояснювальної записки	12.05.25-20.05.25	
8	Перевірка на нормоконтроль	21.05.25-01.06.25	
9	Перевірка на плагіат	21.05.25-01.06.25	
10	Рецензування	21.05.25-01.06.25	
11	Підготовка презентації та доповіді	21.05.25-18.06.25	
12	Занесення роботи в електронний архів	02.06.25-18.06.25	
	Попередній захист кваліфікаційної роботи	02.06.25-18.06.25	

Дата видачі завдання 7 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ _____доц. Руденко Д. О._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 61 с., 9 табл., 29 рис., 27 джерело.

МУЗИЧНИЙ ВЕБЗАСТОСУНОК, ПЕРЕКЛАД ТЕКСТІВ ПІСЕНЬ, ШТУЧНИЙ ІНТЕЛЕКТ, OPENAI API, SPOTIFY API, GENIUS API, REST API, JWT.

Об’єктом роботи є процес взаємодії користувача з музичним і текстовим контентом з метою вивчення мови через інтерфейс вебзастосунку.

Метою роботи є розробка музичного вебзастосунку для прослуховування треків, перегляду їхніх текстів, перекладу й аналізу змісту за допомогою технологій штучного інтелекту.

Проведено дослідження сучасних цифрових рішень у сфері музики та мовного навчання. Розроблено вебзастосунок, що поєднує обробку музичного контенту, переклад і лінгвістичний аналіз за допомогою штучного інтелекту. Реалізовано взаємодію з зовнішніми сервісами, збереження даних та інтерфейс для користувача.

MUSIC WEB APPLICATION, SONG LYRICS TRANSLATION, ARTIFICIAL INTELLIGENCE, OPENAI API, SPOTIFY API, GENIUS API, REST API, JWT.

The object of this work is the process of user engagement with musical and textual content for the purpose of language learning through the interface of a web application.

The aim is to develop a music web application that enables users to listen to tracks, view lyrics, translate and analyze their content using artificial intelligence technologies.

A study was carried out on current digital solutions in the field of music and language learning. As a result, a web application was developed to provide users with access to music content, lyrics translation and AI-powered language analysis. The project includes integration with external services, reliable data storage, and a user interface.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 6
Вступ 7
1 Теоретичні основи створення музичного вебзастосунку 8
1.1 Огляд функцій музичних стрімінгових платформ та сервісів 8
1.2 Штучний інтелект у лінгвістичних застосунках 12
1.3 Аналіз і порівняння технологій для серверної частини 14
1.4 Постановка задачі 16
2 Проєктування музичного вебзастосунку 18
2.1 Функціональні вимоги до застосунку 18
2.2 Загальна архітектура системи 18
2.3 Модель бази даних та її логіка 20
2.3.1 Вибір інструменту для роботи з базою даних 21
2.3.2 Опис структури таблиць, типів даних, зв’язків 22
2.4 Серверна частина застосунку 26
2.4.1 REST API, CRUD, HTTP-методи, Dependency Injection 27
2.4.2 Робота з API сторонніх сервісів 28
2.4.3 Кешування за допомогою Redis 30
2.4.4 Перевірка користувача, авторизація (JWT) 31
2.5 Клієнтська частина застосунку 32
2.5.1 Технології для розробки клієнта 32
2.5.2 Структура сторінок застосунку 34
3 Практична реалізація застосунку 35
3.1 Інструменти та середовище розробки 35
3.2 Етапи реалізації основного функціоналу 37
3.3 Тестування і результати роботи застосунку 47
3.4 Перспективи для подальшого розвитку 56
Висновки 58
Перелік джерел посилання 59

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI − Artificial Intelligence (штучний інтелект)

GPT – Generative Pre-trained Transformer (генеративний попередньо навчений трансформер)

JS – JavaScript

JSON – JavaScript Object Notation (нотація об’єктів JavaScript)

JWT – JSON Web Token (вебтокен JSON)

HTML – Hyper Text Markup Language (мова гіпертекстової розмітки)

CSS – Cascading Style Sheets (каскадні таблиці стилів)

MVP − Minimum Viable Product (мінімально життєздатний проєкт)

API − Application Programming Interface (інтерфейс програмування застосунків)

ACID – Atomicity, Consistency, Isolation, Durability (атомарність, узгодженість, ізольованість, стійкість)

CORS – Cross-Origin Resource Sharing (обмін ресурсами між різними джерелами)

CRUD – Create, Read, Update, Delete (створювати, читати, оновлювати, видаляти)

ВСТУП

Музика давно перестала бути просто мистецтвом чи розвагою, сьогодні вона стала способом для самовираження, культурного об’єднання та навіть ефективним засобом для навчання, через який людина може не лише висловити власні емоції чи переживання, а й відчути зв’язок із культурою різних народів та країн, розширюючи таким чином межі власного бачення світу.

Особливо цінною є можливість використовувати музику для вивчення іноземних мов, адже тексти пісень містять розмовну лексику, сленг, ідіоми та граматику з усіма її винятками, які дуже рідко зустрінеш у звичайних навчальних матеріалах, і саме тому таке вивчення мови є досить ефективним.

Сучасні технології, такі як штучний інтелект, відкривають абсолютно новий підхід до взаємодії з музикою та мовами, адже завдяки використанню передових алгоритмів сьогодні можна не лише слухати улюблені треки, але й глибоко занурюватися в їх зміст, отримувати миттєвий переклад та аналіз складних лінгвістичних конструкцій, що значно підвищує ефективність навчання.

Таке поєднання музики й технологій не просто розширює культурні горизонти, а й стає справжнім помічником у вивченні мов, роблячи цей процес більш приємним. Відтепер кожна пісня стає не лише джерелом натхнення, а ще й повноцінним уроком, який захоплює, мотивує та допомагає розвиватися далі.

1 ТЕОРЕТИЧНІ ОСНОВИ СТВОРЕННЯ МУЗИЧНОГО ВЕБЗАСТОСУНКУ

1.1 Огляд функцій музичних стрімінгових платформ та сервісів

Розвиток сучасних технологій дав змогу поєднати музику та навчання в одному середовищі, тому що сьогодні з'являються нові формати взаємодії з музикою, коли йдеться не лише про прослуховування, а й переклад текстів, розбір фраз, візуалізацію лексики та подачу всього цього у зручному вигляді для користувача. Такий підхід відкриває нові перспективи для створення інструментів, які дозволяють об'єднати розвагу та навчання в одному середовищі.

Таким чином, музика стає не просто частиною культури, вона допомагає спілкуватись, вчити нові слова, краще розуміти структуру мови й бачити її в реальному використанні. Область даного дослідження ґрунтується на перетині мистецтва, мови та новітніх технологій.

Сучасний цифровий простір пропонує безліч музичних сервісів, кожен із яких має свої унікальні особливості.

Наприклад, Spotify (рис. 1.1), один із лідерів серед музичних стрімінгових платформ, спеціалізується активно на аудіоконтенті, пропонуючи користувачам багатомільйонну бібліотеку треків, персоналізовані рекомендації та тематичні плейлисти, але попри великий вибір музики, платформа не завжди забезпечує тексти пісень, а якщо й надає, то без перекладів чи пояснень, що є недоліком для користувачів, які хочуть глибше розуміти зміст пісень іноземною мовою.

Apple Music (рис. 1.2) також пропонує великий вибір треків, які доповненні текстами пісень. Особливістю сервісу є глибока інтеграція з пристроями Apple, що дозволяє комфортно слухати музику, отримувати тексти пісень та користуватися ексклюзивними функціями, але така

інтеграція є обмеженням для користувачів інших пристроїв, які не мають доступу до всього функціоналу сервісу.

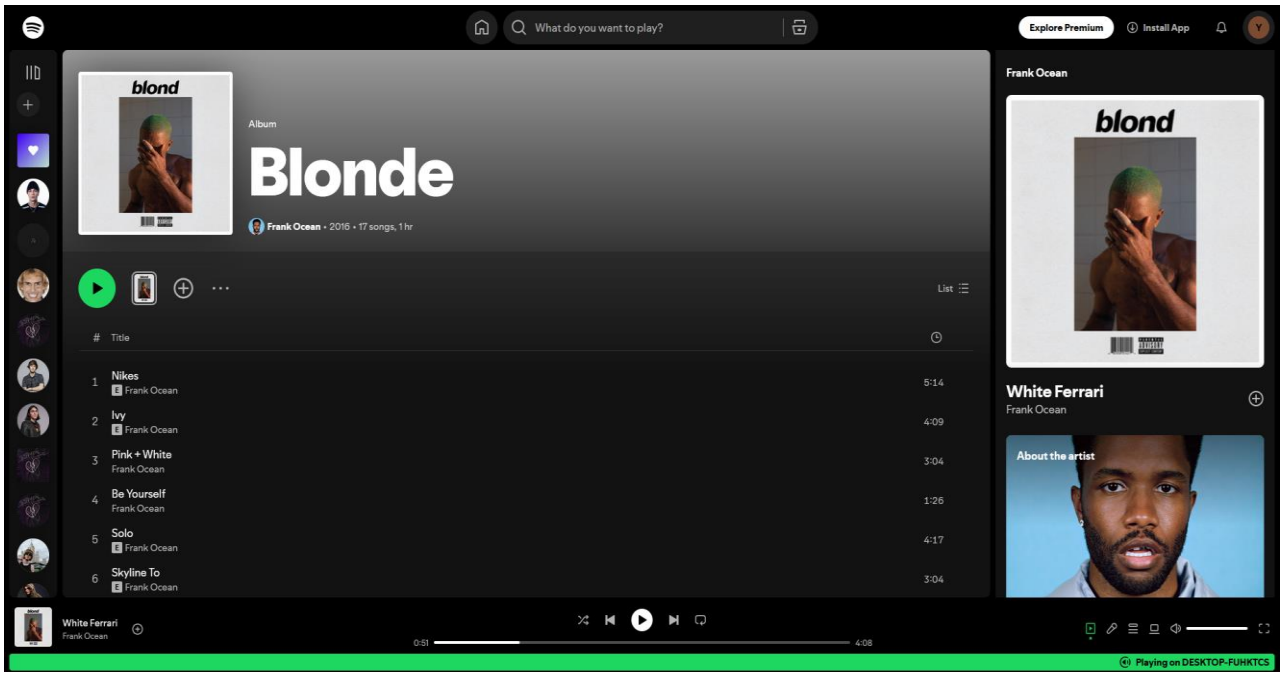

Рисунок 1.1 – Сторінка прослуховування Spotify

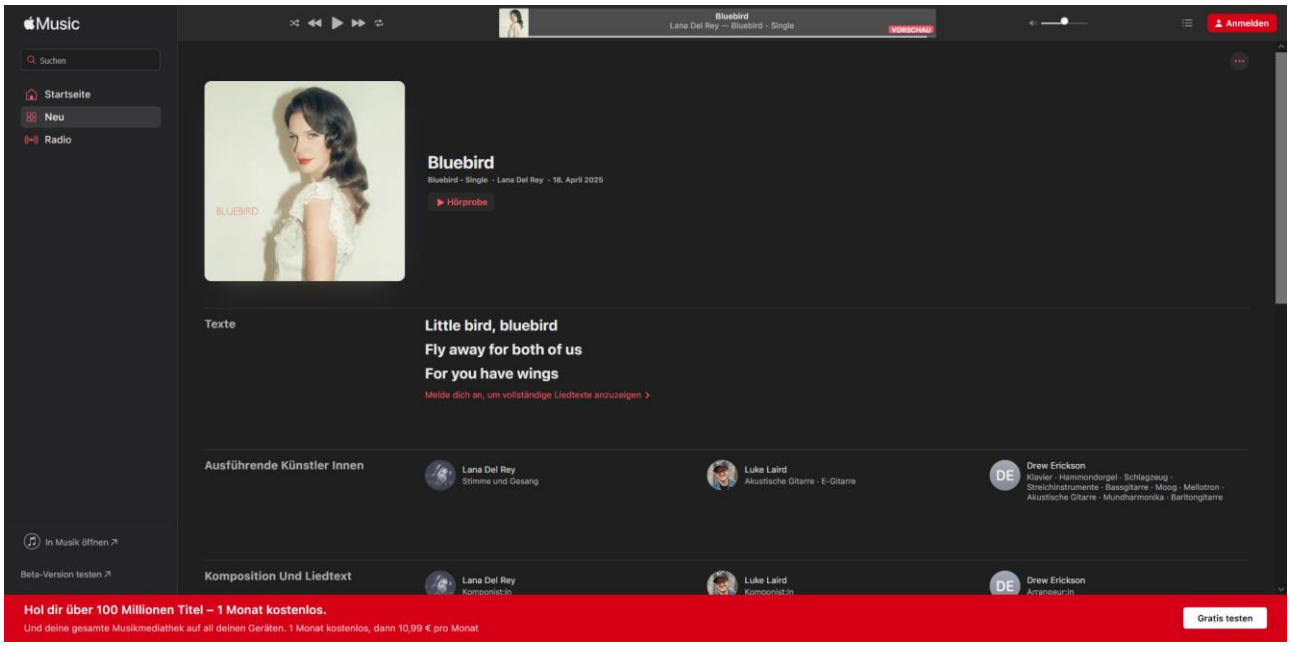

Рисунок 1.2 – Сторінка прослуховування Apple Music

Іншим популярним сервісом є YouTube Music (сторінка прослуховування представлена на рис. 1.3), що значно відрізняється

наявністю як музичного, так і відеоконтенту, включаючи кліпи, концертні записи та інші відеоматеріали артистів, проте, платформа не надає зручних інструментів для аналізу текстів пісень або їхніх перекладів, що обмежує можливості навчання чи занурення у зміст музичних творів.

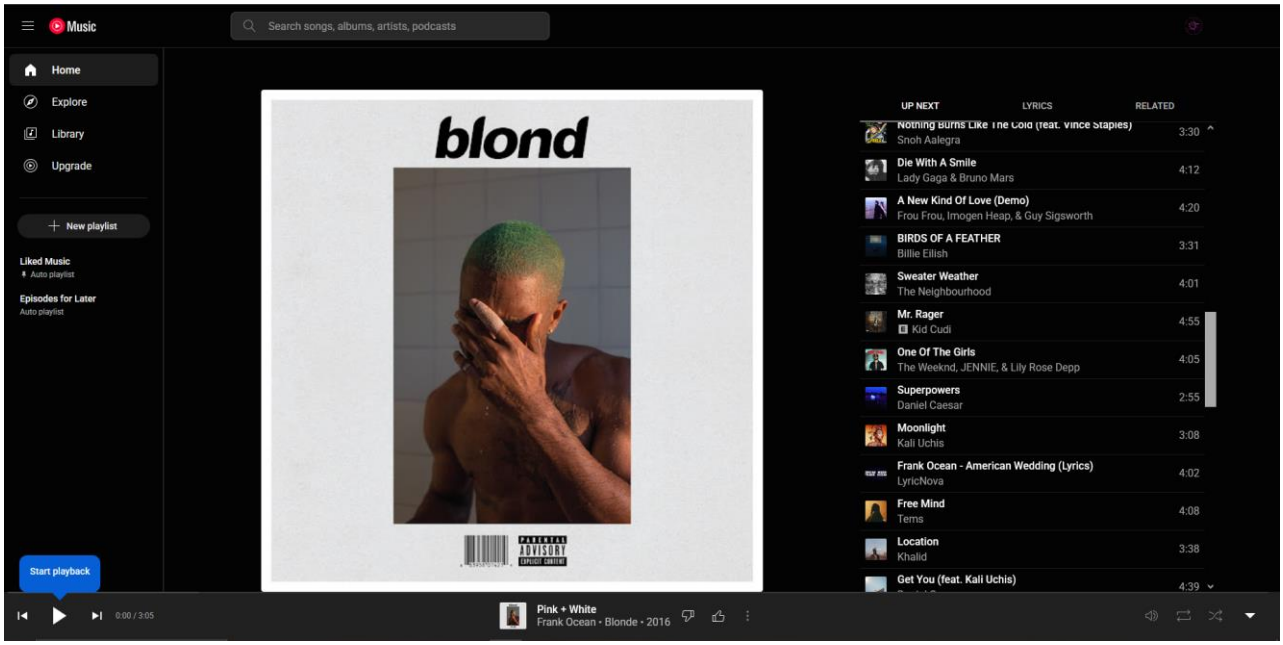

Рисунок 1.3 – Сторінка прослуховування YouTube Music

SoundCloud (рис. 1.4), навпаки, є майданчиком для незалежних виконавців, який дозволяє авторам ділитися власною творчістю без посередників. Через це сервіс дуже привабливий для того, щоб шукати нові імена або незвичайну, експериментальну музику. Однак, текстовий контент не представлений на SoundCloud, що робить неможливим повноцінну роботу з лірикою.

