НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»
Завідувач кафедри
_____Наталія АУШЕВА
“___”_____2025 р.

Дипломна робота
на здобуття ступеня бакалавр

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “Веб-платформа для прослуховування та розповсюдження музичних файлів”

Виконав: студент 4 курсу, групи ТР-15
ФУНДАМЕНТ Даніїл Дмитрович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник _доцент, к.т.н. Ірина МИХАЙЛОВА _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент _доцент, к.т.н. Артур РАЧИНСЬКИЙ _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Н.контроль асистент Антон ПАСІЧНЮК _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА
(підпис)
“___” _____2025 р.

ЗАВДАННЯ
на дипломну роботу студенту

Фундаменту Даніїлу Дмитровичу
(прізвище, ім’я, по батькові)

1. Тема роботи “Веб-платформа для прослуховування та розповсюдження музичних файлів ”
Науковий керівник Михайлова Ірина Юріївна, к.т.н,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “____” _____ 2025 року № _____
2. Термін подання студентом роботи 09.06.2025
3. Вихідні дані до роботи мова програмування TypeScript, фреймворк Nest.js, СУБД PostgreSQL, бібліотека React.js, середовище розробки Cursor
4. Перелік питань, які потрібно розробити _____
1) провести аналіз серед наявних аналогів

2) визначити головні функції веб-платформи для прослуховування та розповсюдження музичних файлів

3) аргументувати вибрані інструменти, алгоритми та технології для реалізації веб-платформи

4) побудувати архітектуру програмного забезпечення та бази даних

5) створити адаптивний користувацький інтерфейс

6) представити процес застосування веб-платформи

5. Орієнтований перелік ілюстративного матеріалу актуальність, мета, завдання, засоби реалізації, діаграма прецедентів, архітектура веб-платформи, схема бази даних, демонстрація взаємодії користувача з платформою.

6. Дата видачі завдання 13.09.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	02.09.24-12.09.2024	
2.	Аналіз методів та засобів розв’язання задачі	15.11.24-12.12.2024	
3.	Розробка архітектури та загальної структури системи	20.12.2024-20.01.2025	
4.	Розробка окремих підсистем	27.03.2025-17.04.2025	
5.	Програмна реалізація системи	18.04.2025-05.05.2025	
6.	Оформлення пояснювальної записки	06.05.2025-14.05.2025	
7.	Захист програмного забезпечення	15.05.2025	
8.	Передзахист	27.05.2025	
9.	Захист	16.06.2026	

Студент _____ ФУНДАМЕНТ Д.Д.
(підпис) (прізвище та ініціали)

Керівник _____ МИХАЙЛОВА І.Ю.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота виконана на 70 сторінках, містить 26 ілюстрацій, 1 додаток, 20 джерел у переліку посилань.

Мета роботи: розробка веб-платформи для прослуховування та розповсюдження музичних файлів з можливістю генерування тексту пісень за допомогою штучного інтелекту.

Методи та засоби: мова програмування TypeScript, фреймворк Nest.js для побудови серверної частини з модульною архітектурою, об’єктно-реляційна проекція Sequelize для роботи з базою даних, хмарне сховище AWS S3 для зберігання файлів, сервіс AWS RDS для керування базами даних, клієнтська бібліотека React.js, засоби інтерфейсної стилізації Material UI та Tailwind CSS, а також мовна нейромережа AssemblyAI для обробки аудіофайлів.

Результат: створена масштабована веб-система з можливістю авторизації, керування файлами, прослуховування аудіо, транскрибування мовлення у текст, виведення розпізнаного тексту у зручному вигляді, а також з надійною інтеграцією з хмарними сервісами для зберігання та обробки даних.

Ключові слова: ВЕБ-ПЛАТФОРМА, МУЗИЧНА ПЛАТФОРМА, NESTJS, REACT, SEQUELIZE, POSTGRES, S3, RDS, MUI, ASSEMBLYAI.

ABSTRACT

The diploma project consists of 70 pages and contains 26 illustrations, 1 appendix, and 16 bibliography references.

The purpose of this work is development of a web platform for listening to and distributing music files with the ability to generate lyrics using artificial intelligence.

Methods and tools: TypeScript programming language, the Nest.js framework for building a server side with a modular architecture, the Sequelize object-relational projection for working with the database, AWS S3 cloud storage for storing files, the AWS RDS service for managing databases, the React.js client library, the Material UI and Tailwind CSS interface styling tools, and the AssemblyAI speech neural network for processing audio files.

Result: Developed a scalable web-based system with the ability to authorize, manage files, listen to audio, transcribe speech into text, display recognized text in a convenient form, and with reliable integration with cloud services for data storage and processing.

Keywords: WEB-PLATFORM, MUSIC PLATFORM, NESTJS, REACT, SEQUELIZE, POSTGRES, S3, RDS, MUI, ASSEMBLYAI.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ

Back-end – серверна частина застосунку, яка відповідає за обробку запитів, логіку бізнес-процесів та взаємодію з базою даних і зовнішніми сервісами.

Front-end – клієнтська частина застосунку, що забезпечує взаємодію з користувачем через графічний інтерфейс.

Nest.js – прогресивний серверний фреймворк для Node.js, який базується на TypeScript та дозволяє будувати масштабовані та модульні серверні застосунки.

React – JavaScript-бібліотека для створення динамічних інтерфейсів користувача на основі компонентів.

.env – конфігураційні файли середовища, у яких зберігаються змінні для налаштування серверної та клієнтської частини застосунку.

npm – система керування пакетами для JavaScript, яка дозволяє встановлювати та запускати залежності проєкту.

API – У контексті веб-платформи це спосіб, яким клієнтська частина взаємодіє з серверною, надсилаючи запити та отримуючи відповіді у стандартизованому форматі (зазвичай JSON).

HTTP – це повідомлення, яке веб-клієнт надсилає на веб-сервер для отримання або надсилання даних.

ORM – це технологія або бібліотека, яка дозволяє працювати з базами даних за допомогою об'єктів у звичній мові програмування.

Фреймворк – це набір готових інструментів, бібліотек та правил, який допомагає розробникам швидше і зручніше створювати додатки.

UI – користувацький інтерфейс.

AWS – Amazon Web Services

ВСТУП

В епоху цифрових технологій музична індустрія стає все доступнішою для ширшої аудиторії. Творці, яким раніше потрібне було студійне обладнання або професійна підтримка, тепер можуть втілити свої творчі ідеї в життя за допомогою онлайн-інструментів. У цьому контексті створення інноваційних платформ, що поєднують практичність, інтелектуальну допомогу та функціональну гнучкість, стає надзвичайно важливим.

Метою цієї дисертації є розробка сучасної веб-платформи для початківців-композиторів, які хочуть створювати тексти пісень. Ключовою особливістю системи є її інтеграція з AssemblyAI, голосовим штучним інтелектом, який автоматично генерує тексти пісень із завантажених файлів. Такий підхід значно спрощує творчий процес, не відволікаючи особливих уваги на додаткові описи пісень.

Платформа реалізована як повноцінний веб-застосунок, що складається з клієнтського компонента на базі React.js та серверного компонента на базі Nest.js. Для зберігання даних використовується хмарна база даних Amazon RDS (PostgreSQL), а для медіафайлів – хмарне сховище AWS S3. Обмін даними між фронтендом та сервером здійснюється через API, розроблений для безпеки та продуктивності. Крім того, реалізовано систему реєстрації, авторизації, особистого облікового запису користувача та перегляду історії згенерованих текстів.

Отримана веб-платформа є інноваційним та корисним інструментом для авторів-початківців, які шукають творчої підтримки. Вона поєднує інтуїтивно зрозумілий інтерфейс, можливості штучного інтелекту та сучасну архітектуру, що забезпечує стабільну роботу, гнучкість та перспективи зростання.

1 РОЗРОБКА ВЕБ-ПЛАТФОРМИ ДЛЯ ПРОСЛУХОВУВАННЯ ТА РОЗПОСЮДЖЕННЯ МУЗИЧНИХ ФАЙЛІВ

Для суспільства споживати музичний контент стало рутиною. Майже кожен не може уявити своєї подорожі у потязі без прослуховування пісень, саме тому і є постійна потреба у музичних платформах. У сучасних умовах музичні веб-платформи стали звичним каналом споживання аудіоконтенту. Проте основна маса таких платформ зводиться до стандартного функціоналу: стрімінг треків, створення плейлистів, рекомендаційні системи. Водночас вони мають низку обмежень: обмежений функціонал для авторів, складна система завантаження власного контенту, недостатня гнучкість інтерфейсу, відсутність цікавих нововведень та нових підходів.

1.1 Постановка задачі розробки веб-платформи для прослуховування та розповсюдження музичних файлів

Метою цієї дипломної роботи є розробка сучасної веб-платформи для авторів-початківців, яка дозволить їм генерувати тексти пісень за допомогою штучного мовного інтелекту, зберігати створений контент, переглядати історію взаємодії та пропонувати інтуїтивно зрозумілий інтерфейс користувача. Ця система має стати корисним інструментом для людей з невеликим досвідом написання текстів, але які хочуть повною мірою розкрити свій творчий потенціал за допомогою інноваційних технологій.

Для досягнення мети необхідно проаналізувати бізнес-сферу та існуючі аналогічні платформи, визначити ключові функціональні вимоги до системи, вибрати відповідні інструменти та технології для розробки веб-застосунку, спроектувати

архітектуру клієнтської та серверної частин, реалізувати функціональність генерації текстів пісень за допомогою штучного інтелекту, забезпечити збереження результатів та авторизацію користувачів, а також протестувати продуктивність платформи.

Ця кваліфікаційна робота спрямована на створення сучасної музичної веб-платформи, яка дозволить користувачам легко завантажувати, слухати та керувати власним аудіоконтентом. Необхідність розробки такої системи пояснюється загальною одноманітністю існуючих музичних сервісів: більшість із них пропонують схожий досвід, тим самим обмежуючи індивідуальні потреби незалежних артистів, слухачів чи освітніх ініціатив. Крім того, такі платформи, як Spotify або YouTube Music, зосереджені переважно на потоковому контенті від великих звукозаписних лейблів, що обмежує можливості артистів-новачків.

Розроблена система повинна забезпечувати легке завантаження аудіофайлів, автоматичне розпізнавання голосу треків за допомогою штучного інтелекту, створення текстових транскрипцій треків, а також можливість створення сторінок візуалізації контенту. Особливу увагу було приділено вибору технологій генерації тексту з мовлення: серед поточних та існуючих рішень було обрано AssemblyAI – сервіс, що поєднує високу точність розпізнавання та легкість інтеграції через REST API.

Головна аудиторія – музиканти початківці, які прагнуть ділитись свою творчістю зі світом без значних вкладень та знайти перших поціновувачів. Також люди, які хочуть знайти для себе нових цікавих виконавців і послухати музику без вкладень.

Користувачі повинні мати можливість створити власні профілі за допомогою яких вони зможуть взаємодіяти з платформою. У кожного користувача має бути змога переглядати загальні списки останніх створених пісень та музичних збірок від усіх користувачів платформи, додавати певні пісні та музичні збірки до списків збереженого, які дозволяють мати швидкий доступ до них у майбутньому. Також

кожен користувач має мати змогу переглядати профілі інших користувачів і переглядати завантажені ними пісні та створені музичні збірки.

Вхідні дані: параметри запиту, введені користувачем у систему, включаючи вкладені музичні файли та обкладинки.

Вихідні дані: згенеровані тексти пісень, що відображаються в інтерфейсі користувача, список збережених композицій, інформація з профілю користувача, повідомлення про успішні або невдалі дії.

1.2 План розв’язання задачі

Розробка нової веб-платформи вимагатиме комплексного підходу, включаючи вирішення ключових проблем, таких як завантаження, зберігання та потокове передавання аудіо, асинхронна обробка запитів на стороні сервера, захист персональних даних користувачів та створення зручного та швидкого інтерфейсу.

Першим кроком буде вирішення проблеми зберігання та потокового передавання аудіо- та графічних файлів. Буде впроваджено механізм, який дозволить будь-якому користувачеві постійно отримувати доступ до завантажених медіафайлів. Щоб уникнути обмежень масштабування, файли не зберігатимуться безпосередньо на сервері. Натомість буде впроваджено хмарне сховище AWS S3, що забезпечить надійне та масштабоване сховище для великих обсягів даних. Доступ до файлів буде здійснюватися через окремо згенеровані посилання, що зберігаються в базі даних. Взаємодія з файлами буде можливою через HTTP-запити, такі як GET (отримання) або PUT (оновлення).

Наступним кроком буде впровадження потокового аудіоплеєра, який працюватиме без затримки та буде сумісним з різними платформами. Це буде досягнуто за допомогою Web Audio API та HTML5 audio. Ці інструменти дозволять

створити легкий та функціональний аудіоплеєр без необхідності підключення сторонніх бібліотек, з підтримкою буферизації та швидкого перемотування треків.

Для забезпечення безпеки даних AWS RDS, хмарний сервіс управління базами даних, забезпечить високу доступність та надійність. Також будуть налаштовані групи безпеки, що обмежують доступ до бази даних за IP-адресою або віртуальною приватною мережею (VPC). Для реалізації безпечної авторизації будуть використовуватися токени Json Web Token, що дозволяють доступ до облікових даних лише протягом терміну їх дії. Паролі користувачів будуть хешуватися за допомогою бібліотеки bcrypt.

Ще одним важливим аспектом стане реалізація асинхронної обробки запитів на сервері. Це дозволить ефективно працювати з великими обсягами даних, уникати блокування потоків та зменшити час відповіді для кінцевого користувача, саме тому для реалізації серверу буде обрано Node.js.

Клієнтський інтерфейс платформи буде реалізовано за допомогою React.js. Завдяки віртуальному DOM ця бібліотека забезпечить створення високопродуктивних та динамічних інтерфейсів, які автоматично оновлюватимуть лише змінені частини сторінки, без повного перезавантаження, тим самим підвищуючи загальну ефективність роботи застосунку.

Ключовим функціональним блоком буде реалізація можливості генерації текстів пісень з аудіофайлів завдяки сервісу AssemblyAI. Цей хмарний сервіс на основі штучного інтелекту дозволить ефективно транскрибувати аудіофайли та генерувати відповідні текстові композиції без завантаження сервера. Завдяки постійним оновленням та технічній підтримці платформа залишатиметься актуальною та сучасною.

Усі ці рішення разом створять безпечну, масштабовану та інноваційну веб-платформу для авторів-початківців, яка не лише забезпечить стабільну роботу, але й виділиться на ринку завдяки своїм унікальним функціям.

1.3 Аналіз існуючих платформ

У сучасному цифровому середовищі існує багато веб-сервісів, які дозволяють слухати музику, створювати списки відтворення та відкривати для себе новий контент. Однак, коли справа доходить до більш просунутої взаємодії з аудіо (отримання текстів пісень або автоматичне створення транскрипцій), стає зрозуміло, що більшість популярних платформ мають значні обмеження. Детальний аналіз цих платформ дозволяє виділити їхні основні переваги та недоліки, а також виявити прогалини, які має заповнити розроблена система.

Spotify – один із найпопулярніших стрімінгових сервісів, що надає доступ до мільйонів музичних треків та подкастів, а також пропонує гнучку систему рекомендацій, інтеграцію із соціальними мережами та привабливий інтерфейс користувача. Однак система закрита для зовнішніх інтеграцій із сервісами транскрипції, не підтримує локальні аудіофайли у веб-версії, а доступ до текстів пісень лише частково забезпечується сторонніми сервісами, такими як Musixmatch. Схожа ситуація спостерігається з YouTube Music, продуктом Google, який дозволяє слухати музику з YouTube в аудіоформаті. Незважаючи на велику базу контенту, включаючи рідкісні кавери та живі виступи, платформа не повністю підтримує тексти пісень, не пропонує користувацьких інструментів обробки аудіо, а доступні субтитри реалізовані переважно для відеоконтенту та недоступні через публічні API [1].

На особливу увагу заслуговує платформа Genius, яка спеціалізується на текстах пісень та їх поясненнях. Genius має велику базу даних текстів пісень та пропонує API для розробників, але він не підтримує обробку нових аудіофайлів, не виконує транскрипцію та значною мірою залежить від інтеграції з такими сервісами, як Spotify або Apple Music.

Хоча музичні платформи демонструють обмеження в галузі інтерактивної обробки аудіо, сегмент технологій автоматичного розпізнавання мовлення розвивається більш динамічно. Для взаємодії з аудіофайлами Whisper від OpenAI є

одним із найперспективніших рішень завдяки відкритому коду та можливості роботи локально. Його головний недолік полягає у високій ресурсоємності та складності впровадження у виробництво без належної технічної бази. У цьому контексті особливо виділяється AssemblyAI: платформа, що спеціалізується на обробці аудіо за допомогою штучного інтелекту. Він пропонує інтуїтивно зрозумілий REST API, підтримує транскрипцію довгих аудіофайлів, пунктуацію, тематичну сегментацію тексту та, найголовніше, постійну доступність завдяки постійному доступу до своїх серверів.

Аналіз дозволяє нам зробити кілька ключових висновків. По-перше, жодна з існуючих музичних платформ не пропонує повної свободи роботи з власними аудіофайлами, не підтримує автоматичну транскрипцію та не дозволяє здійснювати поглиблену семантичну обробку аудіо. По-друге, більшість стрімінгових сервісів закриті, мають обмежені API та не підтримують обробку даних на замовлення.

Ці фактори стали вирішальними у рішенні створити власну веб-систему, що поєднує гнучкість, сучасний дизайн, розширювану архітектуру, інтеграцію з AssemblyAI як основним механізмом транскрипції, а також підтримку повного контролю над даними користувачів. Такий підхід не лише дозволяє реалізувати необхідні функції, але й залишає систему відкритою для майбутніх розробок.

1.4 Аналіз програмних засобів

Щоб розробити стабільну, масштабовану та оптимізовану платформу потрібно дуже ретельно підібрати інструменти, які забезпечать високу продуктивність, масштабованість, надійне використання і зберігання аудіо файлів і зображень та комфорт при розробці для розробників. У межах виконання дипломного проекту було розглянуто низку найпопулярніших та найефективніших технологій як для реалізації серверу так і для клієнтської частини.

Для хостингу бази даних та усіх існуючих файлів на стороні було розглянуто різноманітні провідні хмарні платформи як: AWS, Google Cloud та Azure. Усі ці три платформи мають доволі схожу за структурою рішення та пропозиції для розробників для зберігання даних, утримання баз даних, створення власних серверів. Але AWS було обрано саме через те що він себе зарекомендував надійною та безпечною платформою, тому що він є найстарішим постачальником різноманітних хмарних рішень на ринку, а також надає гнучку можливість налаштування безпеки на вашому персональному обліковому записі. Наприклад, що вам потрібно надати доступ розробникам до вашого акаунту, щоб вони могли використовувати його для розробки, замість того щоб надавати свої дані для входу у вас є можливість створити окремого IAM користувача, якому є можливість надати певні рівні доступів до певних сервісів на платформі.

Для розробки серверної частини веб-платформи було розглянуто такі фреймворки та бібліотеки: Django, Express.js, Nest.js. Django – це фреймворк на основі Python, у ній наявна велика екосистема, через достатньо високу кількість користувачі, але він влаштований на синхронній моделі, що дуже сильно буде псувати ефективність платформи при великій кількості користувачів одночасно. Також було прийнято рішення залишитись на екосистемі JavaScript. Express.js. – дуже гнучка бібліотека, яка дійсно покриває більшість потреб нашої платформи, але вимагає дуже багато персоналізованого налаштування перед початком роботи. Саме тому було обрано використовувати Nest.js. Цей фреймворк одразу від початку налаштований на використання TypeScript, що забезпечить гарну, зручну для інших розробників статичну типізацію, що допоможе зменшити кількість багів на етапі розробки, а також він одразу має закладену модульну структуру, яку також можна кастомізувати. Ця структура також пропонує підхід Dependency Injection, тобто метод ін’єкції залежностей, який допомагає робити структуру бізнес-логіки такою, щоб у будь який момент можна було замінити один сервіс іншим, змінивши буквально пару рядків коду [5].

Для основи клієнтської частини розглядались такі JavaScript фреймворки та бібліотеки, як: Vue, Angular та React. Angular являє собою фреймворк, в якому все йде з коробки, але налаштування проекту на початковій стадії є доволі складним та містким процесом і у нашому випадку у ньому буде не дуже зручна побудова архітектури відповідно до побажань розробника. Vue – це легкий та більше шаблонний фреймворк, який дозволяє дуже швидко будувати користувацький інтерфейс для різних платформ та додаків, натомість через це дуже поступається масштабованістю та структурованістю. Тому вибір впав саме на React. Завдяки широкій спільноті React надає можливість максимально персоналізувати його різноманітними бібліотеки, щоб підійти під будь які потреби користувачів. Також він ввів таке поняття як компонентна структура проекту, що допомогає робити компоненти для багаторазового використання на різноманітних частинах інтерфейсу [4].

Для генерації тексту на основі завантежних файлів було обрано продукт із використанням штучного інтелекту, який спеціалізується на таких задачах і має назву AssemblyAI. Для генерації тексту з файлів, які можна завантажити, було обрано продукт штучного інтелекту, що спеціалізується на цих завданнях, під назвою AssemblyAI.

Його переваги навпроти конкурентів це найвища точність транскприції, що є головною характеристикою при виборі штучного інтелекту для цілей нашої платформи, а також легке інтегрування у платформу і доступна вартість у порівнянні з аналогами.

2 ЗАСОБИ РЕАЛІЗАЦІЇ ВЕБ-ПЛАТФОРМИ ДЛЯ ПРОСЛУХОВУВАННЯ ТА РОЗПОВСЮДЖЕННЯМ МУЗИЧНИХ ФАЙЛІВ

Створення веб-платформи для безкоштовного прослуховування та розповсюдження музики є досить комплексною задачею, тому що потребує реалізації функціонального та оптимізованої серверної частини, яка зможе опрацьовувати та зберігати музичні файли, забезпечити стабільну взаємодію із штучним інтелектом для автоматичної генерації текстів пісень. Також дуже важливо створити інтуїтивно зрозумілий інтерфейс, який забезпечить максимально зручну взаємодію користувача з платформою.