Genius (рис. 1.5) спеціалізується саме на текстах пісень, їх поясненнях та аналітиці. Платформа пропонує детальний аналіз та контекст кожного твору, що робить її незамінною для тих, хто прагне глибше розібратися у значенні пісень, але Genius не має повноцінного стрімінгу музики, обмежуючи користувачів лише короткими уривками через інтеграцію з Apple Music, що є проблемою для користувачів, які не користуються пристроями Apple.

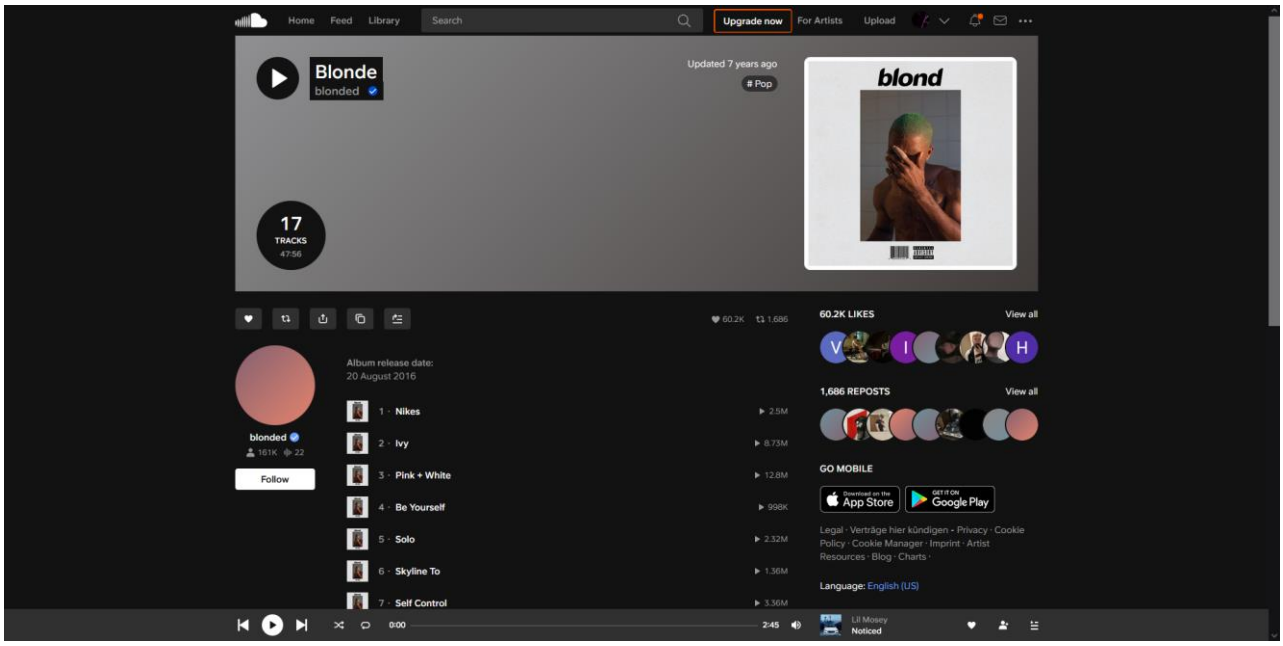

Рисунок 1.4 – Сторінка прослуховування Soundcloud

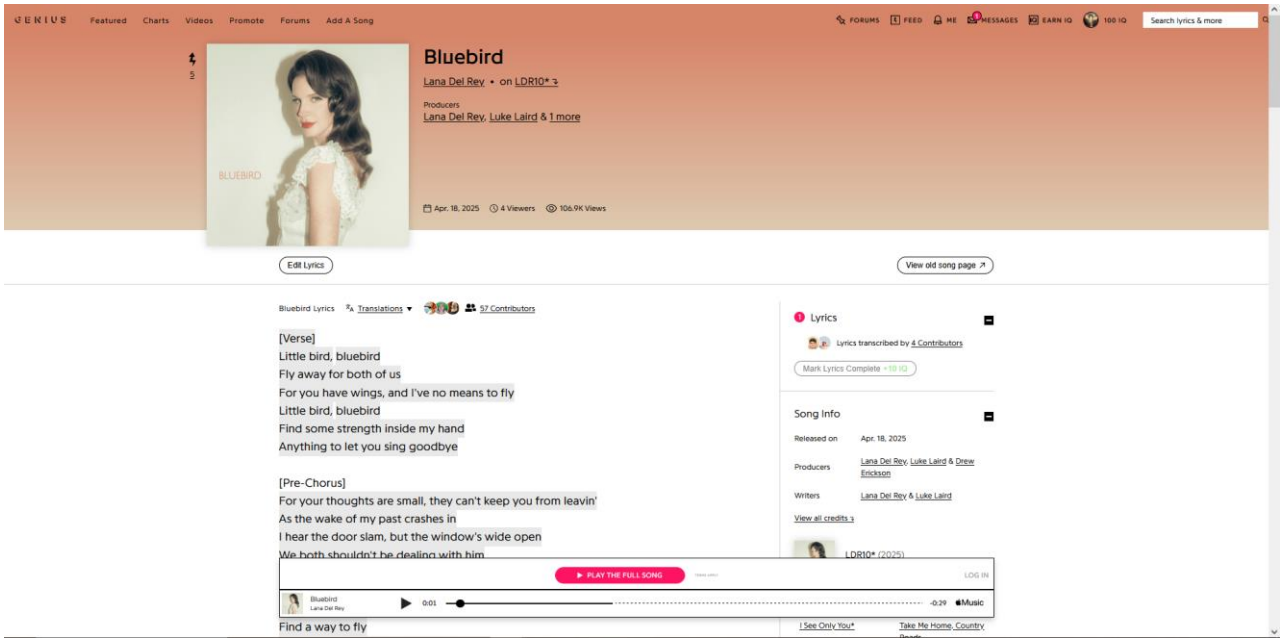

Рисунок 1.5 – Сторінка Genius

Щоб краще зрозуміти, які саме можливості пропонують різні платформи, треба подати їх функціональність у вигляді порівняльної таблиці (табл. 1.1).

Таблиця 1.1 – Порівняння функціональності популярних музичних сервісів

Платформа	**Прослухо-вування**	**Тексти пісень**	**Переклад**	**Пояснення змісту**	**Доступність API**
Spotify	Так	Частково	Ні	Ні	Так
Apple Music	Так	Так	Ні	Ні	Ні
YouTube Music	Так	Ні	Ні	Ні	Ні
SoundCloud	Так	Ні	Ні	Ні	Так
Genius	Частково	Так	Ні	Частково	Так

З цієї таблиці видно, що жодна з платформ не надає повного функціоналу для глибокої роботи з піснями з метою навчання, тому існує загальна потреба у створенні універсального рішення, що поєднує переваги прослуховування, читання, перекладу та аналізу текстів пісень, адже саме таке рішення здатне зробити процес взаємодії з музикою максимально ефективним та приємним.

1.2 Штучний інтелект у лінгвістичних застосунках

Упродовж останніх років штучний інтелект став однією з провідних рушійних сил у багатьох сферах, включно з медициною, правом, освітою, перекладом і мовним аналізом, адже його здатність працювати з текстами на глибшому рівні відкрила нові можливості у вивченні мов. Помітний прорив стався особливо завдяки появі великих мовних моделей, таких як GPT (Generative pre-trained transformer), які можуть не лише перекладати, а й пояснювати, адаптувати та коментувати текст.

У таких перекладачах, як Google Translate або Deepl, основна увага зосереджена на тому, щоб якомога точніше передати зміст оригіналу, часто через дослівний переклад або лише з мінімальним урахуванням контексту. Ці інструменти справляються з простими фразами досить впевнено, коли йдеться про меню в кафе, базову інструкцію чи повсякденне листування, але у випадку з пісенними текстами вони часто втрачають саму суть та не враховують настрій пісні, як саме побудовані фрази, чи присутня в ній гра слів або якийсь підтекст, і от саме цього такі перекладачі часто не помічають. Натомість великі мовні моделі на зразок GPT пропонують принципово інший підхід, вони не просто перекладають, а намагаються зрозуміти, що саме мав на увазі автор, як він сформулював думку, в якому стилі, і якщо в тексті є метафора чи ідіома, GPT не просто її перекладе, а пояснить, чому вона з’явилася, що вона означає і як її краще зрозуміти.

Цінність такого підходу полягає також у тому, що GPT здатен адаптуватися до рівня мовної підготовки користувача, якщо людина щойно починає вивчати мову, модель пояснює зміст максимально просто, без термінології та складних конструкцій, тоді як для досвідченого лінгвіста GPT здатен зануритися глибше, показати стилістичні особливості, вловити емоційний тон, проаналізувати структуру чи навіть римування. До того ж GPT дозволяє вести справжній текстовий діалог, бо користувач може поставити запитання щодо певної фрази, попросити порівняти значення схожих слів, знайти синоніми або запропонувати альтернативне формулювання, що значно перевищує функціонал класичних словників чи перекладачів, які працюють у форматі запит-відповідь. Саме такий гнучкий підхід перетворює штучний інтелект не просто на зручний інструмент, а на справжнього партнера у вивченні мов [1].

Таким чином, при використанні AI (Artificial Intelligence) на базі GPT у поєднанні з музикою маємо змогу перейти від звичайного прослуховування до глибокого занурення в зміст пісень, що перетворює кожну композицію не

лише на джерело емоцій, а й на повноцінний навчальний матеріал, який адаптується до потреб конкретної людини у зручний спосіб.

1.3 Аналіз і порівняння технологій для серверної частини

У процесі розробки вебзастосунків важливу роль відіграє правильний вибір технологій для бекенд-частини, тому що серверна логіка повинна бути швидкою, масштабованою, легкою у підтримці та гнучкою у взаємодії з базами даних, сторонніми API (Application Programming Interface) та з іншими інструментами. У мові програмування Python існує кілька популярних фреймворків для створення вебзастосунків: FastAPI, Django та Flask. Кожен із них має свої особливості, переваги й недоліки, які варто враховувати в контексті поставлених завдань.

Django є одним із найбільш повноцінних вебфреймворків, що забезпечує розробнику повний набір інструментів для швидкого створення вебзастосунків. Основними його перевагами є вбудована адмін-панель, потужна ORM для роботи з базами даних, велика екосистема та готові рішення для більшості завдань, однак цей фреймворк є досить важким і не завжди зручним для реалізації асинхронних операцій, що може впливати на подальшу продуктивність системи при великій кількості одночасних запитів або інтеграції із зовнішніми API, які вимагають асинхронного підходу [2].

Flask є легким мікрофреймворком, який надає велику гнучкість у виборі компонентів і підходів до реалізації, адже він дуже простий у використанні та має низький поріг входження, що робить його чудовим варіантом для менших проєктів або швидкого прототипування, проте розробнику доводиться самостійно налаштовувати та інтегрувати багато додаткових компонентів для складніших проєктів із великою кількістю функціоналу. Це може ускладнювати розробку та підтримку застосунку [3].

FastAPI є сучасним високопродуктивним фреймворком, він орієнтований на швидку розробку та ефективну підтримку асинхронних завдань, а його головними перевагами є швидкість, проста та зрозуміла документація, вбудована підтримка асинхронного програмування та легкість інтеграції з сучасними технологіями.

Вибір фреймворку для реалізації серверної частини застосунку було здійснено на користь FastAPI, через його здатність ефективно обробляти асинхронні запити до сторонніх сервісів, у тому числі музичних платформ та інструментів штучного інтелекту. Оскільки застосунок працює з великою кількістю даних у режимі реального часу, важливо було забезпечити хорошу швидкість та можливість масштабування. FastAPI добре підходить для цього завдяки повній підтримці асинхронного програмування, що базується на сучасному стеку Python [4 – 6].

Однією з важливих переваг фреймворку є автоматична генерація документації API через Swagger UI, що значно спрощує тестування та підтримку сервісу, а використання Pydantic для типізації даних гарантує точну валідацію вхідних і вихідних запитів, що зменшує ризики помилок на всіх етапах розробки. Крім того, у FastAPI легка інтеграція з Redis для кешування та SQLAlchemy для роботи з базами даних. Вона дозволяє побудувати ефективне серверне середовище, яке здатне обробляти велику кількість запитів без втрати продуктивності, що дуже важливо для стабільної роботи застосунку в умовах високого навантаження.

Тому FastAPI забезпечує не лише технічні переваги, а й зручність розробки завдяки зрозумілому синтаксису, гнучкості у побудові структури проєкту та підтримці роботи з сучасними стандартами безпеки, і саме завдяки цьому він став оптимальним вибором для створення швидкого, надійного та масштабованого вебзастосунку.

1.4 Постановка задачі

Попри те, що існуючі музичні сервіси дають змогу слухати треки та переглядати тексти пісень, але жоден із них не пропонує повноцінного інструменту для глибшого занурення в зміст пісень і використання їх у навчальних цілях. Відсутність можливості повноцінного перекладу текстів, аналізу виразів і інтеграції з інструментами штучного інтелекту обмежує такі платформи для вивчення іноземних мов в потенціалі.

Тому постає завдання створення нового вебзастосунку, який об’єднує можливості прослуховування музики, перегляду текстів пісень, перекладу й аналізу змісту за допомогою сучасних технологій.

Об’єктом роботи є процес взаємодії користувача з музичним і текстовим контентом з метою вивчення мови через інтерфейс вебзастосунку.

Метою роботи є розробка музичного вебзастосунку для прослуховування треків, перегляду їхніх текстів, перекладу й аналізу змісту за допомогою технологій штучного інтелекту.

Для досягнення мети необхідно вирішити наступні завдання:

– провести аналіз можливостей існуючих музичних платформ і сервісів роботи з текстами пісень;

– дослідити можливості використання штучного інтелекту для перекладу та аналізу текстів;

– обґрунтувати вибір технологій для реалізації серверної та клієнтської частин;

– розробити архітектуру вебзастосунку, яка забезпечує інтеграцію аудіоконтенту, перегляду текстів пісень і можливостей перекладу;

– створити базу даних для збереження інформації про артистів, треки та тексти;

– реалізувати серверну частину із використанням FastAPI, SQLAlchemy та Redis;

– розробити клієнтську частину вебзастосунку для зручної взаємодії з функціоналом;

– провести тестування застосунку, оцінити його продуктивність та зручність використання.

2 ПРОЄКТУВАННЯ МУЗИЧНОГО ВЕБЗАСТОСУНКУ

2.1 Функціональні вимоги до застосунку

Перед початком практичної реалізації вебзастосунку важливо було визначити, які саме функції мають бути реалізовані, щоб зробити роботу з музикою максимально зручною і водночас корисною для тих, хто хоче вивчати іноземні мови. Саме так і сформувався список основних функціональних вимог:

– користувач повинен мати можливість шукати виконавців або треки через взаємодію з зовнішніми музичними сервісами;

– користувач повинен мати можливість отримувати інформацію про артистів та треки;

– користувач повинен мати можливість отримувати зрозуміле пояснення складних слів або фраз за допомогою штучного інтелекту;

– користувач повинен мати можливість прослуховувати фрагмент треку, де це дозволено;

– система повинна надавати відображення тексту пісні;

– система повинна надавати переклад тексту пісні на обрану мову;

– система повинна кешувати дані для швидшого доступу до запитів, які повторюються.

2.2 Загальна архітектура системи

На початковому етапі розробки досить складно було визначитись, якою саме має бути архітектура системи, бо від того, як буде побудовано внутрішню логіку програми, залежатиме не тільки стабільність роботи застосунку, а й те, наскільки зручно буде вносити зміни, додавати нові функції або знаходити й виправляти помилки в майбутньому. Під час роботи

над проєктом дійшли висновку, що стандартний підхід із хаотичним з'єднанням компонентів швидко перестане працювати, особливо коли з'явиться штучний інтелект, зовнішні API, переклади, текстовий аналіз та багато іншого, що доведеться якось узгодити між собою. Було обрано рішення, яке одночасно дозволить тримати всі частини проєкту в єдиній структурі, але не змішувати їх, а дати кожній чітке місце та завдання. Саме так ми прийшли до модульного моноліту.

Модульна монолітна архітектура добре себе зарекомендувала ще тоді, коли проводилась робота над меншими проєктами MVP (Minimum Viable Product), де потрібно було підтримувати порядок, не створюючи зайвої складності. Суть у тому, що весь застосунок працює як одне ціле, однак усередині всі модулі чітко розподілені: один модуль підключається до музичних сервісів, інший обробляє запити користувача, ще один відповідає за інтеграцію з GPT. Кожен займається лише своїм і не втручається в роботу інших [7].

Приклад модульної монолітної архітектури подано на рисунку 2.1.

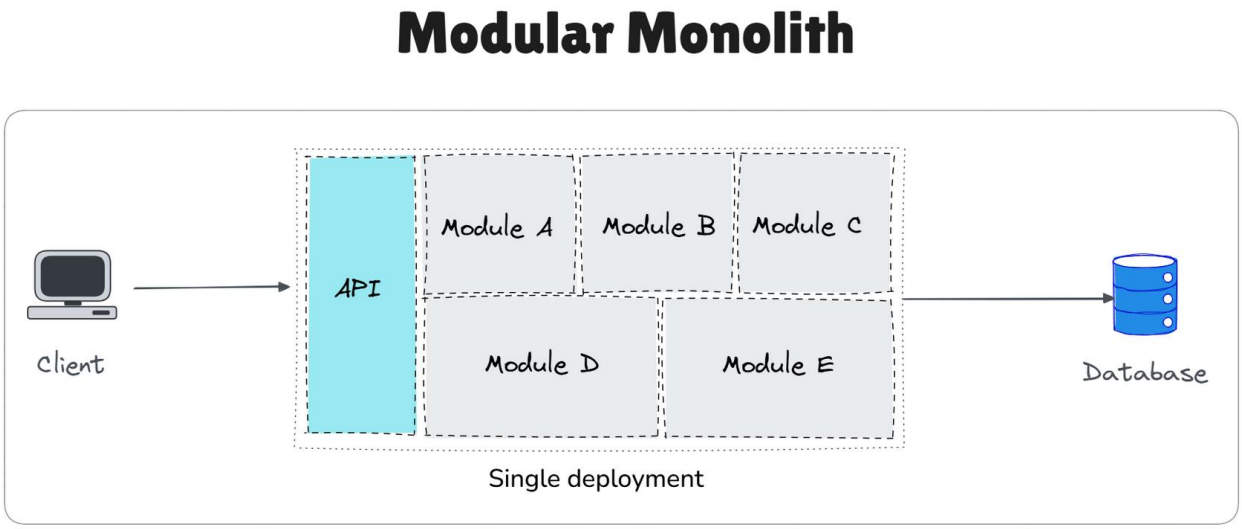

Рисунок 2.1 – Архітектура «модульний моноліт»

Основні характеристики архітектури «модульний моноліт» представлені в таблиці 2.1.

Таблиця 2.1 – Основні характеристики архітектури «модульний моноліт»

Характеристика	**Опис**
Єдине розгортання	Усі модулі включаються в один застосунок.
Модульність	Кожен модуль реалізує окрему бізнес-логіку і має чіткий інтерфейс.
Ізоляція	Модулі логічно ізольовані, але працюють у межах одного процесу.
Зручність підтримки	Код легко читати, тестувати та підтримувати завдяки модульності.
Масштабованість	Підходить для початку, з потенціалом переходу до мікросервісів.

2.3 Модель бази даних та її логіка

Після того як загальна структура застосунку була визначена, почалася робота над проєктуванням бази даних. Основне завдання полягало в тому, щоб організувати дані таким чином, щоб вони не тільки зберігались, а ще й були впорядкованими, зрозумілими та легко пов’язаними між собою. Потрібно було знайти таке рішення, яке дає змогу будувати чітку структуру, де все має свої місця.

Для цього ідеально підійшла реляційна база даних і було обрано саме PostgreSQL, бо ця система давно зарекомендувала себе як стабільна та надійна, і вона використовувалася мною в попередніх проєктах, тому було набагато простіше одразу зрозуміти, як і де її правильно інтегрувати. Основна перевага PostgreSQL в тому, що вона поєднує розвинений функціонал як: зовнішні ключі, транзакції, каскадне оновлення та видалення, унікальні обмеження, індексацію та багато інших інструментів, які дозволяють точно контролювати логіку взаємодії з даними. Також ця база

даних вміє не тільки працювати з класичними рядками чи числами, а й зберігає дані у форматі JSON (JavaScript Object Notation), тобто даними у вигляді звичайних рядків, які потім можна буде повернути до нормального вигляду через серіалізацію [8].

Окрему увагу варто приділити тому, що PostgreSQL повністю відповідає вимогам ACID (Atomicity, Consistency, Isolation, Durability), тобто кожна операція з даними є атомарною, узгодженою, ізольованою та надійною, а це означає, що система або виконує запит повністю, або не змінює дані зовсім. Це дуже важливо, коли система працює в реальному часі й кожна дія користувача щось змінює, навіть дрібний збій може обійтися дорого.

Завдяки цій архітектурі вдалося побудувати не просто базу даних, а цілісну логічну структуру, в якій все пов’язане і яка вже готова до розширення. Є можливість додавати нові поля або таблиці, не переписуючи все з нуля, і це гарний показник того, що база правильно спроєктована.

2.3.1 Вибір інструменту для роботи з базою даних

Коли починається робота з даними у застосунку, стає зрозуміло, що просто мати базу даних недостатньо. PostgreSQL хоч і дуже потужна, але для щоденної розробки хочеться чогось більш зручного та й постійно писати вручну запити SQL стає не тільки незручним, а й ризикованим, бо один неправильний рядок може зламати всю логіку. Тому було вирішено використати ORM (Object-Relational Mapping), яка допоможе простою та зрозумілою мовою поєднати базу даних з кодом. Найкращим варіантом для цього завдання була SQLAlchemy.

Цей інструмент дає змогу описувати таблиці, зв’язки та всю структуру бази прямо в коді. Кожен об’єкт у базі стає класом Python та зв’язки формуються як логічні зв’язки між класами, це дуже зручно і зрозуміло

навіть для тих, хто щойно приєднався до проєкту. Всі ці об’єкти працюють разом, і все це легко масштабувати, якщо знадобиться більше функцій. Уся логіка стає прозорою, і можна будувати серйозну систему без плутанини та дублювання [9 – 12].

Також з самого початку було прийнято важливе рішення використовувати асинхронну версію SQLAlchemy для доступу до бази даних, бо це дозволяє виконувати запити без блокування основного потоку, що критично важливо в умовах високонавантаженого середовища. Завдяки цьому сервер здатен одночасно обробляти десятки запитів користувачів, наприклад: один зберігає улюблений трек до вподобань, другий переглядає інформацію про трек, а третій відправив запит та чекає на відповідь.

Серед важливих інструментів, які ще були використані, був і Alembic. З його допомогою ми керуємо міграціями, тобто всіма змінами в структурі бази, а коли з’являється нова таблиця або змінюється поле, Alembic фіксує це у вигляді окремого файлу, який можна запустити будь-де. Це дає нам впевненість, що локальна версія і серверна завжди узгоджені. Особливо це допомагає, коли над проєктом працює команда, бо кожен бачить, які зміни були внесені, і не треба пояснювати їх вручну.

Разом SQLAlchemy і Alembic створили зручну і надійну основу для роботи з даними.

2.3.2 Опис структури таблиць, типів даних, зв’язків

У цьому розділі розглянуто детальніше, як побудована структура таблиць, які типи полів використовуються та як реалізовано логіку зв’язків між сутностями.

Таблиця artist (табл. 2.2) містить основні дані про виконавця: унікальний ідентифікатор id, ще один ідентифікатор genius_id, дату

отримання інформації у полі parse_date, а також текстове поле json, що використовується для зберігання повної відповіді від API у форматі строки.

Таблиця 2.2 – Таблиця artist музичного вебзастосунку

Data	**Type**	**Example**
id	Integer	1
genius_id	Integer	123456
parse_date	Timestamp	2025-02-04 12:00
json	String	{ "name": "West"}

Таблиця track (табл. 2.3) має поле artist_id, яке є зовнішнім ключем до таблиці artist, а також унікальне поле spotify_song_id, що використовується, як PrimaryKey, щоб ідентифікувати трек в системі та встановити зв’язок з іншими таблицями lyrics і track_details. Крім того таблиця містить текстовий опис, дату релізу, посилання на обкладинку та на прев’ю треку.

Таблиця 2.3 – Таблиця track музичного вебзастосунку

Data	**Type**	**Example**
id	Integer	1
artist_id	Integer (ForeignKey)	1
spotify_song_id	String	4uLU6hMCjMI75M1A2tKUQC
title	String	In the End
release_date	Timestamp	2001-10-09 00:00
cover_url	String	https://i.scdn.co/cover.jpg
preview_url	String	https://i.scdn.co/preview.mp3

Таблиця track_details (табл. 2.4) пов’язана з треками через поле spotify_song_id і зберігає характеристики пісні, такі як bpm, key, energy та інші параметри, які описані як рядки.

Таблиця 2.4 – Таблиця track_details музичного вебзастосунку

Data	Type	Example
id	Integer	1
spotify_song_id	String (ForeignKey)	4uLU6hMCjMI75M1A2tKUQC
key	String	C♯ minor
bpm	String	120
camelot	String	12A
popularity	String	85
energy	String	34
danceability	String	40
happiness	String	23

У таблиці lyrics (табл. 2.5) зберігаються тексти пісень, причому поле text має тип Text, тому що цей тип може зберігати великий обсяг даних, а поле spotify_song_id використовується як ForeignKey, що пов’язує конкретний текст із відповідним треком.

Таблиця 2.5 – Таблиця lyrics музичного вебзастосунку

Data	Type	Example
id	Integer	1
spotify_song_id	String (ForeignKey)	4uLU6hMCjMI75M1A2tKUQC
text	Text	It starts with one thing...

Таблиця user (табл. 2.6) виконує роль центральної в системі, оскільки містить поля, необхідні для автентифікації та авторизації користувача.

Додаткові таблиці user_liked_artist (табл. 2.7) і user_liked_track (табл. 2.8) реалізують логіку зв’язку користувача з виконавцями та треками, використовуючи комбінації зовнішніх ключів до таблиць user, artist і track.

Таблиця 2.6 – Таблиця user музичного вебзастосунку

Data	Type	Example
id	Integer	1
email	String	test@gmail.com
username	String	test
registered_at	Timestamp	2024-12-12 10:30
hashed_password	String	$2b$12$...
is_active	Boolean	true
is_superuser	Boolean	false
is_verified	Boolean	true

Таблиця 2.7 – Таблиця user_liked_artist музичного вебзастосунку

Data	Type	Example
id	Integer	1
user_id	Integer (ForeignKey)	1
artist_id	Integer (ForeignKey)	3
liked_at	Timestamp	2025-03-15 10:45

Таблиця 2.8 – Таблиця user_liked_track музичного вебзастосунку

Data	Type	Example
id	Integer	1
user_id	Integer (ForeignKey)	1
track_id	String (ForeignKey)	4uLU6hMCjMI69M1A2tKUQC
liked_at	Timestamp	2025-03-15 11:45

На рисунку 2.2 зображена логічна структура бази даних, яка включає основні таблиці та відображає взаємозв’язки між ключовими сутностями системи.

Рисунок 2.2 – Логічна структура бази даних

2.4 Серверна частина застосунку

Backend частина відповідає за те, щоб кожен запит, який надсилає користувач, не залишився без логічної і зрозумілої відповіді. Саме на цьому рівні система взаємодіє з базою даних, звертається до зовнішніх сервісів, обробляє інформацію і формує те, що бачить і отримує користувач у себе на екрані.

2.4.1 REST API, CRUD, HTTP-методи, Dependency Injection

Серверна частина застосунку була побудована за принципами REST, і це було свідоме рішення, оскільки такий підхід дає чітку та зрозумілу структуру, коли клієнт і сервер спілкуються між собою мовою стандартних HTTP-запитів (рис. 2.3). Це означає, що кожен об’єкт у системі має свою адресу і з ним можна взаємодіяти простими методами. Якщо користувач хоче переглянути інформацію про конкретний трек, достатньо на бекенді надіслати запит через метод GET на відповідний маршрут track з вказаним id, а щоб зберегти його, використовується POST. Якщо виникає потреба щось змінити, підійде PUT або PATCH, а для видалення даних потрібно надіслати метод DELETE. Уся маршрутизація побудована максимально логічно та без жодних складних рухів чи зайвих рішень для виконання операцій CRUD (Create, Read, Update, Delete) [13].

На мою думку великий плюс цього архітектурного стилю, що REST дотримується принципу Stateless (безстановості). Інакше кажучи, кожен запит, який надсилає клієнт, обробляється сервером окремо, без прив’язки до попередніх. Сервер не зберігає інформацію про сесію користувача, тому щоразу, коли відбувається якась дія, клієнт надсилає все, що потрібно для обробки. Це робить систему стабільнішою і зменшує навантаження, бо сервер не витрачає ресурси на збереження станів, щоб уникнути перенавантаження [14].

Коли будувалися маршрути, важливо було не тільки підтримувати гнучкість, а й уникнути дублювання коду. Саме тому використовувався шаблон проєктування, який називається Dependency injection (впровадження залежності). FastAPI має дуже зручний механізм, який дозволяє один раз описати потрібну логіку, а далі просто підключати її в тих місцях, де вона потрібна. Була створена окрема функція, яка повертає готовий об’єкт, замість того щоб кожного разу вручну створювати підключення до бази даних у кожному маршруті. Потім цей об’єкт автоматично підставляється, коли

запускається обробка запиту. Механізм працює так само і з іншими компонентами застосунку. За приклад можна взяти момент, коли користувач надсилає свій токен при автентифікації і залежність перевіряє його, витягує профіль і визначає, чи має він право виконувати запит.