Ефективна розробка сучасної веб-системи вимагає використання широкого спектру інструментів, що дозволяють автоматизувати процеси, контролювати якість коду, а також спрощувати налагодження, тестування та розгортання. У цьому розділі представлені основні інструменти, що використовуються на різних етапах створення платформи, від початкового проектування до впровадження та підтримки.

Інструменти розробки відіграють життєво важливу роль у забезпеченні стабільної та передбачуваної роботи системи. Завдяки таким інструментам, як лінтери, інтегровані середовища розробки, інструменти тестування та аналізу продуктивності, команда розробників може ефективно виявляти помилки, дотримуватися єдиного стилю кодування та швидко адаптуватися до нових вимог. Крім того, розумна організація інструментів допомагає зменшити ризики регресії, покращити якість кінцевого продукту та пришвидшити розробку нових функцій.

У цьому розділі описано основні інструменти, що використовуються для фронтенд- та бекенд-розробки, а також інструменти для аналізу, тестування, документування та співпраці над проектом.

2.1 Середовище розробки Cursor

Веб-платформу було реалізовано з використанням сучасного середовища розробки Cursor, модифікованої версії Visual Studio Code з вбудованим помічником на основі штучного інтелекту, спеціально оптимізованої для продуктивної та інтелектуально підтримуваної роботи над програмними проектами. Cursor поєднує зручність, стабільність та розширюваність класичного Visual Studio Code з новими функціями, що підвищують ефективність написання, рефакторингу та аналізу коду.

Одна з головних переваг Cursor – це його глибока інтеграція зі штучним інтелектом, що дозволяє йому генерувати код, автоматично доповнювати фрагменти, пояснювати логіку існуючих функцій та виявляти логічні помилки. Ця функція значно скорочує час, витрачений на написання повторюваного або шаблонного коду, і допомагає підтримувати високу якість проекту, зменшуючи кількість помилок навіть під час етапу розробки.

Cursor підтримує повний набір функцій Visual Studio Code, включаючи інтеграцію з Git, підтримку терміналу, систему розширень, інтеграцію з ESLint та Prettier для автоматичної перевірки та форматування коду, а також зручну підтримку TypeScript та JavaScript. Для фронтенд-розробки платформи Cursor було налаштовано з необхідними плагінами для React, Tailwind CSS, MUI та React Query, що дозволило ефективно та безперебійно використовувати технологічний стек.

Для бекенд-розробки на базі NestJS Cursor також виявився дуже ефективним: завдяки автодоповненню, підказкам типів, швидкому переходу до визначень та підтримці контексту штучного інтелекту стало можливим швидко створювати контролери, сервіси, DTO та інтегрувати сторонні бібліотеки.

Вибір Cursor як основного середовища розробки пояснюється його функціональним багатством, турботою про продуктивність, підтримкою новітніх практик програмування та інтеграцією інструментів штучного інтелекту, що особливо важливо для динамічної та гнучкої розробки складної веб-системи.

2.2 Хмарне сховище файлів AWS S3

В рамках впровадження веб-платформи для зберігання, обробки та розповсюдження мультимедійних файлів користувачів (музичні треки, обкладинки альбомів, уривки та аудіофайли для транскрипції) було обрано хмарне сховище Amazon Simple Storage Service (AWS S3). Цей сервіс, одна з ключових технологій в екосистемі Amazon Web Services, пропонує масштабоване, надійне та високопродуктивне сховище файлів із гнучким контролем доступу [7].

AWS S3 дозволяє організовувати зберігання даних у «відра», які служать ізольованими контейнерами для об’єктів, що ідеально підходить для модульної архітектури сучасних веб-застосунків. У нашій системі кожен файл, завантажений користувачем, зберігається у відповідному кошику, доступному через унікальну URL-адресу. Файли можна завантажувати або безпосередньо через серверну частину за допомогою AWS SDK, або через попередньо підписані URL-адреси, згенеровані сервером, що дозволяє клієнту безпечно завантажувати файл без передачі облікових даних.

Завдяки підтримці масштабованості та розподіленого сховища, S3 забезпечує високу доступність (стійкість до втрати об'єктів 99,999999999%, відома як "11 дев'яток") та швидку доставку контенту, незалежно від обсягу даних. Це особливо важливо для нашої платформи, де медіафайли можуть досягати великих розмірів, і користувачам потрібно мати до них швидкий доступ у будь-який час.

Керування файлами інтегровано в логіку сервера на основі NestJS, де функції завантаження, видалення та генерації підписаних посилань реалізовані за допомогою модулів AWS SDK. Крім того, для оптимізації продуктивності та безпеки реалізовано обмеження типів MIME, контроль розміру файлу та перевірку розширень перед завантаженням.

S3 також служить проміжним сховищем для аудіофайлів, завантажених до стороннього сервісу транскрипції AssemblyAI. Після завершення обробки текст

транскрипту зберігається в базі даних, а аудіофайли можна видалити або зберегти для повторного використання в майбутньому, якщо це необхідно.

Причиною вибору AWS S3 є його стабільність, висока масштабованість, широкий спектр функцій безпеки (включаючи політики IAM, шифрування об'єктів тощо) та повна інтеграція з іншими сервісами AWS, що дозволяє ефективно впроваджувати гнучку та безпечну інфраструктуру зберігання файлів для потреб сучасної веб-платформи

2.3 Хмарний сервіс AWS RDS

В рамках реалізації серверної частини веб-платформи, призначеної для зберігання структурованих даних про користувачів, пісні, альбоми, плейлисти, сеанси прослуховування та результати аудіотранскрипції, було обрано використання Amazon Relational Database Service (AWS RDS). Ця послуга надає повністю керовану інфраструктуру для роботи реляційних баз даних у хмарному середовищі з автоматичними оновленнями, масштабуванням, резервним копіюванням та високою доступністю [8].

Серед доступних у RDS механізмів баз даних (MySQL, PostgreSQL, MariaDB, Oracle та SQL Server), PostgreSQL було обрано для цього проєкту завдяки його стабільності, розширеним можливостям обробки даних JSON, підтримці транзакцій, гнучким механізмам індексації та високій сумісності з бібліотеками ORM, що використовуються в NestJS (включаючи TypeORM та Prisma) [9].

Використання RDS дозволяє розробникам позбутися рутинних завдань з обслуговування інфраструктури бази даних. Наприклад, автоматичне щоденне резервне копіювання забезпечує швидке відновлення системи у разі втрати даних, а можливості моніторингу через Amazon CloudWatch дозволяють відстежувати навантаження, продуктивність та поведінку запитів у режимі реального часу.

Завдяки масштабованості, база даних легко адаптується до зростання аудиторії та обсягу інформації. В рамках архітектури системи серверна частина, реалізована на NestJS, підключена до AWS RDS, яка обробляє клієнтські запити та взаємодіє з базою даних через ORM. Це забезпечує високий рівень абстракції, зменшує ризик помилок під час використання SQL та пришвидшує розробку.

Особлива увага приділяється безпеці: підключення до бази даних захищене SSL/TLS, а доступ до RDS обмежений на рівні мережі через групи безпеки AWS VPC, що обмежує діапазони IP-адрес, з яких можливе підключення. Ролі IAM також використовуються для захисту доступу до метаданих бази даних під час масштабування або інтеграції з іншими сервісами.

Крім того, AWS RDS пропонує безперешкодну інтеграцію з іншими сервісами AWS, включаючи S3 для імпорту/експорту даних, CloudWatch для ведення журналу та AWS Secrets Manager для безпечного зберігання облікових даних бази даних.

Таким чином, AWS RDS забезпечує надійну, масштабовану та безпечну систему зберігання структурованих даних у рамках загальної архітектури веб-платформи. Обраний підхід поєднує продуктивність, надійність та простоту обслуговування, що є важливим елементом комерційного розгортання сучасного веб-застосунку.

2.4 Мова програмування TypeScript

Під час розробки веб-платформи основною мовою програмування як для фронтенд-, так і для бекенд-частин було обрано TypeScript. Це надмножина JavaScript, яка додає статичну типізацію та підтримку сучасних функцій об'єктно-орієнтованого програмування. Такий вибір був мотивований бажанням підвищити надійність коду, покращити масштабованість системи та зменшити кількість помилок під час розробки та підтримки [2].

TypeScript поєднує гнучкість динамічного середовища JavaScript з перевагами компіляції та перевірки типів, що дозволяє виявляти значну частину помилок під час компіляції, а не під час виконання. Це особливо важливо в контексті побудови складної розподіленої системи, що включає велику кількість клієнтської та серверної логіки. Типізація допомагає чітко визначити структуру даних, запитів API, відповідей та внутрішніх сервісів, мінімізуючи ризик неправильного використання змінних або функцій.

На фронтенді TypeScript використовується разом з React, що дозволяє створювати типізовані компоненти, типізовані пропи, гачки та стани. Це покращує передбачуваність поведінки інтерфейсу та дозволяє інтеграцію інструментів автоматичної генерації типів на основі контрактів API.

На серверній частині платформа використовує NestJS, сучасний фреймворк на основі TypeScript, який повною мірою використовує його можливості, включаючи декоратори, інтерфейси, дженерики, модулі та інверсію залежностей. Це дозволяє організувати код як модульну структуру з чітким розподілом обов'язків та зрозумілими контрактами між сервісами. Окрім основних переваг статичної типізації, TypeScript значно спрощує інтеграцію з зовнішніми бібліотеками та API, завдяки наявності декларативних описів типів (типів з @types).

Крім того, TypeScript позитивно впливає на довговічність коду. У співпраці типи слугують документацією, яка дозволяє швидше ознайомитися з кодом один одного, перевірити правильність використання функцій та значно зменшити витрати на інтеграцію для нових розробників.

Таким чином, використання TypeScript не лише покращило якість коду та пришвидшило розробку, але й заклало основу для безпечної та масштабованої архітектури системи. Це покращує стабільність, передбачуваність поведінки та спрощує інтеграцію між усіма компонентами платформи, як на рівні клієнта, так і сервера.

2.5 Фреймворк Nest.js

Фреймворк Nest.js було обрано основою для серверної реалізації веб-платформи. Це сучасне рішення на Node.js, що поєднує переваги об'єктно-орієнтованого, функціонального та реактивного підходів програмування. Nest.js базується на Express (або, за бажанням, Fastify) та повністю використовує можливості TypeScript, ідеально задовольняючи вимоги розробки масштабованого, модульного та типологічно безпечного бекенду [5].

Однією з головних сильних сторін Nest.js є його архітектура, натхненна Angular, яка забезпечує чіткий розподіл логіки між модулями, контролерами, сервісами та провайдерами. Це допомагає ефективно організувати кодову базу, спрощуючи обслуговування, тестування та масштабування застосунків. Завдяки інверсії залежностей, реалізованій вбудованим контейнером для впровадження залежностей, компоненти системи можна легко замінити, ізолювати або використовувати повторно.

Nest.js також пропонує повну підтримку декораторів, що дозволяє визначати маршрути, валідацію, авторизацію, впровадження залежностей та інші аспекти поведінки за допомогою чітких декларативних конструкцій. Наприклад, створення REST API з описом методів, параметрів, кодів стану та обробки винятків реалізовано у зрозумілій формі, яка добре масштабується разом із зростанням проекту.

Крім того, Nest.js підтримує GraphQL, WebSocket, мікросервіси та черги повідомлень (наприклад, RabbitMQ або Kafka), а також інтегрується зі Swagger для автоматичної генерації документації API. Для цієї платформи було використано Swagger для створення документації REST API, що значно спростило внутрішню розробку та зовнішню інтеграцію.

Ще однією важливою перевагою є активна спільнота Nest.js та велика кількість офіційних та сторонніх модулів. Це допомагає скоротити час розробки завдяки перевіреним рішенням для перевірки, ведення журналу, обробки помилок, кешування, надсилання повідомлень тощо.

Завдяки Nest.js, серверна частина веб-платформи не тільки стабільна та продуктивна, але й масштабована. Він легко масштабується як вертикально (шляхом розширення окремих модулів), так і горизонтально (шляхом розгортання в кластерному середовищі AWS ECS), що особливо важливо в контексті хмарної інфраструктури. Таким чином, Nest.js став основою для надійної бекенд-архітектури, яка ефективно підтримує всі функції системи.

2.6 Об'єктно-реляційна проекція Sequelize

Під час розробки бекенду веб-платформи використання бази даних відіграє життєво важливу роль. Для забезпечення ефективної, безпечної та типологічно безпечної взаємодії з реляційною базою даних Amazon RDS (з використанням PostgreSQL) було обрано бібліотеку ORM Sequelize. Цей потужний інструмент об'єктно-реляційного відображення дозволяє маніпулювати таблицями бази даних за допомогою об'єктів та класів у коді, значно спрощуючи написання SQL-запитів [6].

Sequelize підтримує більшість популярних реляційних СУБД, включаючи PostgreSQL, MySQL, MariaDB, SQLite та MSSQL. Однак цей проєкт використовує PostgreSQL, який завдяки своїй гнучкості, надійності та масштабованості ідеально підходить для складних вебзастосунків з великою кількістю взаємопов'язаних сутностей.

Завдяки Sequelize стало можливим створити чітко структуровану систему моделей, які легко описують зв'язки між сутностями: користувачами, списками відтворення, аудіофайлами, лайками, коментарями, історією прослуховування тощо. Це спрощує реалізацію складних запитів (наприклад, отримання списку відтворення з вбудованими треками та відповідними метаданими), уникаючи необхідності вручну вводити довгі SQL-запитувачі.

Крім того, Sequelize включає механізми міграції для створення, відстеження та розгортання змін схеми бази даних контрольованим чином. Таким чином, оновлення структури таблиці можна виконувати поетапно, без втрати даних та з повним контролем над змінами, що особливо важливо у виробничому середовищі.

У поєднанні з Nest.js, Sequelize дозволив реалізувати надійний, безпечний та масштабований рівень доступу до даних. Завдяки підходу ORM розробники можуть зосередитися на бізнес-логіці застосунку, не витрачаючи час на написання необробленого SQL-коду, зберігаючи при цьому повний контроль над структурою бази даних та зв'язками. Це значно підвищило продуктивність розробки, зменшило кількість потенційних помилок та спростило підтримку системи в майбутньому.

2.7 Бібліотека React.js

Для розробки клієнтської частини веб-платформи було використано бібліотеку React.js, одну з найпопулярніших та найпоширеніших технологій у сучасній фронтенд-розробці. Створений Facebook (тепер Meta), він вирізняється компонентно-орієнтованим підходом, що дозволяє створювати складні інтерфейси шляхом поділу їх на логічно ізольовані та повторно використовувані компоненти [4].

Одна з головних переваг React – це його декларативний підхід до розробки інтерфейсу користувача, який описує не те, як змінювати DOM, а те, яким він має бути у відповідь на заданий стан. Це значно спрощує логіку інтерфейсу, робить код більш передбачуваним та легшим для налагодження. React використовує віртуальний DOM, який дозволяє ефективно оновлювати інтерфейс, змінюючи лише ті частини, які потребують повторного рендерингу.

На цій платформі React став основою для розробки односторінкових застосунків (SPA), які пропонують високий рівень динамічності та інтерактивності. Його переваги особливо помітні в таких функціях, як динамічне завантаження контенту (треків,

альбомів, плейлистів), оновлення даних без перезавантаження сторінок, фільтрація, сортування, керування бібліотеками користувачів та взаємодія зі сторонніми API.

Крім того, для керування станом у застосунку React інтегровано інструментарій Redux. Це офіційно рекомендований спосіб організації глобальних станів у React-проектах. Це дозволяє ефективно працювати з великими обсягами даних та централізовано зберігати стани користувачів, налаштування, прогрес відтворення музики тощо. Для взаємодії з асинхронними даними з сервера використовується React Query, який забезпечує кешування, інвалідацію, попереднє завантаження та інші аспекти використання API.

Усі маршрути застосунку реалізовані за допомогою React Router, що дозволяє навігацію без повного перезавантаження сторінки, що є важливим елементом для високоякісного користувацького досвіду.

Також важливо зазначити, що як інструмент збірки використовувався Vite, сучасна альтернатива Webpack, що забезпечує надзвичайно швидкий запуск та гаряче оновлення модулів під час розробки [13].

Використання React дозволило нам створити гнучку, масштабовану та продуктивну клієнтську частину, яку було легко підтримувати, тестувати та розширювати. Компонентний підхід спростив повторне використання елементів інтерфейсу користувача в різних частинах системи, а величезна екосистема React забезпечила доступ до перевірених бібліотек та інструментів. Все це зробило React ідеальним вибором для створення веб-інтерфейсу для музичної платформи.

2.8 Бібліотека MaterialUI

Одним із ключових аспектів розробки будь-якого сучасного веб-застосунку є створення зручного, адаптивного та захопливого інтерфейсу користувача. Для реалізації фронтенду музичної платформи ми обрали Material UI, одну з

найпопулярніших бібліотек інтерфейсу користувача для React, засновану на філософії Material Design від Google [11].

Material UI пропонує велику кількість готових до використання компонентів інтерфейсу: кнопки, форми, діалоги, вкладки, таблиці, індикатори завантаження, системи навігації, сповіщення (панелі сповіщень) тощо. Це значно прискорило процес розробки та забезпечило єдиний дизайн застосунку, що позитивно вплинуло на взаємодію з користувачем. Завдяки MUI інтерфейс системи є сучасним, узгодженим та відповідає принципам доступності.

Бібліотека розроблена з використанням компонентного підходу React, що дозволяє легко створювати власні інтерфейси. Система тем дозволяє гнучко налаштовувати зовнішній вигляд застосунку: змінювати колірну палітру, типографіку, розміри елементів та анімацію. У проекті реалізовано єдину тему з узгодженими кольорами, полями та стилями по всьому застосунку.

Веб-платформа використовувала кілька компонентів MUI для створення меню навігації, карток відображення треків, модальних вікон авторизації та реєстрації, списків відтворення, елементів керування плеєром, повідомлень про помилки, а також форм введення та пошуку.

Використовуючи Material UI, розробники змогли створити інтуїтивно зрозумілий та привабливий інтерфейс, який відповідає сучасним стандартам веб-дизайну, забезпечуючи послідовну взаємодію користувача з платформою та сприяючи загальній якості програмного забезпечення.

2.9 Мовний штучний інтелект AssemblyAI

В рамках розробки музичної платформи необхідно було використовувати зовнішній інтелектуальний сервіс для перетворення аудіофайлів у текст. Це особливо важливо для таких функцій, як автоматична генерація текстів пісень, вилучення

метаданих з вокальних треків, створення субтитрів, покращення доступності контенту та майбутнє розгортання інструментів для аналізу емоційного забарвлення голосу або теми. Для реалізації цієї функціональності було обрано сервіс AssemblyAI, провідного постачальника API для транскрипції, аналізу та обробки голосових даних на основі штучного інтелекту [10].

AssemblyAI пропонує розширену якість розпізнавання мовлення завдяки використанню сучасних мовних моделей, навчанню на великомасштабних наборах даних та регулярному оновленню своїх моделей на основі останніх досліджень у галузі NLP та ASR (автоматичного розпізнавання мовлення). Однією з головних переваг сервісу є легкість інтеграції через REST API, що дозволило швидко підключити функцію транскрипції без необхідності розгортання складних систем машинного навчання. AssemblyAI також пропонує розширені функції, такі як ведення щоденника мовця, автоматичне виявлення ненормативної лексики, виявлення ключових слів, аналіз настроїв, класифікація тем, автоматичне форматування тексту та пунктуація.

Під час аналізу альтернатив розглядалися інші популярні сервіси, такі як Google Cloud Speech-to-Text, Amazon Transcribe та Deepgram. Однак Google та AWS вимагають складнішого налаштування та вищого порогу входу для швидкого впровадження. Deepgram, з іншого боку, поступається AssemblyAI за точністю транскрипції згідно з незалежними бенчмарками. AssemblyAI також пропонує зручну систему керування ключами, ведення журналу запитів та адаптацію до спеціалізованих доменів.

В рамках архітектури проекту AssemblyAI працює як зовнішній мікросервіс, на який надсилаються аудіофайли, що зберігаються на AWS S3. Серверна частина платформи, реалізована на Nest.js, ініціює процес транскрипції, обробляє відповіді сервісу та зберігає текстові результати для подальшої обробки (відображення в інтерфейсі користувача, пошук за контентом, синхронізація з відтворенням тощо).

Завдяки AssemblyAI платформа має надійний, масштабований та точний інструмент обробки голосу, що відкриває нові можливості для розробки функцій, включаючи голосове керування, автоматичний вибір тегів, аналіз тексту на токсичність та тематичну орієнтацію. Таким чином, AssemblyAI значно підвищує інтелектуальну цінність системи та є важливим елементом її технологічного стеку.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ВЕБ-ПЛАТФОРМ ДЛЯ ПРОСЛУХОВУВАННЯ ТА РОЗПОВСЮДЖЕННЯ МУЗИЧНИХ ФАЙЛІВ

У цьому розділі представлено вичерпний огляд архітектурних рішень, логіки побудови програмної структури та практичної реалізації веб-системи управління музичним контентом. Розроблена платформа є сучасним рішенням для відтворення, зберігання, обробки та організації аудіофайлів, надаючи пріоритет гнучкості, масштабованості та розширюваності. Проєкт базується на клієнт-серверній архітектурі з чітким розподілом обов'язків між фронтендом, бекендом та зовнішніми сервісами.

Однією з ключових особливостей платформи є автоматична транскрипція аудіофайлів у текст. Для цього було обрано AssemblyAI, лідера ринку автоматичного розпізнавання мовлення. Його конкурентні переваги включають високу точність, підтримку сучасних моделей машинного навчання, легку інтеграцію через REST API та можливість розпізнавання великих файлів. Порівняно з аналогами, такими як Google Speech-to-Text або Whisper від OpenAI, AssemblyAI пропонує кращу якість для класичної музики або домашніх записів без необхідності складного налаштування. Важливо зазначити, що платформа не обмежується лише базовим функціоналом транскрипції. У перспективі передбачено реалізацію можливостей аналізу змісту аудіо

Усі архітектурні рішення та функціональні модулі платформи реалізовані з урахуванням принципів модульності, безпеки типів, асинхронності, компонентного підходу та масштабованості. Це дозволяє ефективно розробляти систему, інтегрувати нові сервіси (системи рекомендацій, аналітика прослуховування, соціальні функції тощо) та забезпечує комфорт користувача у щоденній роботі з музичним контентом.