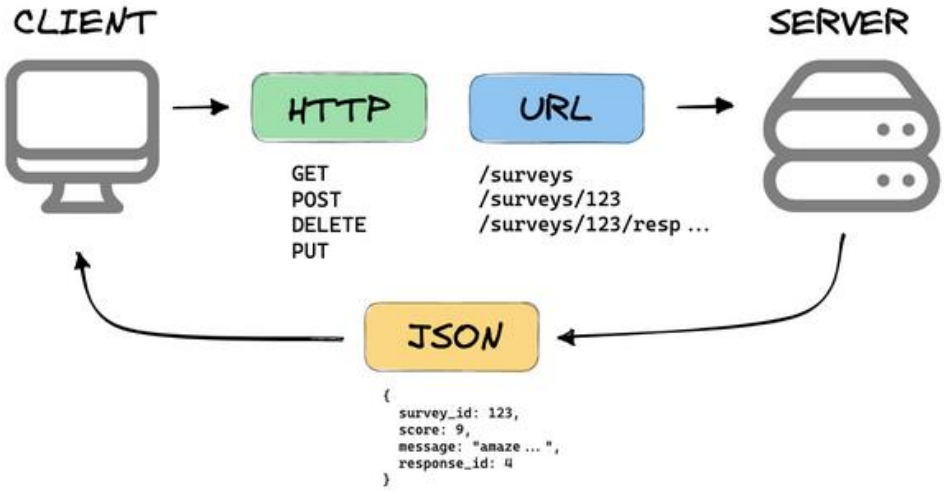

Рисунок 2.3 – Приклад роботи REST API

2.4.2 Робота з API сторонніх сервісів

Було вирішено під’єднати зовнішні сервіси до застосунку, щоб він був не просто пустою оболонкою, а дійсно працював з динамічним контентом. Без цих сервісів було б неможливо отримувати музику, тексти пісень чи забезпечувати функції штучного інтелекту, такі як переклад або пояснення складних фраз. Загальна ідея полягала в тому, щоб зібрати все потрібне з різних джерел і об’єднати в одному середовищі, яке має увесь необхідний функціонал, щоб не було необхідності користувачеві відкривати три окремі вікна. У результаті були підключені три API, а саме платформи Spotify, Genius і OpenAI, які разом дають усе необхідне для роботи системи.

Інтеграція зі Spotify API дозволяє користувачеві шукати треки, отримувати про них базову інформацію, серед якої ім’я виконавця, назва

треку, дата релізу, посилання на обкладинку, а також технічні деталі пісні, такі як темп, тональність або настрій. Щоб користувач міг працювати з цим API, було налаштовано авторизацію через токени, які оновлюються автоматично, коли його дія закінчується. Весь цей процес налаштований так, що користувач сам навіть не помічає, коли відбувається оновлення цього ключа. Також на сервері запити надсилаються через aiohttp, бо ця бібліотека відкриває асинхронну сесію і це означає, що система не зупиняється під час обробки запитів, а продовжує відповідати на інші дії користувача.

Інтеграція зі Genius API вийшла трохи складніше. Офіційне API нажаль не повертає повні текста, тому було вирішено реалізувати парсинг HTML (Hyper Text Markup Language). Ми використали для цього бібліотеку BeautifulSoup, яка дозволяє витягувати текст із розмітки прямо зі сторінки. А перед тим як зберігати текст до бази, відбувається пошук потрібного артиста, перевірка результатів і вже тільки тоді запускається завантаження повного тексту та як у випадку з Spotify API, усі ці запити працюють асинхронно.

Саме інтеграції з OpenAI було приділено найбільше уваги, адже ця частина відповідає за переклад пісень, пояснення незрозумілих слів та фраз, а також комунікацію у форматі діалогу. Наприклад, якщо пісня містить метафори або сленгові вирази, prompt будується так, щоб модель розуміла, що працює з фрагментом, у якому важливо передати емоцію та стиль. Щоб отримати якісний результат, було додатково налаштовано запити параметрами *temperature* і *max_tokens*. Перший параметр керує тим, наскільки креативною буде відповідь, а другий обмежує її довжину, щоб уникнути перевантаження та непотрібних витрат токенів. На виході отримується або повноцінний переклад, або пояснення з прикладами, іноді з коментарями до граматики.

2.4.3 Кешування за допомогою Redis

Важливо було додати ефективний механізм кешування, щоб забезпечити стабільну та швидку роботу серверної частини застосунку, особливо коли є повторні запити або затримки у відповіді від зовнішніх сервісів. Доступ до кешу набагато швидший, ніж комунікація з базою даних через мережу, і це не лише прискорює роботу системи, але й суттєво знижує мережевий трафік. Клієнтське кешування додатково зменшує навантаження на сервер бази даних, дозволяючи оптимізувати використання апаратних ресурсів. Це особливо ефективно у тих випадках, коли, за принципом Парето, невелика частина даних використовується значно частіше за інші [15]. Саме тому було вирішено використати Redis, як основне рішення для кешування.

Redis було обрано, тому що він працює дуже швидко, в нього доволі проста інтеграція з FastAPI та дозволяє зручно зберігати дані, які він відправляє до оперативної пам’яті, тому відповіді повертаються миттєво. Зокрема у цьому проєкті застосовується механізм кешування для збереження результатів запитів до джерела OpenAI, що дає змогу повторно використовувати вже отримані відповіді, коли користувач звертається до тієї ж інформації, тобто при повторному перекладі. Наприклад, якщо користувач вже отримував переклад певної чистини, система тимчасово збереже ці дані для нього, щоб уникнути повторного звернення до API, бо це дозволяє знизити навантаження на зовнішні сервіси [16].

Таке кешування особливо допомогло під час тестування, бо система почала працювати швидше і навіть зменшилась кількість звернень до зовнішніх сервісів. Також це знизило ймовірність перевищення лімітів у сервісі OpenAI, де кожен запит має свою вартість або обмеження за частотою. Була не тільки підвищена швидкість завдяки Redis, а ще й зроблена робота системи більш економною, бо кешування дає змогу розробникам використовувати системні ресурси оптимально.

Приклад роботи з кешуванням наведений на рисунку 2.4.

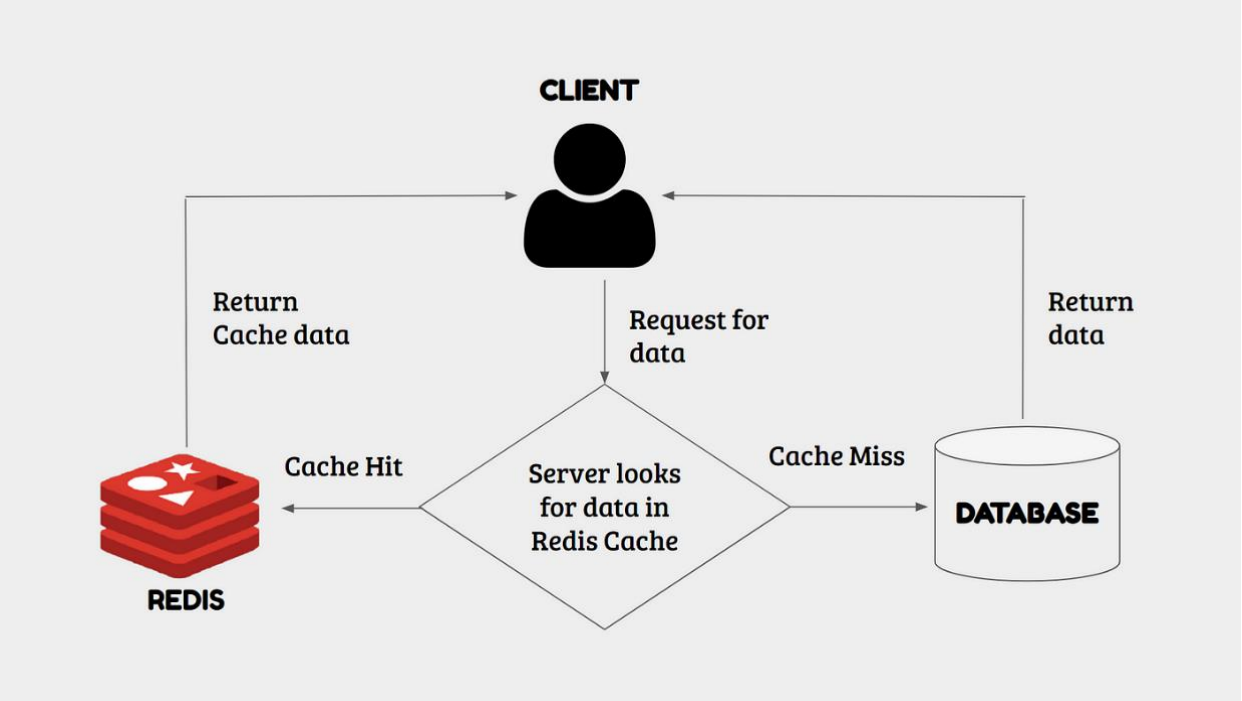

Рисунок 2.4 – Приклад роботи з кешуванням

2.4.4 Перевірка користувача, авторизація (JWT)

Безпека відіграє важливу роль у сучасному вебзастосунку, тому було прийнято рішення реалізувати систему авторизації на базі токенів JWT (JSON Web Token). Це сучасний спосіб безпечної передачі інформації між клієнтом і сервером, який дозволяє зберігати дані про користувача у вигляді зашифрованого рядку, що створюється після успішної автентифікації. Сам по собі токен містить три частини, які розділені крапкою всередині: заголовок, який включає інформацію про те, що за алгоритм буде використовуватися, корисне навантаження, яке містить основну інформацію, та підпис, який створюється на основі перших двох. Алгоритм HMAC-SHA256 використовується для створення підпису в JWT, який забезпечує достовірність даних і дає змогу серверу переконатися, що токен не було змінено, перш ніж надати доступ до захищених даних [17 – 20].

У цьому проєкті було задано термін дії токена всього дві години і це означає, що після входу користувач може працювати із застосунком

протягом цього часу, але коли час спливає, токен стає недійсним. Тому, якщо він знову намагається звернутися до захищеного ендпоінта, сервер повертає відповідь з кодом 401 Unauthorized, що є сигналом для повторної авторизації користувача.

Саме бібліотека FastAPI Users дозволила без зайвих зусиль реалізувати логін і контроль доступу у REST API, вона значно спростила роботу з авторизацією і дозволила не витрачати час на реалізацію всіх механізмів вручну. Ця бібліотека має вже все необхідне для створення облікових записів, а також хешування паролів та перевірку токенів.

Було отримано повний контроль над тим, хто і коли має доступ до тієї чи іншої частини застосунку.

2.5 Клієнтська частина застосунку

2.5.1 Технології для розробки клієнта

Клієнтська частина цього застосунку була створена за допомогою HTML, CSS (Cascading Style Sheets) та JS (JavaScript). Вона відповідає за те, як виглядає інтерфейс для користувача та яким чином він взаємодіє з функціоналам, починаючи з пошуку артистів до перегляду текстів пісень, закінчуючи отриманням відповідей від штучного інтелекту. HTML відповідає за структуру сторінки, адже розмітка визначає, де саме буде розміщено пошук, текст пісні або чат із моделлю, а в той час CSS керує тим, як все виглядає та налаштовує кольори, шрифти, відступи та адаптацію під різні екрани. JavaScript додає вже динаміки у системі, забезпечує взаємодію з сервером, обробку запитів і швидке оновлення контенту без перезавантаження сторінки.

Також у цьому проекті було важливо враховувати особливості, які властиві всім веббраузерам, оскільки коли JavaScript працює з API та активно надсилає запити до бекенду, у свою чергу браузер дуже уважно слідкує за

тим, звідки саме цей запит надходить. І тут важливо було розуміти, що браузер вважає різними джерелами ті, що мають хоча б незначну відмінність у домені, порту або протоколі. Тобто навіть якщо два сервери працюють локально на одному комп'ютері, але на різних портах, наприклад один на 8006, а інший на порту 8007, браузер буде все одно сприймати це запитом з іншого джерела та відразу ж блокувати його. Саме так працює механізм безпеки, який називається CORS (Cross-Origin Resource Sharing) [21].

Це справді дуже корисна річ, коли йдеться про безпеку користувачів в інтернеті. Тому розробник повинен подбати та налаштувати все так, щоб людина потім не хвилювалася за те, що її важлива інформація, наприклад дані банківської карти, потрапили до рук зловмисників.

Якщо у Backend частині все ж таки буде розширення та з'явиться ще один сервер, треба буде вказати, які джерела мають доступ до ресурсів в веббраузері та їх завантаження. Настроїти цю конфігурацію було доволі легко, бо CORS middleware вже є частиною бібліотеки Starlette, поверх якої була вже побудована бібліотека FastAPI. Цей Middleware допомагає повідомити браузеру, що запити з певного джерела є безпечними і можуть виконуватись (рис. 2.5).

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:8007",
                   "http://127.0.0.1:8007"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Рисунок 2.5 – Налаштування CORS middleware

2.5.2 Структура сторінок застосунку

Структура сторінок застосунку була побудована для користувача максимально просто та зрозуміло, щоб не перевантажувати інтерфейс зайвими елементами. Макети сторінок охоплюють дві основні частини: сторінку виконавця (рис. 2.6) та сторінку окремого треку (рис. 2.7).