3.1 Архітектура та структура веб-платформи

Розроблена музична веб-платформа реалізована за принципами класичної багаторівневої клієнт-серверної архітектури. Система чітко розрізняє три основні рівні: клієнтський інтерфейс, серверну логіку та зовнішні сервіси, що забезпечують зберігання, обробку та аналіз даних. Такий підхід дозволяє легко масштабувати кожен компонент окремо, забезпечуючи гнучкість, надійність та можливість розширення функціональності.

3.1.1 Розробка серверної частини

Основними вимогами є забезпечення гнучкості, масштабованості та підтримка системи у майбутньому. Платформа повинна обробляти завантаження та відтворення аудіофайлів, зберігати користувацьку активність, реалізовувати механізм збереження, збірок та музики, авторизації, генерації текстів пісень за допомогою штучного інтелекту, а також забезпечувати роботу з базою даних і хмарними сервісами. Усі ці задачі вимагали сучасного та структурованого підходу до розробки.

Саме тому, як було вказано у розділі вище за основу для розробки backend частини було обрано фреймворк Nest.js. Враховуючи вбудовану підтримку модульності у Nest.js, для досягнення гнучкості серверної частини та дотримання принципів чистої архітектури було додатково застосовано підхід гексагональної архітектури. Такий підхід дозволив чітко розділити логіку застосунку від зовнішніх можливих залежностей, основної бізнес логіки, взаємодії з базою даних та сторонніми сервісами. Це допомагає легкій заміні модулів без впливу на основну логіку продукту логіку.

Приклад створеної архітектури зображений на рисунку 3.1 у середовищі розробки з ініціалізованими сутностями:

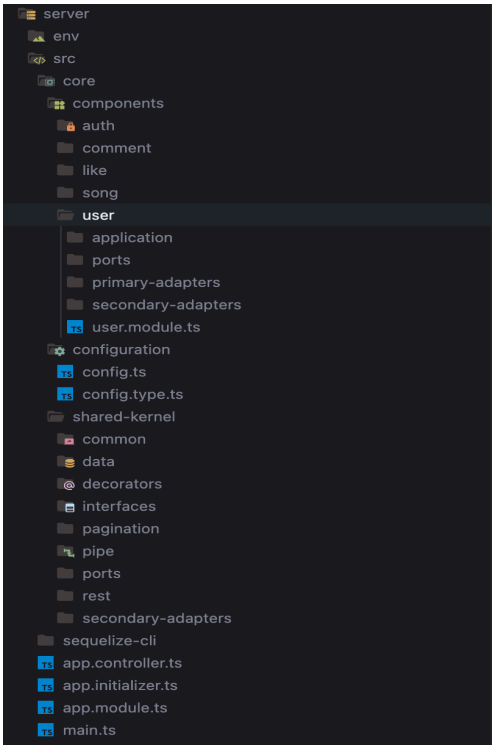

Рисунок 3.1 – Архітектура серверної частини

Звертаючи увагу на каталог components ми можемо пробачити приклад модульності Nest.js, бо саме так виглядає розбиття продукту на сутності для зручності розробки та можливості підходу ін’єкції залежностей.

Також на основі модуля user дуже гарно видно розбиття кожного окремого модуля на певні шари, для імплементації гексогональної архітектури.

Каталог application – відповідальний за опис основної бізнес-логіки у компоненті та опис вищих шарів взаємодії.

Каталог ports – зберігає у собі усі потрібні інтерфейси для імплементації класів необхідних для певного модуля.

Каталог primary-adapters – слугує для збігання усіх контроллерів, які описуються кінцеві точки, що потрібно для інтеграції на користувацькій частині.

Каталог secondary-adapters – відповідальний за шар взаємодії серверної частини з створеною раніше базою даних, саме у ньому описуються моделі та репозиторії, які допомагають налагодити взаємодію backend з базою даних.

Наступним етапом після опису архітектури серверної частини є опис моделей та репозиторіїв, які забезпечать правильну та зручну взаємодію з базою даних. Для цього було обрано використовувати Sequelize ORM.

Звертаючись до раніше створеної діаграми бази даних опишемо першу модель користувача, рисунок 3.2.

```
@Table({ tableName: 'user' })
export default class UserModel extends Model<UserModel> {
  @PrimaryKey
  @AutoIncrement
  @Column
  id: number;

  @Column
  username: string;

  @AllowNull
  @Column
  description: string;

  @AllowNull
  @Column
  avatar: string;

  @Default(UserType.user)
  @Column(DataType.ENUM(...Object.values(UserType)))
  type: UserType;

  @Default(false)
  @Column
  is_verified: boolean;

  @Column
  created_at: Date;

  @AllowNull
  @Column
  deleted_at: Date | null;

  @HasMany(() => SongModel)
  songs: SongModel[];

  @BelongsToMany(() => SongModel, () => LikeToSongModel)
  liked_songs: SongModel[];

  @HasMany(() => PlaylistModel)
  playlists: PlaylistModel[];

  @BelongsToMany(() => PlaylistModel, () => LikeToPlaylistModel)
  liked_playlists: PlaylistModel[];
}
```

Рисунок 3.2 – Модель користувача

Кожна модель описується завдяки декораторам, які надає сам Sequelize і кожен з яких описує певну властивість полів у тій чи іншій таблиці. Є декоратори які

відповідають за налаштування полів у певній таблиці, для прикладу - @Column, а є такі я слугують для опису зв'язків між моделями – @HasMany.

Для коректного застосування ін'єкції залежностей або Dependency Injection, для кожної моделі потрібно розробити свій репозиторій. Це запезпечить можливість змінювати моделі для взаємодії шляхом мінімальної зміни коду у файлах бізнес-логіки. Для кожного репозиторія описується свій особистий інтерфейс, де повинні знаходитись необхідні CRUD функції. Основний файл репозиторія – адаптер у імплементованому вигляді зображений на рисунку 3.3.

```
export class UserRepositoryAdapter implements UserRepository {
  constructor(
    @InjectModel(UserModel) private readonly userRepository: typeof UserModel,
  ) {}

  public async getOneWithLikedSongs(userId: number): Promise<User> {
    const model = await this.userRepository.findOne({
      where: { id: userId, deleted_at: null },
      include: [
        {
          association: 'liked_songs',
          where: { deleted_at: null },
          through: { attributes: [] },
          include: [
            {
              association: 'user',
              required: true,
              where: { deleted_at: null },
            },
          ],
        },
      ],
    });
    return model?.toJSON() || null;
  }

  public async getOneWithLikedPlaylists(userId: number): Promise<User> {
    const model = await this.userRepository.findOne({
      where: { id: userId, deleted_at: null },
      include: [
        {
          association: 'liked_playlists',
          where: { deleted_at: null },
          through: { attributes: [] },
```

Рисунок 3.3 – Частина файлу адаптера репозиторія

Наступним кроком є створення бізнес-логіки платформи. Майже жоден продукт не може існувати без функціоналу з автентифікацією користувача, веб платформа для прослуховування та розповсюдженням музичних файлів не є винятком. Користувачі повинні мати змогу переглядати списки, профілі інших користувачів, прослуховувати

музичні файли, завантажувати власні виключно після реєстрації на платформі. Для створення функціоналу авторизації було обрано підхід з використанням JWT токенів.

JWT токен – це безпечний спосіб передачі інформації між клієнтом і сервером у вигляді JSON-об'єкта, найчастіше використовується у функціоналі авторизації. Основною ідею є генерація та присвоєння токена певному користувачеві при реєстрації та подальшою його перевіркою при використанні платформи. Токен має можливість закінчитись і тоді платформа має знову попросити користувача авторизуватись.

Бізнес-логіка розділяється на багато файлів із підназвою UseCase, щоб відповідати паттерну single responsibility. Метою цього паттерна є розбиття великих класів з безліччю різноманітної функціональності на більш менші, зручніші для використання, щоб попередити нагромадження інформації та зробити більш зрозумілим код та проект для потенційних колег-розробників.

Як приклад оберемо функцію реєстрації користувача. У цьому класі реалізована функція для створення сутності користувача на платформі разом із усіма необхідними додатковими сутностями, як user_auth. Для створення записів у базі даних потрібно ініціалізувати репозиторії за допомогою конструктора, що можна побачити на прикладі на рисунку 3.4. Наступним кроком є прийняття аргументів переданих у цю функцію, а саме email, username, password, description та avatar. Деякі з цих аргументів зобов’язані, такі як пароль чи електронна поштова скринька повинні пройти шифрування чи форматування перед додаванням до бази даних, щоб це було безпечно для акаунтів користувачів. Також якщо користувач має бажання додати фотографію під час реєстрації свого акаунту, наша функція повинна обробити отриманий файл та передати його у хмарне сховище S3, а у базу даних зберегти посилання на отримання цього зображення. Вся взаємодія (створення, редагування чи видалення записів) з таблицями бази даних відбувається виключно за використання транзакцій, бо вони забезпечують безпечне збереження або видалення даних з бази.

```
@Injectable()
export default class RegisterUseCase
  implements UseCase<RegisterArguments, RegisterResponse>
{
  constructor(
    @Inject(TokenServiceInterfaceType)
    private readonly tokenService: TokenServiceInterface,
    private readonly queryBus: QueryBus,
    private readonly commandBus: CommandBus,
    private readonly getEncryptedPasswordUseCase: GetEncryptedPasswordUseCase,
    private readonly sequelize: Sequelize,
    @Inject(S3ServiceInterfaceType)
    private readonly s3Service: S3ServiceInterface,
  ) {}

  public async execute({
    email,
    username,
    password,
    description,
    avatar,
  }: RegisterArguments): Promise<RegisterResponse> {
    const lowercaseEmail = email.toLowerCase();

    const [foundUserByEmail, foundUserByUsername] = await Promise.all([
      this.queryBus.execute(new GetUserAuthByEmailQuery(lowercaseEmail)),
      this.queryBus.execute(new GetUserByUsernameQuery(username)),
    ]);

    if (foundUserByEmail) {
      throw new BadRequestException('User with this email is already exists');
    }

    if (foundUserByUsername) {
      throw new BadRequestException(
        'User with this username is already exists',
      );
    }

    const encryptedPassword =
      await this.getEncryptedPasswordUseCase.execute(password);
```

Рисунок 3.4 – Зразок файлу, що відповідає за логіку реєстрації користувача

Одною з найбільших проблематикою веб-платформи є збереження та прослуховування музичних файлів та обкладинок пісень і збірок. Дуже важливо щоб користувач завжди мав доступ до прослуховування музики, міг додавати власні пісні, видаляти власні існуючі файли. Саме для такої взаємодії було обрано хмарне сховище Amazon S3. Воно дозволяє за допомогою коду завантажувати файли до хмарного сховища.

Для прийому файлів у цьому продукті реалізований сервіс для взяємодії з S3. Він надає можливість обробити файл, присвоїти йому певне ім’я у хмарі та загрузити його у сховище, а у результаті backend отримує посилання для взаємодії із цим файлом, яке вже зберігається у базі даних. Сервер може відображати ці зображення чи вмикати аудіофайли роблячи GET запити на ці згенеровані посилання.

На рисунку 3.5 зображена функція для завантаження файлів у сховище S3.

```
public async uploadFile(
  file: Express.Multer.File,
  key: string,
  group: string[],
  isPublicRead: boolean,
): Promise<string> {
  const groupKey = this.createFileKey(key, group);
  const bucketName = this.configService.get<string>('aws.filesBucketName');

  const command = new PutObjectCommand({
    Bucket: bucketName,
    Body: Buffer.from(file.buffer),
    Key: groupKey,
    ACL: isPublicRead ? 'public-read' : 'private',
    ContentType: file.mimetype,
    ContentLength: file.size,
  });

  await this.s3.send(command);

  const fileUrl = this.getFileUrl(groupKey);

  return fileUrl;
}
```

Рисунок 3.5 – Функція, що відповідає за завантаження файлів у хмарне сховище Amazon S3

Кінцеві точки для підключення клієнтської частини описуються у контроллерах, за допомогою певних декораторів, які пропонує Nest.js: @GET, @POST, @PUT, @DELETE – кожен з цим декораторів описує певний вид запитів, які надсилаються серверній частині на обробку. Також за допомогою декораторів @Body чи @Param є влаштована можливість приймати аргументи як у body запиту, так і у параметрах пошукового рядку. Для імплементації документації backend частини було обрано Swagger, який теж описується за допомогою декораторів у найвищій частині – контролері. Переглянути створення кінцевої точки реєстрації у контролері можливо на рисунку 3.6

```
@Controller()
@ApiTags('auth')
export class AuthController {
  constructor(
    private readonly loginUseCase: LoginUseCase,
    private readonly logoutUseCase: LogoutUseCase,
    private readonly registerUseCase: RegisterUseCase,
  ) {}

  @Post('/register')
  @ApiResponseDoc(RegisterResponse)
  @UseInterceptors(FileInterceptor('avatar'))
  @ApiConsumes('multipart/form-data', 'application/json')
  async registerUser(
    @Body() registerDto: RegisterRequest,
    @UploadedFile(new TransformFilePipe({ isRequired: false }))
    avatar: Express.Multer.File,
  ) {
    return this.registerUseCase.execute({
      username: registerDto.username,
      email: registerDto.email,
      password: registerDto.password,
      description: registerDto.description,
      avatar,
    });
  }
```

Рисунок 3.6 – Імплементація кінцевої точки для реєстрації користувача

Підключення штучного інтелекту відбулось в точності однаково, як підключення S3 сховища. Було створено окремий сервіс для взаємодії з наданим API. Замість запиту на збереження файлів, як це було зроблено у прикладі з S3, відбувався запит з аудіофайлом на певний ресурс наданий AssemblyAI з секретними ключами для верифікації, тобто Access Keys, у наслідок якого після обробки аудіо було отримано текст і збережено у базі даних наданого музичного файлу, розробку цього функціоналу продемонстровано в Додаток А. Отриманий текст було отримано і одразу передано до функції для створення запису у таблиці song, щоб потім під час розробки можна було зручно звертатись за цією змінною.

У серверній частині реалізований весь функціонал наведений у діаграмі прецедентів, повністю задокументовані всі кінцеві точки за допомогою бібліотеки Swagger та готові до підключення клієнтської частини у вигляді кінцевих точок, або API.

3.1.2 Розробка клієнтської частини

Розробка клієнтської частини розпочалась одразу після завершення серверної. Саме у цьому пункті приступимо до розробки архітектури клієнтської частини, опису функціональних компонентів для використанні у розмітці на різних сторінках платформи та підключення серверної частини.

React.js було обрано основною бібліотекою для клієнтської частини через його популярність, продуктивність та активній використання у спільноті. Крім того, для пришвидшення розробки та адаптивності користувацького інтерфейсу було використано бібліотеку Material UI.

Загальна структура фронтенд частини зображена на рисунку 3.7.

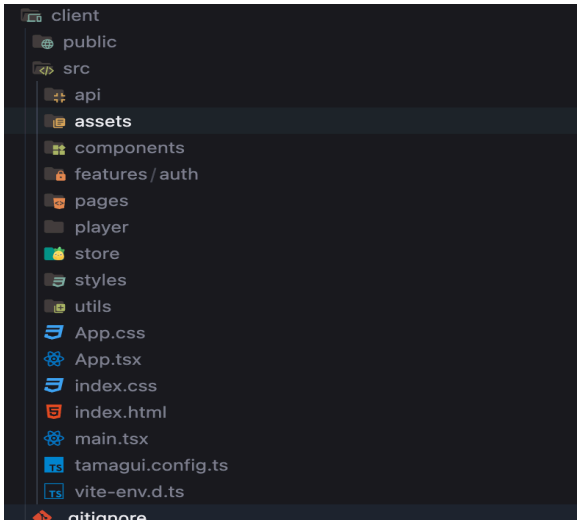

Рисунок 3.7 – Структура клієнтської частини

Каталог pages відповідає за окремі сторінки платформи, такі як: Головна, збережені, профіль, перегляд треку або плейлисту, сторінка авторизації чи реєстрації.

Каталог components – це бібліотека повторно використовуваних елементів інтерфейсу (кнопки, форми, карточки музики);

Каталог api слугує для файлів які відповідають за взаємодію з серверною частину, тобто роблять запити на отримання даних, створення, видалення і тд.

Каталог store – відповідає за ініціалізацію сховища даних, які мають зберігати на фронтенді під час сесії користувача.

Однією з ключових вимог до клієнтської частини була масштабованість та можливість швидкого розширення функціональності. Щоб досягти цього було використано розділення інтерфейсу на компоненти, що допомагає керувати стилями, відображати логіку та повторно використовувати частини інтерфейсу в усьому застосунку.

Веб-платформа повинна мати головну сторінку, сторінки збережених, профілі, детальні сторінки музичного файлу та збірки, реєстрації, авторизації, тому потрібно описати їх імплементація у каталозі pages.

Розглянемо використання структуру навігації там варіанти використання наших сторінок у головному файлі App.tsx на рисунку 3.8.

```
function App() {
  return (
    <ThemeProvider theme={theme}>
      <CssBaseline />
      <Router>
        <PlayerProvider>
          <Box
            sx={{
              display: "flex",
              flexDirection: "column",
              minHeight: "100vh",
              width: "100%",
            }}
          >
            <Header />
            <Box
              component="main"
              sx={{
                flexGrow: 1,
                py: 3,
                width: "100%",
                display: "flex",
                flexDirection: "column",
                alignItems: "center",
              }}
            >
              <Box sx={{ width: "100%", maxWidth: "1200px", px: 2 }}>
                <Routes>
                  <Route path="/register" element={<Register />} />
                  <Route path="/login" element={<Login />} />
                  <Route
                    path="/feed"
                    element={
                      <PrivateRoute>
                        <Feed />
                      </PrivateRoute>
                    }
                  />
                  <Route
                    path="/profile/:id"
                    element={
                      <PrivateRoute>
                        <Profile />
                      </PrivateRoute>
                    }
                  />
                  <Route
```

Рисунок 3.8 – Приклад використання сторінок та налаштування роутингу

Наступним ключовим етапом клієнтської частини є завантаження музичних файлів на платформу. Для цього було створено окрему сторінку із використанням React Hook Forms, що допомагає дуже швидко та легко реалізовувати форми на клієнтській частині та надавати можливість використовувати її для взаємодії з чимось.

У нашому випадку ця можливість використовується для взаємодії із серверною частиною, щоб надіслати запит зі всіма отриманими даними від користувача. Цей функціонал відображений на рисунку 3.9:

```
const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault();
  if (!validateForm()) return;
  setIsLoading(true);
  try {
    const data = new FormData();
    data.append("name", formData.name);
    if (formData.cover) data.append("cover", formData.cover);
    if (formData.audio) data.append("audio", formData.audio);
    await songApi.createSong(data);
    navigate("/feed");
  } catch (error) {
    setErrors({ name: "Failed to create song. Please try again." });
  } finally {
    setIsLoading(false);
  }
};
```

Рисунок 3.9 – Функція для надсилання запиту на збереження інформації про музичний файл

Для взаємодії з backend частиною платформи було розроблено прошарок з функціями, у яких групується уся надана інформація та за допомогою axios. Це дозволяє підтримувати чисту архітектуру клієнтської частини та легко змінювати маршрути запитів або обробку помилок без зміни основного коду компонентів

Реалізація цього прошарку продемонстрована на рисунку 3.10.

```
const authApi = {
  register: async (data: FormData): Promise<AuthResponse> => {
    const response = await axios.post(`${API_URL}/register`, data, {
      headers: {
        "Content-Type": "multipart/form-data",
      },
    });
    return response.data.data;
  },

  login: async (data: LoginRequest): Promise<AuthResponse> => {
    const response = await axios.post(`${API_URL}/login`, data);
    return response.data.data;
  },
```

Рисунок 3.10 – Прошарок для взаємодії з серверною частиною

Останньою і найважливішою частиною розробки клієнтської частини є прийняття та виведення даних отриманих із серверу. Використовуючи запити описані заздалегідь у нашому прошарку ми будемо отримувати дані для нашої платформи. На клієнті наявні також прописані прошарки для отримання музичних файлів та музичних збірок. Дані приходять у вигляді масивів, клієнт повинен рендерити їх, щоб користувачам було зручно взаємодіяти з платформою (рисунок 3.11).

```
  return (
    <Container maxWidth="lg">
      <Box sx={{ mt: 4 }}>
        <Typography variant="h4" component="h1" gutterBottom>
          Liked Songs
        </Typography>
        <Box
          sx={{
            display: "grid",
            gridTemplateColumns: {
              xs: "1fr",
              sm: "1fr 1fr",
              md: "1fr 1fr 1fr",
            },
            gap: 3,
          }}
        >
          {songs.map((song, idx) => (
            <SongCard key={song.id} song={song} songs={songs} index={idx} />
          ))}
        </Box>
      </Box>
    </Container>
  );
```

Рисунок 3.11 – Частина коду для форматування отриманих результатів

В результаті, під час розробки клієнтської частини було створено сучасний та ефективний інтерфейс, який виконує усі функції, наявні у серверній частині, відповідає принципам масштабованості та адаптивності.

3.2 Функціональна декомпозиція веб-платформи

Розроблена веб-платформа – це сучасний інструмент для комплексного управління музичним контентом, що охоплює як прослуховування, так і керування аудіофайлами, включаючи їх транскрипцію. Основні функції системи спрямовані на кінцевих користувачів (слухачів), виконавців або власників контенту та інтеграцію із зовнішніми сервісами для обробки та зберігання даних.

Система дозволяє користувачам легко переглядати та прослуховувати доступні аудіодоріжки. Це досягається завдяки інтеграції швидкого завантаження сторінок, медіаплеєру з підтримкою основних функцій керування відтворенням (відтворення, пауза, перемотування), а також можливості додавання треків до обраного або до власного списку відтворення. Усі дії користувача записуються для створення історії взаємодій, яку можна використовувати пізніше для персоналізації контенту.

Авторизовані користувачі, незалежно від того, чи це слухачі, чи власники, отримують переваги від комплексних функцій управління контентом. До них належать завантаження нових аудіофайлів у форматах, сумісних з AssemblyAI (наприклад, .mp3), додавання метаданих (назва, жанр, автор) та обкладинки, а також автоматичне отримання текстових транскрипцій аудіофайлів. Ця транскрипція генерується за допомогою голосового штучного інтелекту AssemblyAI та може використовуватися як текст пісні або як додаткова інформація для досліджень та аналізу.