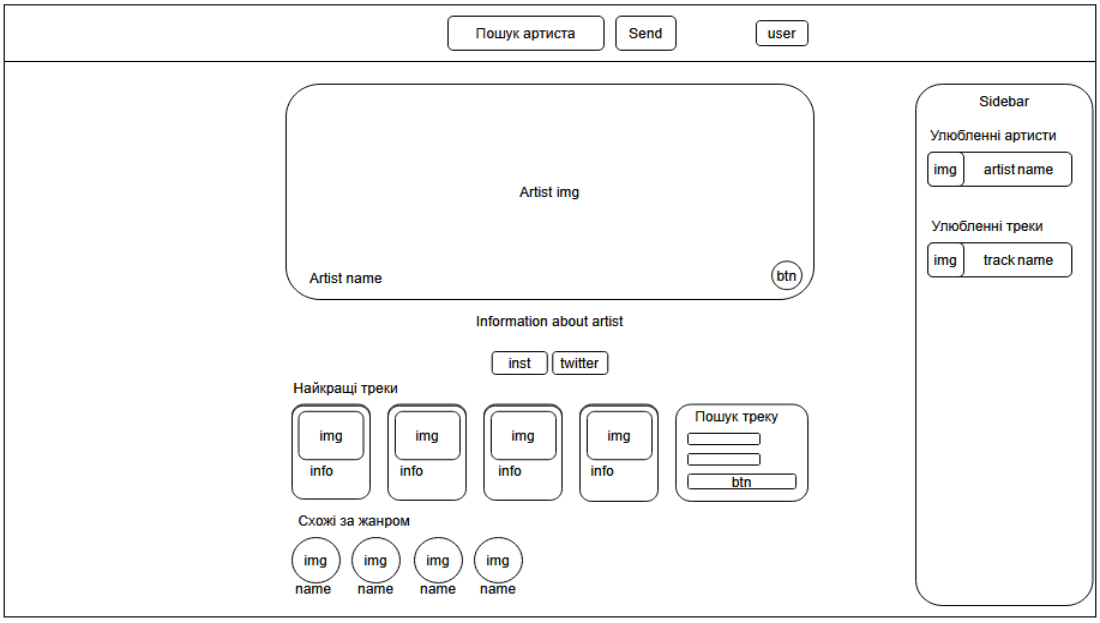

Рисунок 2.6 – Макет сторінки виконавця

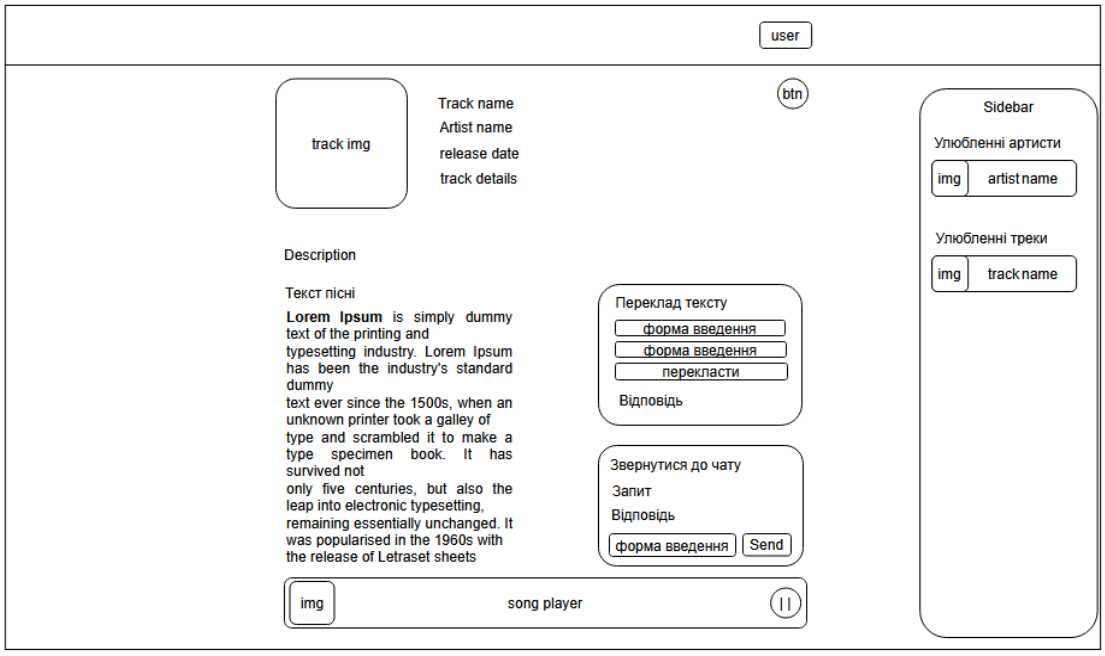

Рисунок 2.7 – Макет сторінки треку

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

3.1 Інструменти та середовище розробки

Visual Studio Code став основним середовищем програмування для розробки, адже цей редактор легкий, зручний і водночас дає все необхідне для повноцінної роботи. Саме завдяки підтримці Python, інтеграції з Git, вбудованому терміналу та великій кількості розширень було легко організувати повноцінний робочий процес. Окремо варто відзначити підтримку роботи з віртуальним середовищем, а також розширення, такого як Pylance, яке забезпечувало підсвічування синтаксису, перевірку типів та помилок. Усе це дозволяло не витрачати час на налаштування середовища вручну, а зосередитися на написанні та тестуванні коду. Тому редактор гарно підійшов як для розробки серверної логіки, так і для роботи з конфігураціями контейнерів Docker, що спростило процес реалізації проєкту.

Swagger UI є гарним вибором у процесі розробки та тестування серверної частини застосунку, бо його використання дало змогу суттєво спростити перевірку роботи API на всіх етапах, починаючи з створення перших ендпоінтів і до повної інтеграції з клієнтською частиною. Документація генерується автоматично на основі маршрутів у FastAPI, що дозволяє одразу бачити, які запити можна надсилати, які параметри вони приймають і яку відповідь формують. У процесі розробки Swagger фактично замінив такий інструмент, як Postman, і дозволив значно швидше перевіряти зміни після оновлення коду.

Docker було використано для того, щоб упакувати застосунок разом із усіма його залежностями, конфігураціями та середовищем виконання в один ізольований контейнер, що може бути запущений на будь-якому сервері незалежно від операційної системи чи хосту. Завдяки цьому повний серверний стек з FastAPI, PostgreSQL, Redis та з всіма потрібними бібліотеками розгортається у відокремленому середовищі, яке гарантує

стабільність і однаковий результат запуску як у середовищі розробника, так і на продакшн-сервері, без потреби вручну встановлювати компоненти або конфігурації залежностей.

Контейнеризація дозволила зібрати застосунок у вигляді Docker-образу, в якому чітко визначено інструкції для встановлення пакетів, налаштування змінних середовища, відкриття необхідних портів і запуску сервера. При цьому за основу береться стандартний Dockerfile, де описано базовий образ Python, встановлення всіх залежностей із файлу requirements, конфігурацію та запуск сервера FastAPI за допомогою Uvicorn, а для паралельного запуску декількох сервісів, таких як база даних PostgreSQL і кеш-сховище Redis, використовується інструмент docker-compose, який поєднує всі сервіси в єдину логічну структуру, як показано на рисунку 3.1.

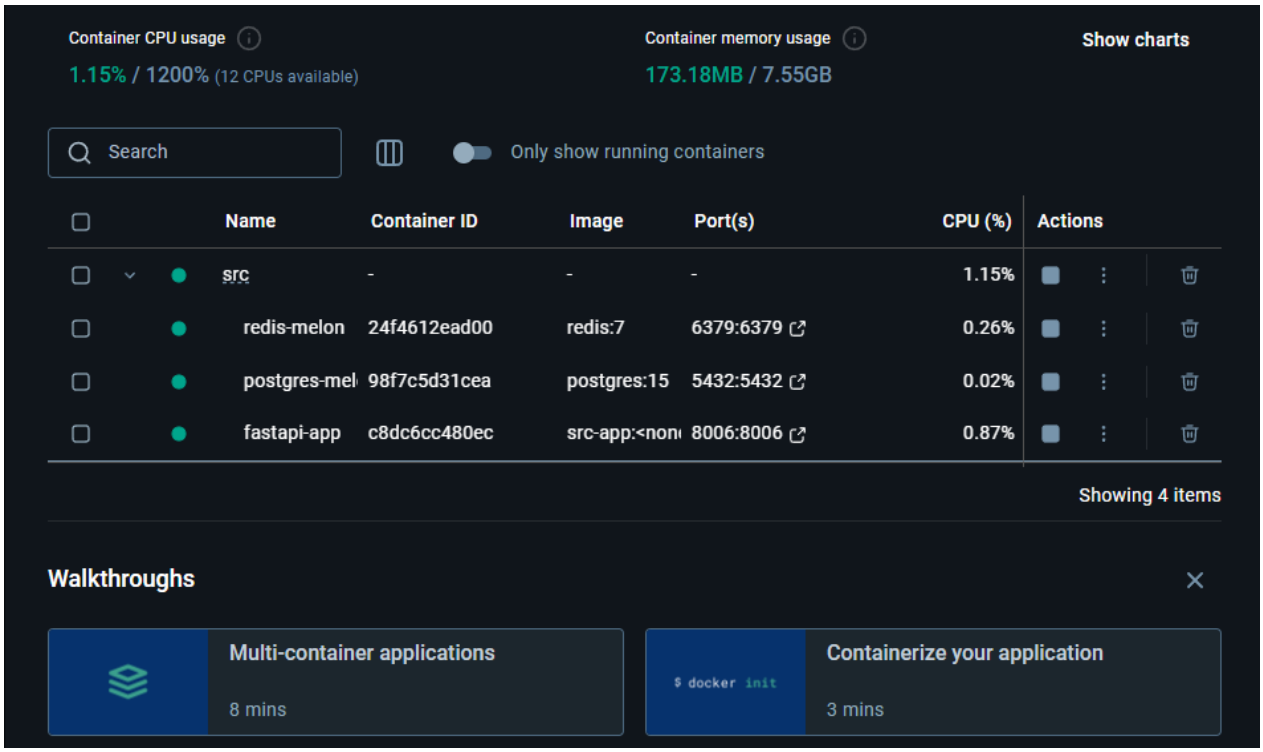

Рисунок 3.1 – Приклад роботи Docker

У такому середовищі всі компоненти застосунку взаємодіють між собою через внутрішню мережу, яка автоматично налаштовується при старті контейнерів, без потреби вказувати вручну IP-адрес або порти. Наприклад,

FastAPI звертається до бази даних за допомогою сервісного імені postgres, а до кеш-системи через redis. Всі ці сервіси запускаються одночасно за допомогою сценарію docker-compose, і це гарантує, що середовище буде стабільним. Такий підхід добре працює як для локального тестування, так і для перенесення проєкту в хмару без додаткових кроків.

Також Docker виявився зручним інструментом для оновлення сервісів, ізоляції середовища під час експериментів та створення окремих контейнерів для розробки. Саме тому він став надійною основою, яка допомагає впевнено керувати інфраструктурою в будь-яких умовах.

3.2 Етапи реалізації основного функціоналу

Перед тим, як звернутися до штучного інтелекту для аналізу або перекладу тексту пісні, потрібно було спочатку настроїти систему на збір усієї необхідної інформації. Саме тому було реалізовано інтеграцію з двома основними музичними сервісами, а саме Genius API та Spotify API.

Для отримання текстів пісень і пошуку артистів застосунок використовує повну взаємодію з Genius. Як було сказано раніше, усі запити до цього сервісу виконуються асинхронно за допомогою бібліотеки aiohttp, яка дозволяє відкривати сесію HTTP та надсилати запити асинхронно без блокування основного потоку.

Коли користувач вводить ім’я виконавця, система формує запит до ендпоінту пошуку https://api.genius.com/search, де параметром q передається зазначене ім’я, до запиту також додається токен авторизації, який зберігається у словнику request_params. Увесь запит відбувається в конструкції контекстного менеджеру async with, що відкриває HTTP-сесію і дозволяє працювати з нею асинхронно та безпечно. У разі успішної відповіді з кодом 200 результат розпаковується як JSON, після чого з нього витягується список hits, що містить інформацію про треки артиста. При

цьому для уникнення помилок під час доступу до вкладених полів використовується метод .get(), який дає змогу обробити структуру навіть у разі відсутності потрібних ключів та видати пусте значення.

Лістинг 3.1 Логіка роботи з Genius API для пошуку артистів:

```
class GeniusAPI:
def __init__(self, access_token: str):
self._token = access_token
self.request_params = {
"access_token": self._token
}
feature_symbols = [",", "&"]
def featured_artist(self, artist_name: str) -> bool:
for feature_symbol in GeniusAPI.feature_symbols:
if feature_symbol in artist_name:
return True
return False
async def get_artist_id(self, artist_name:str) -> int:
async with aiohttp.ClientSession() as session:
url = "http://api.genius.com/search"
request_params = self.request_params
request_params['q'] = artist_name
async with session.get(url=url, params=request_params) as response:
if response.status != 200:
raise HTTPException(status_code=response.status, detail="Failed to fetch
Genius data")
data = await response.json()
hits = data.get("response", {}).get("hits", [])
target_name = artist_name.strip().lower()
def is_valid_artist(name: str) -> bool:
```

```
return name.strip().lower() == target_name and not self.featured_artist(name)
# 1. Спочатку type == "artist"
for hit in hits:
if hit.get("type") == "artist":
name = hit["result"]["name"]
if is_valid_artist(name):
return hit["result"]["id"]
# 2. Шукаємо серед song
for hit in hits:
if hit.get("type") == "song":
result = hit["result"]
if result.get("featured_artists"):
continue
primary_artist = result.get("primary_artist", {})
name = primary_artist.get("name", "")
if is_valid_artist(name):
return primary_artist["id"]
raise Exception("Cannot find a suitable artist in Genius hits")
async def get_artist(self, artist_id: int) -> GeniusArtist:
async with aiohttp.ClientSession() as session:
url = f"http://api.genius.com/artists/{artist_id}"
async with session.get(url=url, params=self.request_params) as response:
if response.status != 200:
raise HTTPException(status_code=response.status, detail="Failed to fetch Genius data")
data = await response.json()
artist_dict: dict = data["response"]["artist"]
```

```
if
artist_dict["image_url"].startswith("https://assets.genius.com/images/default_avat
ar"):
raise Exception("Artist has default avatar")
artist = GeniusArtist(**artist_dict)
return artist
```

Після того, як система знаходить *artist_id*, наступним кроком стає пошук конкретного треку та отримання його тексту. Для цього формується новий запит до того самого маршруту http://api.genius.com/search, цього разу параметр *q* має не лише ім’я виконавця, а й назву самого треку, у відповідь надходить список об’єктів типу song, з якого система обирає перший підходящий результат і витягує поле url на сторінку HTML треку. Саме за цим посиланням ми і можемо дістати повний текст пісні.

Лістинг 3.2 Логіка роботи з Genius API для пошуку треків:

```
async def get_artist_song(self, artist_name: str, track_title: str):
async with aiohttp.ClientSession() as session:
url = "http://api.genius.com/search"
request_params = self.request_params
query = f"{artist_name} {track_title}"
request_params['q'] = query
async with session.get(url=url, params=request_params) as response:
if response.status != 200:
raise HTTPException(status_code=response.status, detail="Failed to fetch
Genius data")
data = await response.json()
hits = data.get("response", {}).get("hits",[])
for hit in hits:
if hit["type"] == "song":
```

```
return hit["result"]["url"]
return None
async def get_songs_text(self, track_url: str)-> list[str]:
async with aiohttp.ClientSession() as session:
async with session.get(track_url) as response:
html = await response.text()
soup = BeautifulSoup(html, 'lxml')
lyrics_div = soup.find_all('div', attrs={"class":
re.compile(r"^Lyrics__Container-sc-")}) # Lyrics-sc-7c7d0940-1 gVRfzh
if not lyrics_div:
return None
full_text = "\n".join(div.get_text(separator="\n").strip() for div in
lyrics_div)
start_keywords = ["[Intro", "[Verse", "[Chorus", "[Bridge", "[Outro",
"[Refrain",
"[Post-Chorus", "[Pre-Chorus", "[Breakdown", "[Interlude:", "[Skit:"]
for keyword in start_keywords:
idx = full_text.find(keyword)
if idx != -1:
full_text = full_text[idx:]
break
```

Аналогічно до взаємодії з минулим API, для роботи зі Spotify API в системі також було реалізовано асинхронне з’єднання через бібліотеку aiohttp. Однак на відміну від Genius у цьому випадку дані запитуються з іншого ендпоінту під шляхом /v1/search, а авторизація потребує наявності дійсного токена доступу, який створюється на основі значень *client_id* і *client_secret*. Тобто перш ніж надіслати запит, система повинна мати дійсний токен, який підтверджує право доступу до Spotify API.

Щоб отримати *artist_id*, у параметри запиту передається *q* з ім’ям виконавця, а також тип об’єкта artist. Якщо з якихось причин токен стає недійсним, наприклад закінчився час дії або перевищено ліміт, система автоматично викликає метод оновлення, який через маршрут https://accounts.spotify.com/api/token надсилає запит на видачу нового токена, отримує його і оновлює заголовки для подальшої роботи.

Лістинг 3.3 Логіка роботи з Spotify API для пошуку артистів:

```
class SpotifyAPI:
def __init__(self, access_token: str, client_id:str, client_secret: str) -> None:
self._token = access_token
self.client_id = client_id
self.client_secret = client_secret
self.dheaders = {
"Authorization": f"Bearer {access_token}",
"Content-Type" : "application/x-www-form-urlencoded"
}
async def refresh_token(self):
auth_base64 =
base64.b64encode(f"{self.client_id}:{self.client_secret}".encode()).decode()
url = f"https://accounts.spotify.com/api/token"
headers = {
"Content-Type": "application/x-www-form-urlencoded",
"Authorization": f"Basic {auth_base64}"
}
data = {
"grant_type": "refresh_token",
"refresh_token": os.environ["SPOTIFY_REFRESH"]
}
async with aiohttp.ClientSession() as session:
```

```
        async with session.post(url=url, data=data, headers=headers) as response:
        resp_text = await response.json()
        self._token = resp_text["access_token"]
        self.dheaders["Authorization"] = f"Bearer {self._token}"
        async def get_artist_id(self, artist_name:str) -> int:
        async with aiohttp.ClientSession() as session:
        url = f"https://api.spotify.com/v1/search"
        params = {
        'q': artist_name,
        'type': "artist"
        }
        async with session.get(url=url, headers=self.dheaders, params=params) as
response:
        data = await response.json()
        try:
        artists = data["artists"]
        except KeyError:
        await self.refresh_token()
        return await self.get_artist_id(artist_name)
        first_artist_id = artists['items'][0]['id']
        return first_artist_id
        async def get_artist(self, artist_id: int):
        async with aiohttp.ClientSession() as session:
        url = f"https://api.spotify.com/v1/artists/{artist_id}"
        async with session.get(url=url, headers=self.dheaders) as response:
        data = await response.json()
        try:
        artist = SpotifyArtist(
        name=data["name"],
        genres=data["genres"],
```

```
followers_count=data["followers"]["total"],
avatar_photo=data["images"][0]["url"],
popularity=data["popularity"]
)
return artist
except Exception as e:
raise Exception (f"Помилка під час пошуку артиста: {e}")
```

Також окрему роль у логіці взаємодії зі Spotify API відіграє запит до ендпоінту /v1/artists/{id}/top-tracks, який дозволяє отримати список найпопулярніших пісень виконавця. Уся ця інформація структурується у вигляді об’єктів і передається на клієнтську частину для відображення у вигляді карток треків.

Лістинг 3.4 Логіка роботи з Spotify API для пошуку треків:

```
async def get_artist_top_tracks(self, artist_id: str) -> list[SpotifyTrack]:
async with aiohttp.ClientSession() as session:
url = f"https://api.spotify.com/v1/artists/{artist_id}/top-tracks?market=ES"
async with session.get(url=url, headers=self.dheaders) as response:
data = await response.json(content_type=None)
tracks = data.get("tracks", [])
tracks_items: list[SpotifyTrack] = []
for track in tracks[:10]:
artist_names = ", ".join(artist["name"] for artist in track["artists"])
track_obj = SpotifyTrack(
spotify_song_id=track["id"],
artists=artist_names,
title=track["name"],
release_date=track["album"]["release_date"],
cover_url=track["album"]["images"][0]["url"],
```

```
preview_url=track["preview_url"],
)
tracks_items.append(track_obj)
return tracks_items
```

Після завершення етапу збору інформації з зовнішніх джерел, система переходить до головного сценарію цього застосунку – це взаємодія користувача з інструментом штучного інтелекту, який допомагає краще зрозуміти зміст пісні, пояснює складні фрази, перекладає текст і підлаштовує відповідь відповідно до мовного рівня людини. Саме цей функціонал був реалізований через інтеграцію з OpenAI API, що дозволяє надсилати запити до мовної моделі GPT і отримувати відповіді у вигляді пояснення.

Усі запити до GPT були організовані через окремий клас OpenAIClient, де в методі analyze_text формується спеціальний prompt, у якому враховується мова пояснення, рівень знань користувача і сам фрагмент пісні. При цьому роль моделі задається як роль викладача, тобто GPT поводиться не просто як перекладач, а як справжній вчитель, який пояснює матеріал, враховуючи рівень слухача. Така подача робить відповіді інформативними та зрозумілими.

Лістинг 3.5 Логіка виклику OpenAI:

```
import openai
class OpenAIClient:
def __init__(self, openai_key: str):
openai.api_key = openai_key
def analyze_text(self, text: str, level: str, language: str):
prompt = (
f"You are an experienced language teacher."
f"Translate some text based on candidate's proficiency level:\n{text}\n\n"
f"Explain translated text in the user's native language : {language}\n"
```

```
f"Candidate's proficiency level: {level}\n"
)
try:
response = openai.chat.completions.create(
model="gpt-4o-turbo",
messages=[
{"role": "system", "content": "You are an experienced language teacher."},
{"role": "user", "content": prompt}
], max_tokens=1500,
temperature=0.7
)
reply = response.choices[0].message.content
return {
"analysis": reply.strip()
}
except openai.OpenAIError as e:
raise RuntimeError(f"Error OpenAI API: {str(e)}")
```

В контролері зі штучним інтелектом була використана система кешування через Redis, тому перед кожним запитом формується унікальний ключ, який базується на тексті пісні, вибраній мові та рівні користувача. Якщо відповідь уже є в кеші, вона повертається одразу та не надсилає жодного запиту до моделі, якщо ж її немає, система відправляє новий запит до GPT. Отриманий результат зберігається в кеш з обмеженим терміном дії і лише після цього повертається користувачу.

Лістинг 3.6 Приклад роботи TranslatorController:

```
class TranslatorController:
def __init__(self, openai_client: OpenAIClient, redis_client: Redis):
self.openai_client = openai_client
```

```
self.redis_client = redis_client
async def get_text_explanation(self, text: str, language: str, level: str):
text_hash = sha256(text.strip().lower().encode()).hexdigest()
cache_key = f"result:{text_hash}:{language}:{level}"
cache_data = await self.redis_client.get(cache_key)
if cache_data is not None:
try:
return json.loads(cache_data)
except json.JSONDecodeError as e:
print(f"Error decoding cached data: {e}")
# Delete corrupted data from the cache
await self.redis_client.delete(cache_key)
try:
result = self.openai_client.analyze_text(
text=text,
level=level,
language=language
)
except RuntimeError as e:
raise HTTPException(status_code=400, detail=str(e))
await self.redis_client.set(cache_key, json.dumps(result), ex=3600)
return result
```

3.3 Тестування і результати роботи застосунку

Перед початком основної взаємодії із застосунком користувач повинен пройти процедуру реєстрації (рис. 3.2) або автентифікації (рис. 3.3).

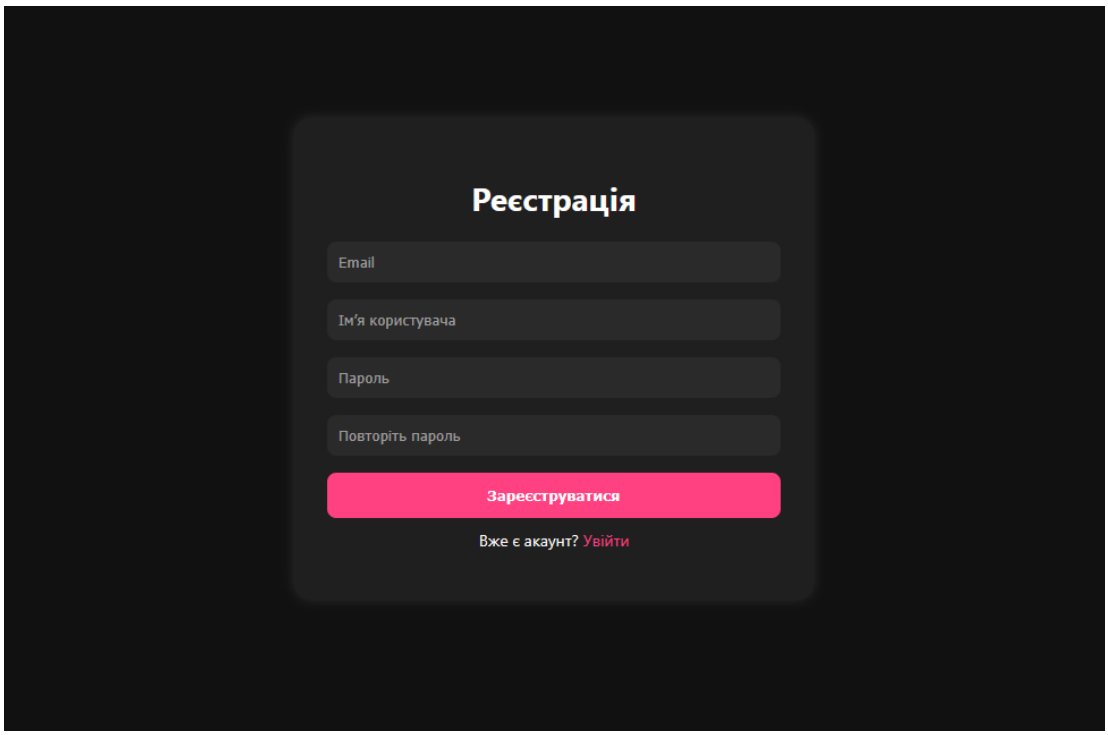

Рисунок 3.2 – Реєстрація

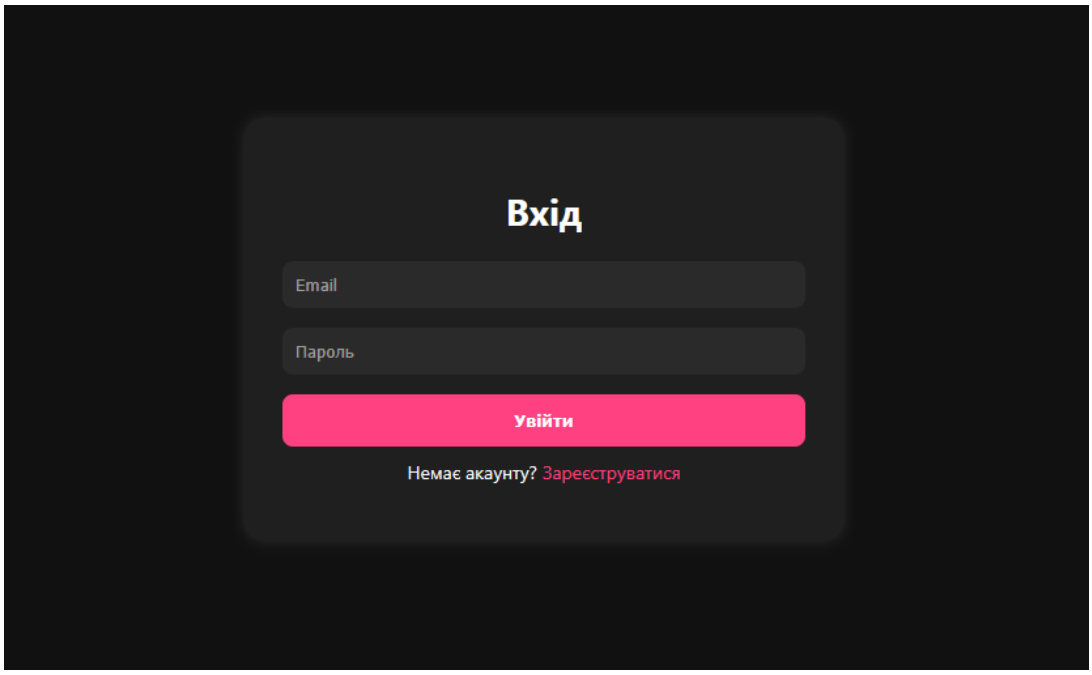

Рисунок 3.3 – Автентифікація

Після успішної автентифікації користувач автоматично потрапляє на головну сторінку вебзастосунку. У правому верхньому куті інтерфейсу з'являється індикатор входу (рис. 3.4), а саме кнопка з ім'ям облікового

запису, що свідчить про те, що система розпізнала користувача й надала йому доступ до персоналізованого функціоналу.

Рисунок 3.4 – Індикатор входу

Ця кнопка є не лише візуальним підтвердженням автентифікації, а й відкриває додаткове меню (рис. 3.5), де є можливість здійснити вихід із профілю, якщо це потрібно.

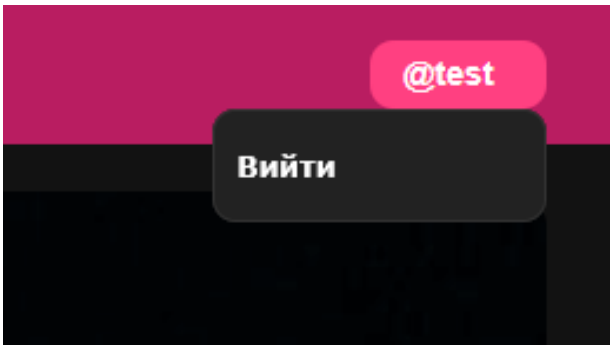

Рисунок 3.5 – Кнопка виходу

Центральним елементом головної сторінки виступає форма пошуку артиста, саме з неї починається основна взаємодія користувача з платформою. Достатньо ввести ім’я улюбленого виконавця і натиснути кнопку «Знайти», щоб система надіслала запит до сторонніх сервісів та завантажила інформацію про артиста (рис. 3.6).

Рисунок 3.6 – Форма пошуку артиста

У разі успішного пошуку користувач бачить картку артиста з фотографією, ім’ям, з псевдонімами, кількістю підписників, рівнем популярності та жанром, а також доступні кнопки для переходу на сторінки артиста в соціальних мережах, таких як Instagram та Twitter (або X) (рис. 3.7).

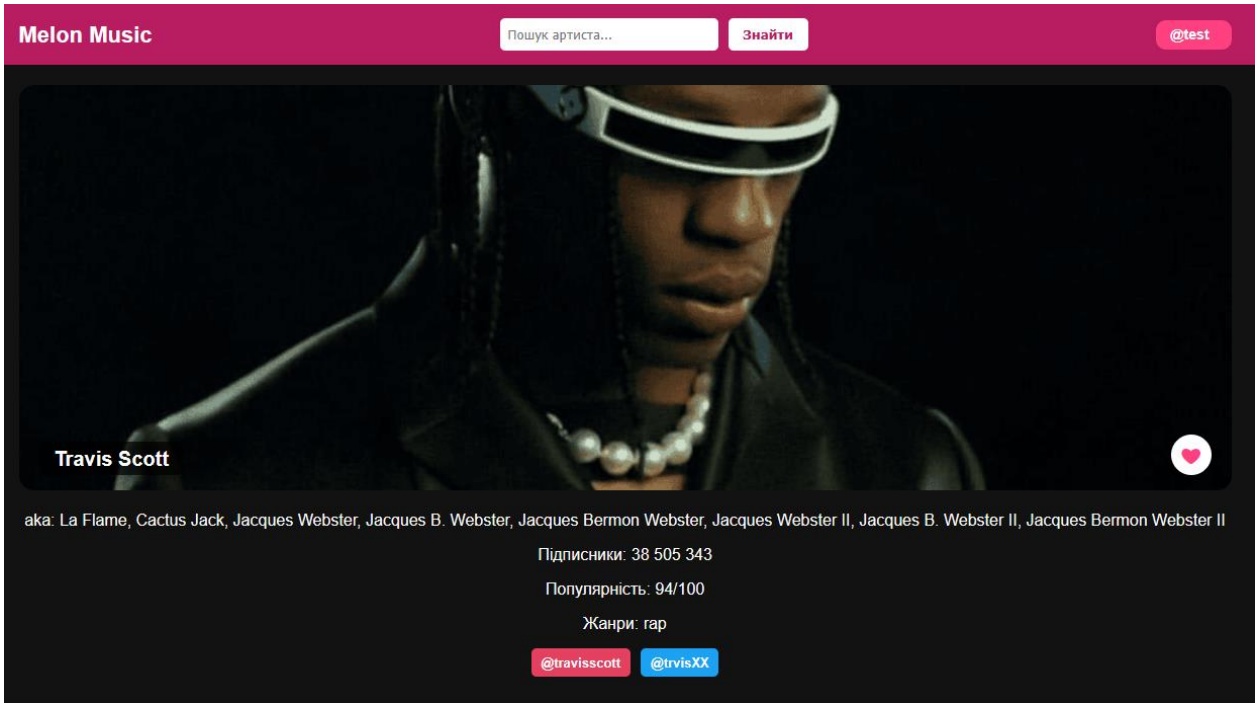

Рисунок 3.7 – Інформація про артиста

Нижче на сторінці відображаються найпопулярніші треки виконавця у вигляді карток, кожна з них містить обкладинку, назву композиції та ім’я виконавця, усе оформлено так, щоб користувачу було зручно переглядати й швидко орієнтуватися у списку (рис. 3.8).