Платформа пропонує повну автентифікацію та авторизацію користувачів. Залежно від їхньої ролі, користувачі мають доступ до різних рівнів функціональності:

звичайні користувачі можуть переглядати та слухати контент, виконавці можуть завантажувати нові треки, а адміністратори можуть модерувати, редагувати або видаляти контент. Вся система розроблена з урахуванням прав доступу та можливостей, визначених для кожного типу користувача.

Функціональність платформи охоплює весь робочий процес аудіо: від завантаження та зберігання до відтворення та автоматичної обробки. Це створює потужну, практичну та масштабовану систему, яка може служити основою як для комерційних послуг, так і для освітніх чи дослідницьких цілей.

Для того щоб створити надійну та оптимізовану базу даних нам потрібно остаточно вирішитись з основним функціоналом нашої платформи, описати які сутності має налічувати наша платформа, які вони мають мати можливості та рівні доступу на взаємодію з даними чи іншими користувачами. Саме для вирішення цієї проблеми ми будемо використовувати діаграму прецедентів.

Діаграма прецедентів – являється одним з основних елементів моделювання та розробки основної архітектури продукту, який показує, як користувачі взаємодіють із системою за допомогою набору деяких сценаріїв. Він допомагає визначити основні функції системи, також може допомогти визначити поведінку у певних ситуаціях.

Кожен користувач платформи повинен мати змогу зареєструватись на платформі чи увійти у свій особистий обліковий запис для подальшої взаємодії з платформою. Перегляд загальних списків завантажених музичних файлів, створених музичних збірок, додавання музичних файлів та музичних збірок до списків збереженого, прослуховування аудіо, додавання нових пісень, створення нових музичних підбірок, видалення музичної збірки чи пісні, перегляд профілів інших користувачів мають бути доступні для кожної людини яка взаємодіє з платформою. Автоматична генерація тексту до завантажених файлів буде відбуватись за допомогою підключеного хмарного сервісу із штучним інтелектом.

Створена діаграма прецедентів відповідно до усіх наведених вище вхідних параметрах зображена на рисунку 3.12.

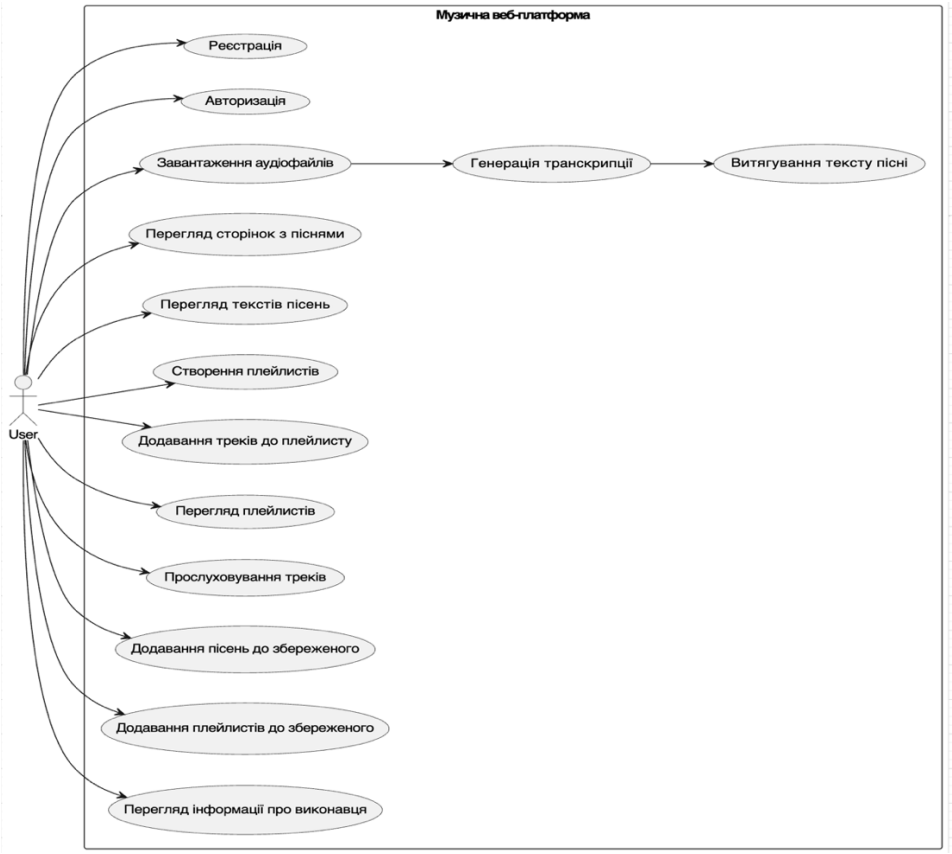

Рисунок 3.12 – Діаграма прецедентів проекту

Згідно з представленою діаграмою прецедентів у розробника з’являється можливість продумати структуру бази даних та описати основні таблички, які будуть використовуватись, а також визначити логіку взаємозв’язків між ними. Цей етап моделювання є критично важливим, адже саме на його основі формується каркас усієї серверної логіки системи.

Усі таблички в базі даних мають бути нормалізованими, щоб уникнути дублювання даних, зменшити надлишковість і забезпечити узгодженість записів. Нормалізація також дозволяє ефективно організувати структуру зберігання інформації.

3.3 Схема бази даних веб-платформи

Для того щоб створити надійну та оптимізовану базу даних нам потрібно остаточно вирішитись з основним функціоналом нашої платформи, описати які сутності має налічувати наша платформа, які вони мають мати можливості та рівні доступу на взаємодію з даними чи іншими користувачами.

Кожен користувач платформи повинен мати змогу зареєструватись на платформі чи увійти у свій особистий обліковий запис для подальшої взаємодії з платформою. Перегляд загальних списків завантажених музичних файлів, створених музичних збірок, додавання музичних файлів та музичних збірок до списків збереженого, прослуховування аудіо, додавання нових пісень, створення нових музичних підбірок, видалення музичної збірки чи пісні, перегляд профілів інших користувачів мають бути доступні для кожної людини яка взаємодіє з платформою. Автоматична генерація тексту до завантажених файлів буде відбуватись за допомогою підключеного хмарного сервісу із штучним інтелектом.

Опишемо усі основні сутності платформи у вигляді таблиць з їх відповідальність за певний функціонал.

User – табличка користувача, яка відповідає за збереження усієї інформації за певного автора чи слухача, та до якої будуть підв’язуватись більшість наступних таблиць, уся основна інформація зберігається у форматі itring.

User_auth – табличка де буде зберігатись уся чуттєва інформація про користувача, яка пов’язна з функціональністю реєстрації, авторизації, така як: захешований пароль, електронна поштова скринька та токени.

Song – таблиця для збереження усієї інформації про завантажені музичні файли на платформу, таку як: назва, автор, опис, текст, посилання на музичний файл і прев’ю та кількість прослуховувань

Playlist – таблиця що відповідає за дані про створену користувачем музичну збірку – назву, опис, автора, публічна чи приватна вона та посилання на обкладинку.

Song_to_playlist – таблиця яка призначена для створення зв’язку між таблицями song та playlist, щоб описати можливість одної збірки мати безліч музичних файлів.

Like_to_playlist – таблиця, що допомагає користувачам додавати списки відтворення до збереженого, для майбутнього прослуховування. Вона слугує для виконання зв’язку багато до багатьох

Like_to_song – аналогічна таблиця, тільки для музики.

Listens – таблиця, що відповідає за збереження прослуховувань.

Після опису кожної з необхідних сутностей з’являється можливість створити таблиці у базі. Діаграма структури бази даних зображена на рисунку 3.13

Рисунок 3.13 – Схема бази даних проекту

Послідовно виконуючи усі наведені вище дії вийшло створити зручну, нормалізовану та масштабовану базу даних, яка значно полегшить та пришвидшить усі майбутні кроки розробки веб-платформи.

Для створення цих таблиць було використано міграції, описані мовою TypeScript. Одна міграція відповідає за опис однієї таблиці, приклад опису однієї з міграцій наведений на рисунку 3.14.

```
/** @type {import('sequelize-cli').Migration} */
module.exports = {
  up: async (queryInterface, Sequelize) =>
    queryInterface.sequelize.transaction(async (t) => {
      await queryInterface.sequelize.query(
        `CREATE TYPE user_type AS ENUM ('user', 'admin');`,
        {
          transaction: t,
        },
      );

      await queryInterface.createTable(
        'user',
        {
          id: {
            type: Sequelize.INTEGER,
            primaryKey: true,
            autoIncrement: true,
          },
          username: {
            type: Sequelize.STRING,
            allowNull: false,
            unique: true,
          },
```

Рисунок 3.14 – Приклад створеної міграції

Крім того, використання полів created_at та deleted_at у всіх основних таблицях дозволяє ефективно логічно видаляти записи, зменшуючи ризик втрати важливих даних та спрощуючи аудит і скасування змін. Це також покращує масштабованість та спрощує реалізацію функціональності відновлення або аналізу історії змін.

4 ВЗАЄМОДІЯ КОРИСТУВАЧА ІЗ ВЕБ-ПЛАТФОРМОЮ ДЛЯ ПРОСЛУХОВУВАННЯ ТА РОЗПОВСЮДЖЕННЯ МУЗИЧНИХ ФАЙЛІВ

У цьому розділі будуть описані повні вимоги та продемонстровано як інсталювати програму, розгорнути її, що будь який користувач зміг зробити це самостійно за своїм комп’ютером. А також продемонстровано усі можливості програми після завершення її розробки.

4.1 Інсталяція та розгортання веб-платформи

На цьому етапі розробки здійснено повне встановлення та локальне розгортання музичної веб-платформи. Вона складається з двох окремих частин: клієнтської (фронтенд) та серверної (бекенд).

Після завантаження проекту на комп'ютер користувача першим кроком є створення файлів конфігурації .env для клієнтської та серверної частин. Ці файли містять змінні середовища, які зберігають важливі налаштування, такі як адреса бази даних, ключі доступу до хмарного сховища AWS S3, токен для голосового ШІ AssemblyAI. Без належного налаштування цієї інформації система не працюватиме належним чином.

Наступним кроком є встановлення всіх необхідних залежностей. Для цього перейдіть до відповідного каталогу, спочатку на сервері, а потім на клієнті, та виконайте команду npm install. Ця команда зчитує вміст файлу package.json та встановлює всі зовнішні бібліотеки, необхідні для роботи програми.

Після налаштування змінних середовища та встановлення залежностей можна безпосередньо запустити систему. Серверна частина запускається командою npm run

start і за замовчуванням запускається на порту 3000. Перевірте цей порт, наприклад, відкривши http://localhost:3000 у браузері або звернувшись до однієї з доступних кінцевих точок API.

Потім інтерфейс користувача запускається командою npm run dev і зазвичай відкривається автоматично в браузері, зазвичай на порту 5001. Він починає взаємодіяти з сервером через API, використовуючи адресу, визначену у змінних середовища.