У самому кінці сторінки користувач бачить блок зі схожими артистами за жанром. Цей елемент створений для того, щоб розширити музичний досвід і запропонувати додаткові варіанти для прослуховування. Система автоматично визначає жанр обраного виконавця та на його основі формує список схожих артистів (рис. 3.9).

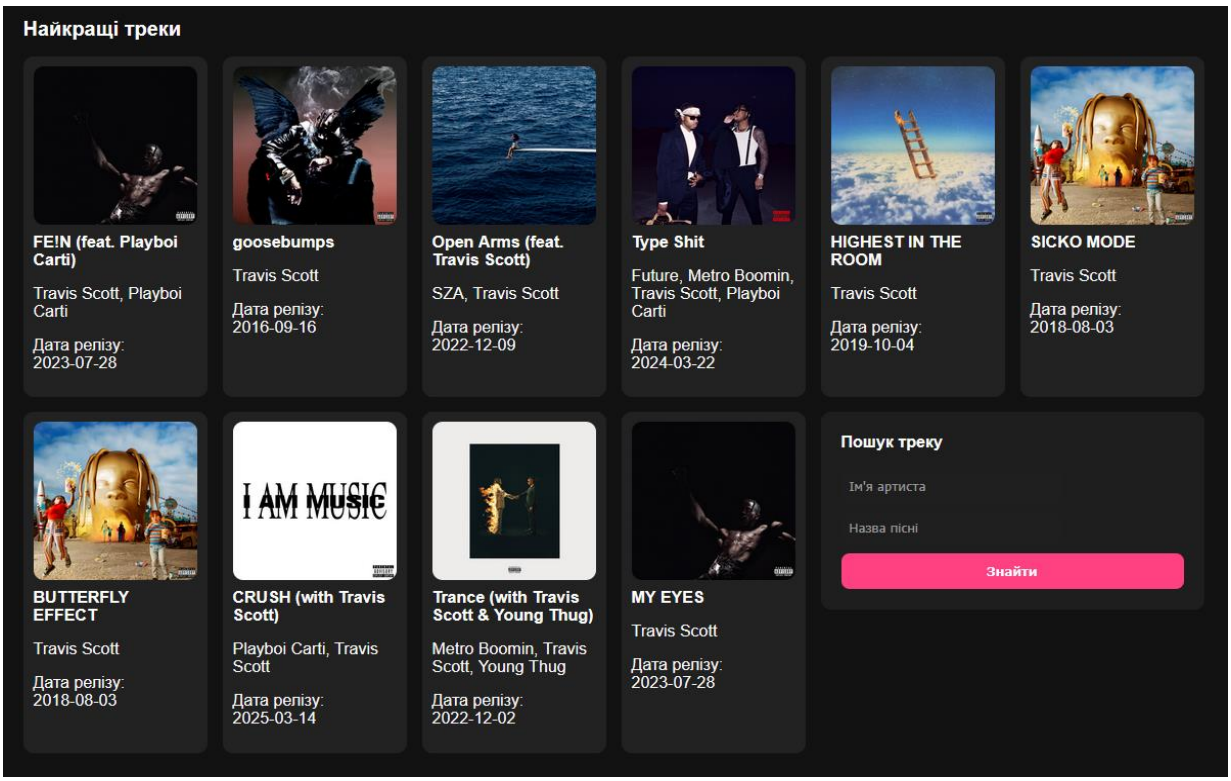

Рисунок 3.8 – Список найкращих треків

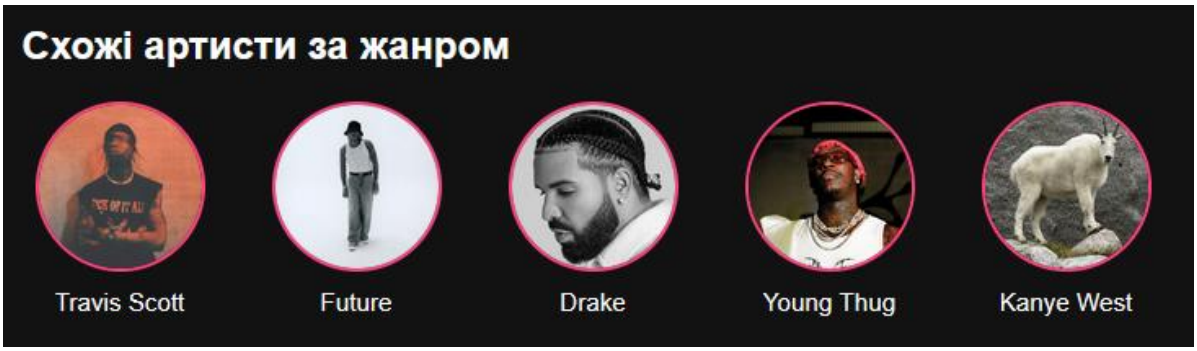

Рисунок 3.9 – Список схожих за жанром артистів

У тих випадках, коли система не змогла знайти схожих артистів за жанром, користувачеві відображається відповідне повідомлення (рис. 3.10).

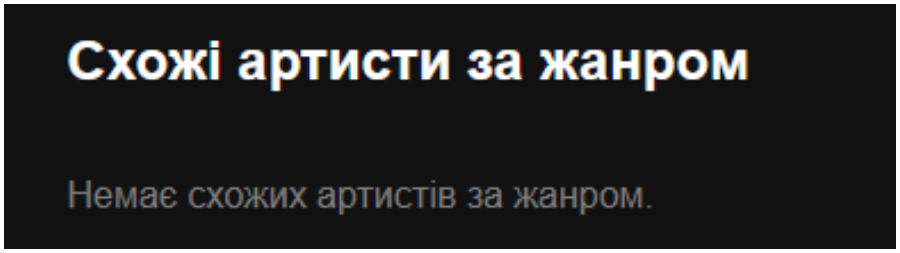

Рисунок 3.10 – Повідомлення

Сайдбар у вебзастосунку виконує роль панелі вподобань (рис. 3.11), він постійно доступний для користувача та слугує зручним елементом для швидкої навігації до улюблених артистів і треків. Він візуально оформлений як окремий блок праворуч на сторінці, містить дві основні категорії, а саме «Улюблені артисти» та «Улюблені треки».

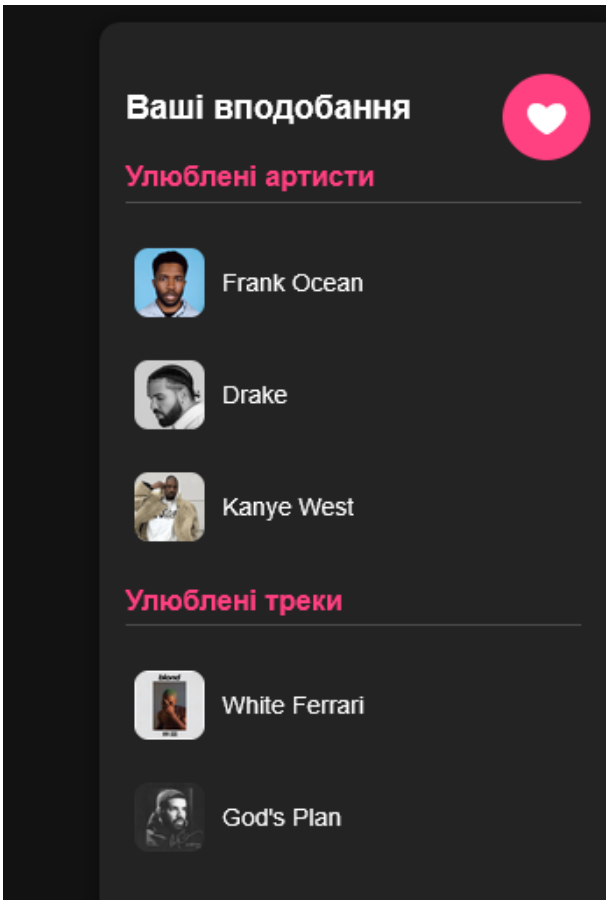

Рисунок 3.11 – Панель вподобань

Користувач може додати артиста або трек до списку вподобаних, натиснувши на іконку у формі сердечка, яка розміщена на сторінці виконавця або пісні. Після натискання ця кнопка змінює колір на рожевий (рис. 3.12) і вибраний елемент одразу з’являється на панелі, що дозволяє швидко повертатися до улюбленого контенту без потреби щось заново шукати. Якщо ж вподобання користувача змінюються та він більше не хоче бачити той чи інший об’єкт у своєму списку, його так само легко можна прибрати одним кліком.

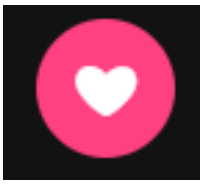

Рисунок 3.12 – Зміна кольору

Після того, як користувач обрав артиста та ознайомився з основною інформацією, наступним логічним кроком стає глибше занурення в конкретну композицію. Для цього достатньо натиснути на картку з треком або скористатися формою пошуку, щоб знайти пісню, якої немає у списку.

На цій сторінці користувач бачить повну інформацію про композицію, яка розміщена у верхній частині (рис. 3.13).

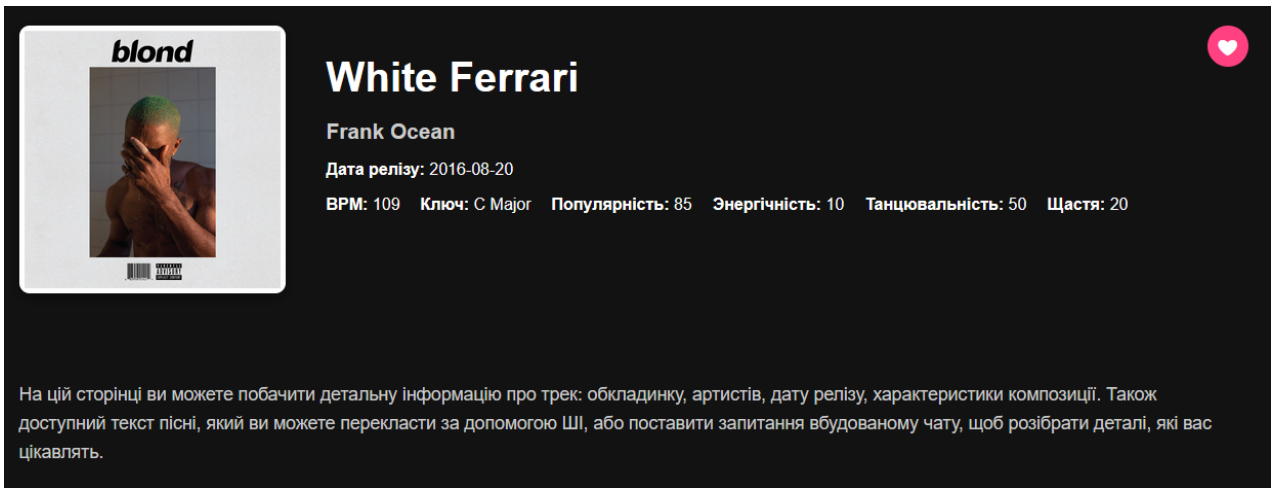

Рисунок 3.13 – Інформація про трек

Трохи нижче розміщено основний блок з текстом пісні, користувач може ознайомитися з повним змістом треку з чітким розділенням на куплети, приспіви та інші частини (рис. 3.14).

Праворуч від тексту знаходиться форма, яка дозволяє перекласти та пояснити пісню за допомогою штучного інтелекту. Користувач обирає мову перекладу, вказує рівень складності, натискає кнопку і система надсилає запит до OpenAI. Отриманий переклад одразу з’являється на сторінці з урахуванням зазначених параметрів (рис. 3.15).

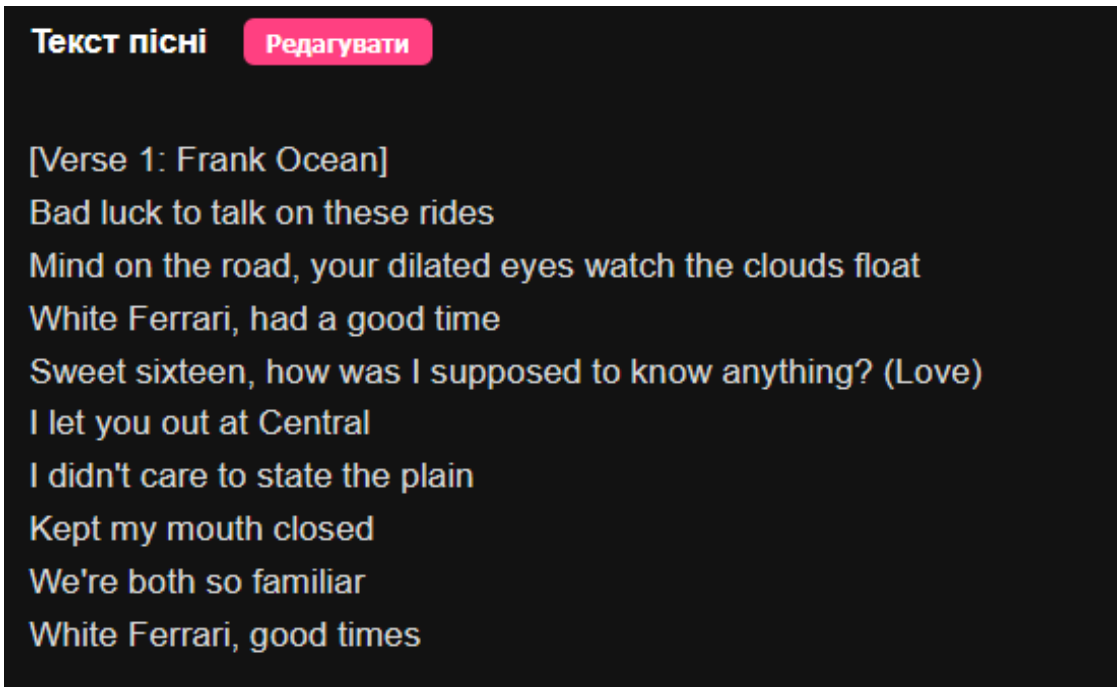

Рисунок 3.14 – Блок тексту пісні

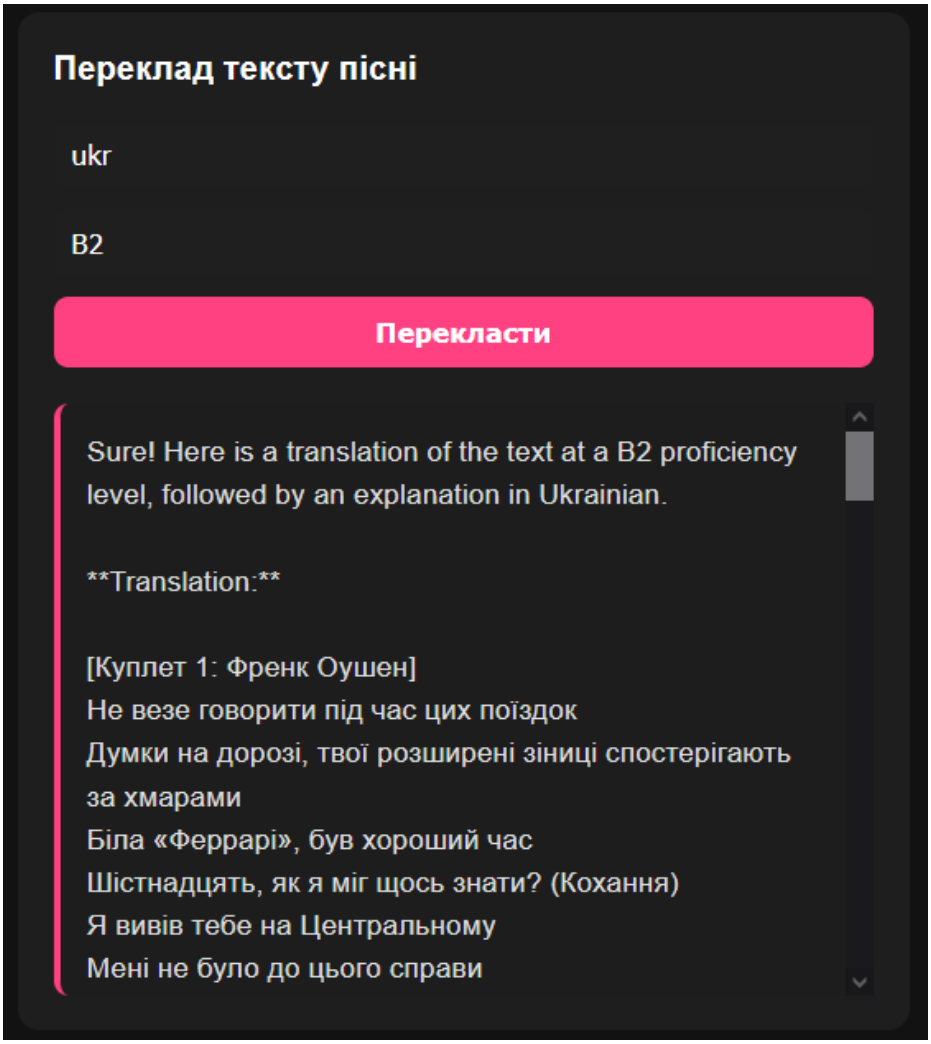

Рисунок 3.15 – Інтерактивна форма для перекладу

Окрім перекладу, на сторінці треку доступний чат зі штучним інтелектом (рис. 3.16), який дозволяє у вільній формі поставити будь-яке запитання щодо тексту пісні, зокрема питання про граматику, значення слів, контекст чи стилістику, усе відбувається інтерактивно і без потреби залишати сторінку.

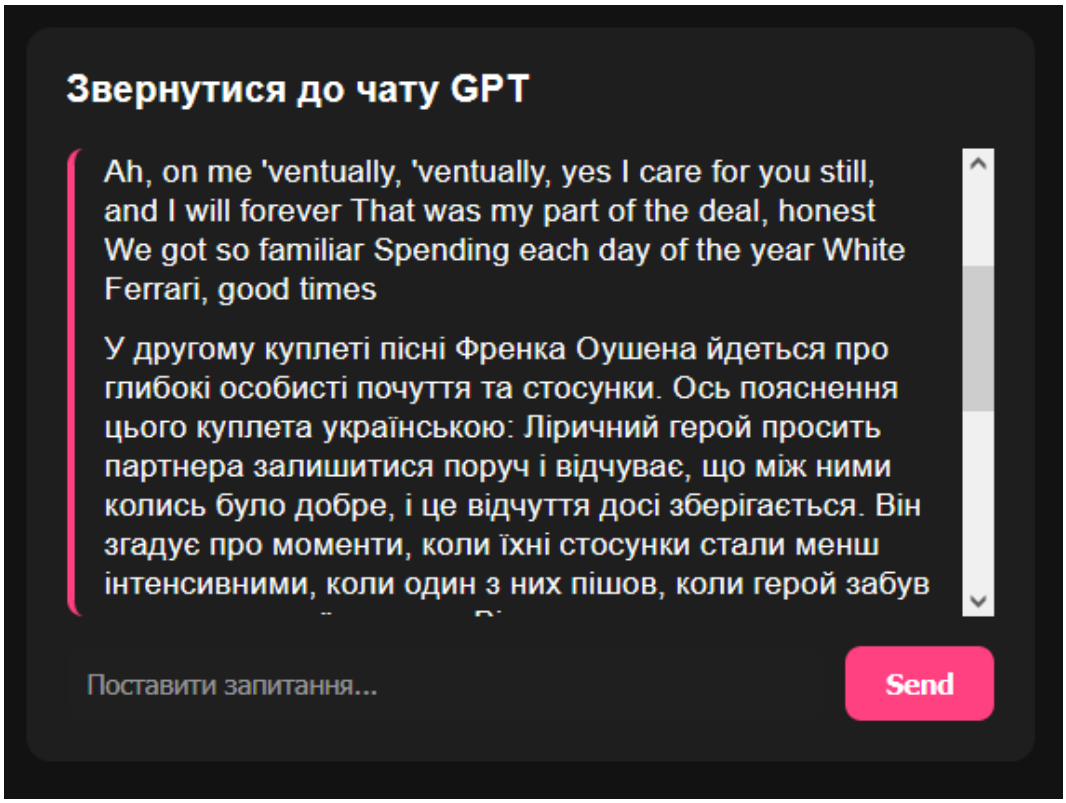

Рисунок 3.16 – Чат зі штучним інтелектом

У нижній частині сторінки інтегрований візуальний музичний плеєр від Spotify (рис. 3.17), який дозволяє прослухати всю композицію безпосередньо в інтерфейсі.

Рисунок 3.17 – Музичний плеєр

3.4 Перспективи для подальшого розвитку

Попри вже реалізований функціонал, застосунок має великий потенціал для подальшого вдосконалення [22 – 27]. У перспективі планується розширення як технічних можливостей, так і зручності користування з акцентом на гнучкість, доступність і персоналізацію. Нижче наведено основні напрямки, які можуть стати основою наступних оновлень.

Одним із важливих кроків у розвитку платформи може стати перехід на мікросервісну архітектуру, тобто поділ системи на окремі незалежні сервіси, де кожна відповідає за свою ділянку роботи. Такий підхід може зробити платформу гнучкішою, спростити підтримку, пришвидшити розробку нових можливостей і дозволити масштабувати все поступово, без потреби змінювати всю систему одразу.

Наступним логічним етапом розвитку стане адаптація інтерфейсу під мобільні пристрої. Ідеться як про розробку повноцінного мобільного клієнта, так і про впровадження адаптивної верстки для смартфонів і планшетів, щоб забезпечити комфортне користування незалежно від формату екрана.

Ще важливо впровадити підтримку мультимовного інтерфейсу, що дасть користувачеві змогу працювати з платформою рідною мовою, зробить її доступнішою для більшої кількості людей.

Крім того, варто реалізувати можливість залишати коментарі до треків, адже це відкриє додатковий простір для взаємодії користувачів, дозволить ділитися враженнями, обговорювати зміст лірики, радити щось іншим і поступово формувати справжню спільноту навколо платформи.

У подальшому розвитку важливо зробити систему ролей, зокрема додати адміністратора, який матиме доступ до розширених можливостей керування платформою. Він зможе редагувати тексти пісень у випадках, коли отримана з API інформація виявиться неповною або неточною, також зможе стежити за коментарями, модерувати обговорення, видаляти небажаний

контент, реагувати на скарги користувачів і при потребі обмежувати доступ до певних функцій.

Окремо планується додати функцію індивідуального словника, щоб користувач міг зберігати нові слова та фрази, отримані під час роботи з текстами пісень, додавати до них переклади, пояснення та приклади вживання. Таким чином поступово можна створити власну мовну базу для повторення та самонавчання.

Серед додаткових функцій є можливість голосових запитів, яка здатна зробити платформу ще більш інтерактивною. Застосунок зможе обробляти звернення до штучного інтелекту не лише у текстовій формі, а й через голос, що особливо зручно на мобільних пристроях. Для реалізації цього планується інтеграція з Web Speech API або подібними інструментами.

Усі ці покращення не лише розширюють функціонал, а й формують цілісне середовище для мовної практики через музику.

ВИСНОВКИ

У рамках кваліфікаційної роботи було розроблено музичний вебзастосунок, що дає змогу користувачам вивчати іноземні мови шляхом інтерактивної взаємодії з музичним і текстовим контентом. Основна увага приділялася створенню інструменту, який поєднує функції прослуховування треків, перегляду текстів пісень, їх перекладу та аналізу з використанням технологій штучного інтелекту.

Під час роботи було проведено аналіз популярних сервісів і вже на цьому етапі стало помітно, що більшість із них мають певні обмеження, в них бракує перекладу, немає пояснень складних фраз, лексики, іноді відсутня гнучкість, яка потрібна саме для навчального процесу. Це дало змогу краще зрозуміти, які функції справді важливі для користувача, допомогло чітко розставити пріоритети та обґрунтувати необхідність створення власного застосунку з додатковими можливостями для мовного навчання.

У процесі реалізації було розроблено архітектуру застосунку, створено інтерфейс користувача, налагоджено взаємодію з зовнішніми ресурсами та впроваджено модуль штучного інтелекту для перекладу та аналізу текстів. Вся логіка побудована з урахуванням зручності, простоти використання та адаптації під потреби конкретного користувача.

Результати тестування показали, що розроблене рішення є функціональним, зручним у використанні та справді може бути корисною як навчальний інструмент. Застосунок дозволяє не лише переглядати тексти та прослуховувати музику, а й глибше розуміти її зміст, що значно покращує сприйняття іноземної мови.

Реалізація проєкту стала прикладом того, як сучасні цифрові інструменти можуть працювати не лише для розваги, а й для навчання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. John, A. (2025). Exploring the impact of artificial intelligence on language acquisition, linguistic development, and language use: A case study from India. In *Forum for Linguistic Studies* (Vol. 7, No. 3, pp. 1104-1117).

2. Django introduction. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction (дата звернення 07.02.2025).

3. What is Flask? URL: https://www.educative.io/courses/flask-develop-web-applications-in-python/what-is-flask (дата звернення 07.02.2025).

4. History, Design and Future. URL: https://fastapi.tiangolo.com/history-design-future/ (дата звернення: 07.02.2025).

5. Django vs FastAPI: Which is the Best Python Web Framework. URL: https://blog.jetbrains.com/pycharm/2023/12/django-vs-fastapi-which-is-the-best-python-web-framework/ (дата звернення 07.02.2025).

6. Солохін, А. Є. (2024). Дослідження ефективності використання фреймворків веб-розробки на продуктивність веб-додатку: порівняльний аналіз Django, Flask, та FastAPI.

7. Modular Monolith Architecture. URL: https://www.linkedin.com/pulse/modular-monolith-blueprint-bogdan-veliscu-z5swe/ (дата звернення: 13.03.2025).

8. Бутрин, Л. (2015). Реляційна модель бази даних. *Збірник тез VIII всеукраїнської студентської науково-технічної конференції „Природничі та гуманітарні науки. Актуальні питання “*, *1*, 119-120.

9. Руденко Д.О., Бондар В.О. (2020). Огляд можливостей використання стратегій об’єктно-орієнтованого маппінгу для зіставлення сутностей при розробці web додатків. Матеріали III Міжнародної науково-практичної конференції «Priority Directions of Science and Technology Development» Київ, Україна.с.377-380.

10. Valeria, B., Hryhorii, K., Diana, R., & Vyacheslav, L. (2023). Phase Portrait Models as a Tool for Analyzing Banking Activities.

11. Shafronenko, A. Y., Bodyanskiy, Y. V., & Rudenko, D. A. (2020). Adaptive Neuro-Fuzzy Methods for Distorted Data Clustering.

12. Шафроненко, А. Ю., Бодянський, Є. В., & Руденко, Д. О. (2023). Модифікований рекурентний метод достовірної нечіткої кластеризації з використанням оптимізаційної процедури на основі косяків риб. Системи обробки інформації, (1 (172)), 92-96.

13. HTTP request methods. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods (дата звернення: 13.03.2025).

14. Surwase, V. (2016). REST API modeling languages-a developer's perspective. *Int. J. Sci. Technol. Eng*, *2*(10), 634-637.

15. Client-side caching introduction. URL: https://redis.io/docs/latest/develop/clients/client-side-caching/ (дата звернення: 18.02.2025).

16. Da Silva, M. D., & Tavares, H. L. (2015). *Redis Essentials*. Packt Publishing Ltd.

17. Палєга, Р. В., Карпенко, Н. В., & Герасимов, В. В. (2024). ОСОБЛИВОСТІ АВТОРИЗАЦІЇ ЗА ДОПОМОГОЮ JWT ТОКЕНІВ. *Інформаційні технології і автоматизація–2024/Матеріали XVII міжнародної науково-практичної конференції. Одеса, 31 жовтня-1 листопада 2024 р.-Одеса, Видавництво ОНТУ, 2024 р.–847 с. Збірник включає матеріали доповідей учасників конференції, які об'єднані за тематичними напрямками конференції.*, 208.

18. Bucko, A., Vishi, K., Krasniqi, B., & Rexha, B. (2023). Enhancing jwt authentication and authorization in web applications based on user behavior history. *Computers*, *12*(4), 78.

19. Андрєєва, А. Ю., & Руденко, Д. О. (2022, October). Як блокчейн токенізація змінює світ. In The 2 nd International scientific and practical conference "Science and innovation of modern world"(October 26-28, 2022) Cognum Publishing House, London, United Kingdom. 2022. 948 p. (p. 233).

20. Bodyanskiy, Y. V., Shafronenko, A., & Rudenko, D. (2019). Online Neuro Fuzzy Clustering of Data with Omissions and Outliers based on Completion Strategy. In CMIS (pp. 18-27).

21. Golinelli, M., Arshad, E., Kashchuk, D., & Crispo, B. (2023, November). Mind the CORS. In *2023 5th IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)* (pp. 213-221). IEEE.

22. Abu-Jassar, A., Rudenko, D., & Abdalla, H. (2024). Digital medical image as an object of processing and analysis. Multidisciplinary Journal of Science and Technology, 4(1), 28-35.

23. Pomazan, V., Tvoroshenko, I., & Gorokhovatskyi, V. (2023). Development of an application for recognizing emotions using convolutional neural networks, International Journal of Academic Information Systems Research, 7(7), pp. 25-36.

24. В.О. Гороховатський, Д.О. Руденко, Т.О. Сірик. (2019) Дослідження системи ієрархічних ознак при блочному поданні опису у складі множини ключових точок зображення

25. Руденко, Д. О., & Кухарчук, В. А. (2022, October). Сучасні напрямки застосування методів супер роздільності зображення. In The 8 th International scientific and practical conference “Modern research in world science”(October 29-31, 2022) SPC “Sci-conf. com. ua”, Lviv, Ukraine. 2022. 1828 p. (p. 393).

26. V. Lyashenko, D. Rudenko, (2021) Modeling Deformation of Spur Gear.

27. Руденко Д.О., Постніков А.В. (2021) Аналіз загальної класифікації та структури в event-індустрії.