Нарешті, перевірте, чи інтерфейс користувача правильно підключається до сервера. Для цього можна виконати прості дії: зареєструвати нового користувача, увійти, відтворити пісню, додати її до обраного та переконатися, що вся інформація надходить з API та правильно відображається в інтерфейсі користувача.

Це робить процес встановлення системи повністю прозорим та зручним для розробників та тестувальників. Це включає лише кілька ключових кроків: створення файлів .env, встановлення залежностей, запуск обох частин програми та перевірка їхнього з'єднання. Такий підхід дозволяє гнучко тестувати платформу, як локально, так і в хмарі, для операційного використання.

4.2 Огляд можливостей веб-платформи

У цьому пункті будуть продемонстровані основні можливості веб-платформи, які закладались перед початком її реалізації. Від персони користувача буде продемонстрована повноцінна взаємодія користувача з платформою. Буде продемонстровано функціонал авторизації та реєстрації, перегляду головного списку пісень та музичних збірок, списки збереженого, форми для додавання нових музичних файлів та створення плейлистів, перегляд облікових записів інших користувачів, додавання до збереженого та видалення пісень та плейлистів.

4.2.1 Реєстрація та автентифікації користувача

Першим етапом взаємодії з веб-платформою є процедура реєстрації та подальшої автентифікації користувача. Платформа зустрічає користувача формою реєстрації, що зображена на рисунку 4.1, де він може створити собі персональний обліковий запис для подальшого прослуховування чи розповсюдження музичних файлів.

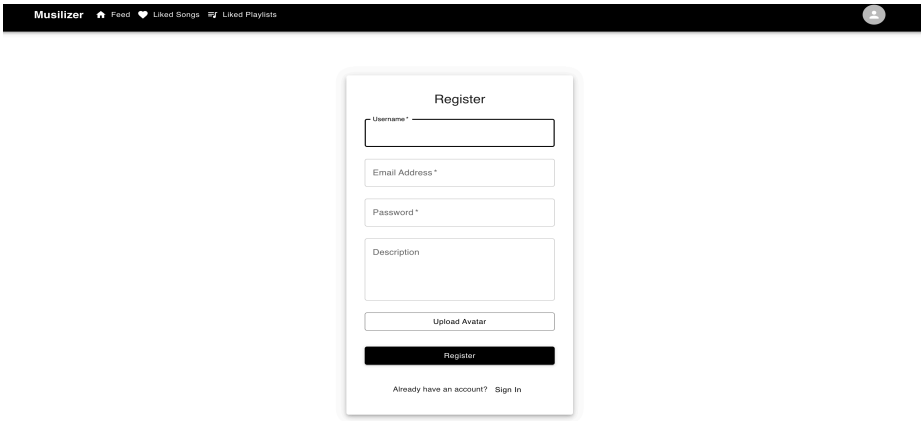

Рисунок 4.1 – Початковий екран з формою реєстрації

Для реєстрації акаунту користувачеві потрібно внести усі обов’язкові дані, які позначені * у полях для введення, та за бажанням додатати опис і зображення аватару.

Також якщо користувач уже має власний акаунт на платформі, йому достатньо натиснути на посилання ‘Sign In’ і тоді платформа надасть користувачеві можливість для входу в особистий профіль. Сторінка з формою для входу зображена на рисунку 4.2:

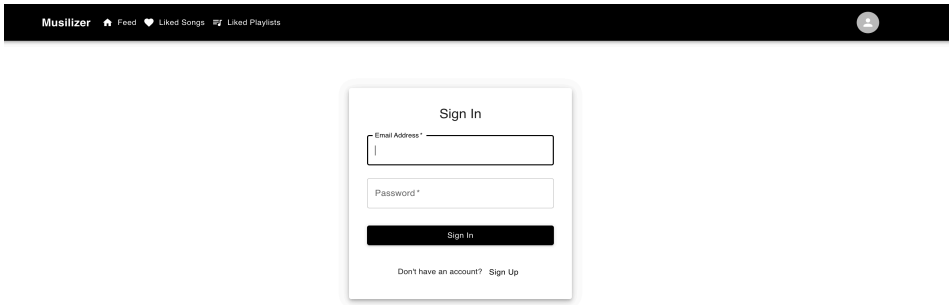

Рисунок 4.2 – Початковий екран з формою входу у систему

На цих двох екранах усі поля форм проходять валідацію як на frontend так і на backend частинах, щоб забезпечити правильну передачу даних та успішно зареєструвати або авторизувати користувача на платформу.

4.2.2 Головний екран, музичні файли та збірки

Одразу після авторизації слухача зустрічає головний екран де показана бібліотека завантажених музичних файлів (рисунок 4.3).

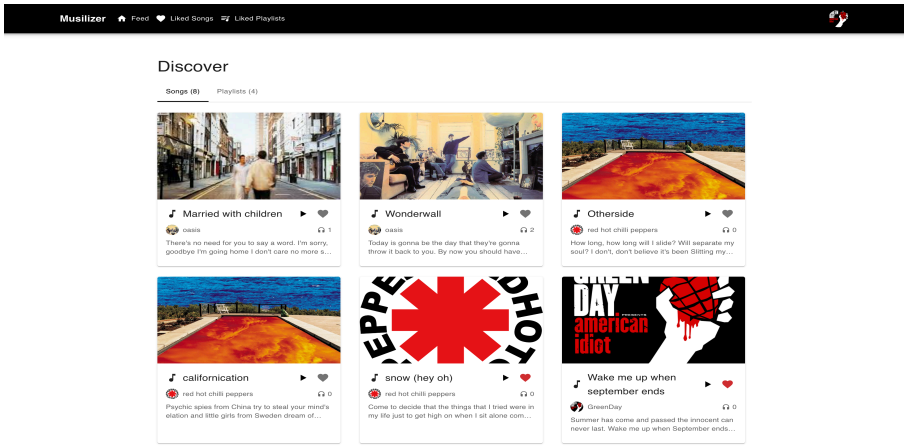

Рисунок 4.3 – Головний екран з бібліотекою завантажених музичних файлів

На цій сторінці відображені усі пісні які були завантажені іншими користувачами на платформу. Вони сортуються відповідно до часу завантаження на платформу: спочатку відображаються найбільш нові аудіофайли, надалі все йде у порядку спадання. Це надає можливість усім слухачам завжди залишатись у курсі новинок, та знаходити щось нове для себе за допомогою прослуховування свіжих виконавців.

На кожній карточці пісень відображається її назва, ім’я автору та шматок тексту пісні. Також наявна кількість прослуховувань кожного треку, які збільшуються одразу після закінчення прослуховування. Кожну публікацію можна прослухати за допомогою кнопки Play та додати до списку збереженого за допомогою кнопки Like.

Значок ноти перед назвою позначає що даних допис є музичним файлом який можливо прослухати, значок ноти зі списком – музичну збірку створену користувачем.

Процес прослуховування починається одразу після натискання на кнопку, виконання музичного файлу відображається у нижній частині екрану у спеціально розробленому плеєрі, який зображений на рисунку 4.4.

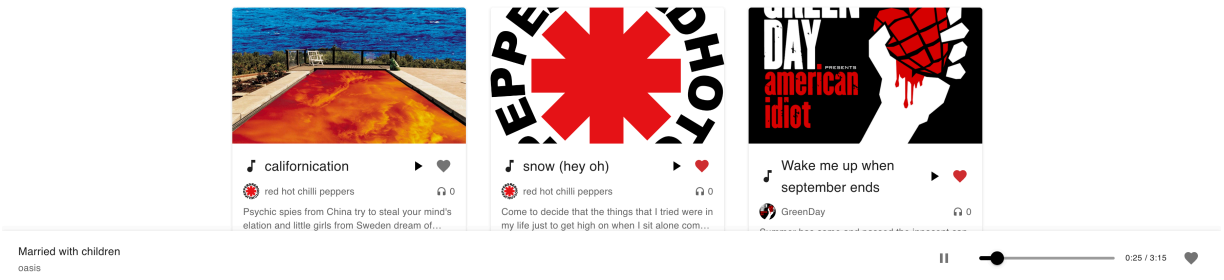

Рисунок 4.4 – Демонстрація вигляду програвача музичних файлів

У ньому є можливість додати пісню до збереженого, зупинити та продовжити виконання та перемотати на певний проміжок часу вперед чи назад.

Натискаючи на назву пісні у програвачі користувача перенаправить на сторінку з детальним описом музичного файлу, також туди можливо перейти натиснувши на назву пісні на головній сторінці програми у бібліотеці (рисунок 4.5).

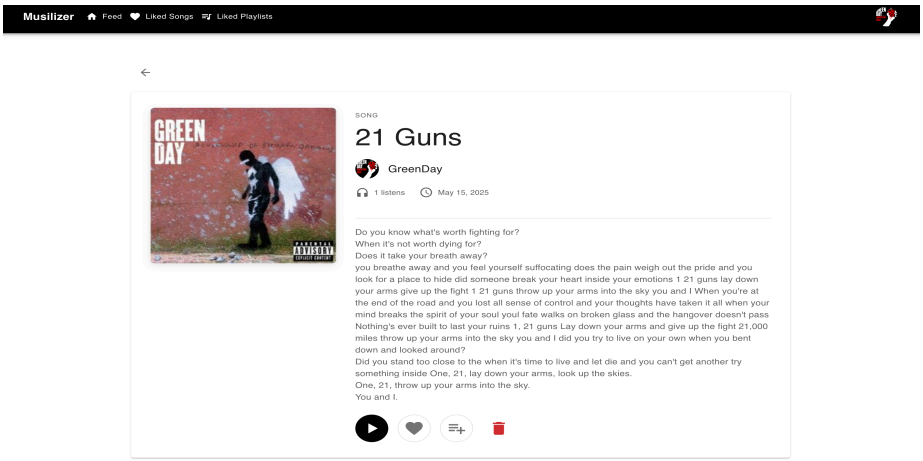

Рисунок 4.5 – Сторінка деталізованого огляду музичного файлу

Ця сторінка додає додаткову інформацію про завантажений музичний файл, відображену у більш зручному форматі, таку як: дата завантаження, повний текст. Після натискання на кнопку додавання у список відтворення з’являється модальне вікно з переліком наявних у власності музичних збірок для додавання. Цей функціонал продемонстрований на рисунку 4.6.

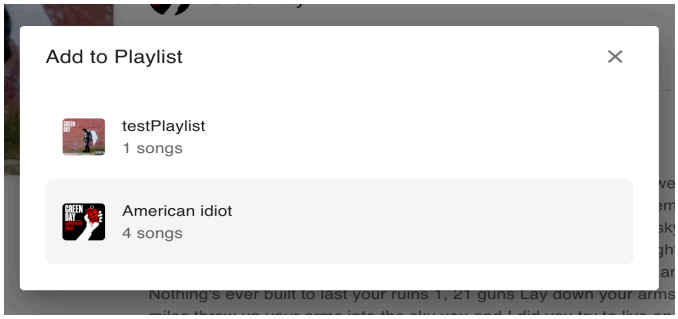

Рисунок 4.6 – Модальне вікно для додавання музичного файлу до створеної збірки

Повертаючись на головний екран користувач може змінити вкладку для перегляду бібліотеки створених музичних збірок замість треків. Картки існуючи плейлистів відрізняються наявністю іншої іконки перед назвою (рисунок 4.7).

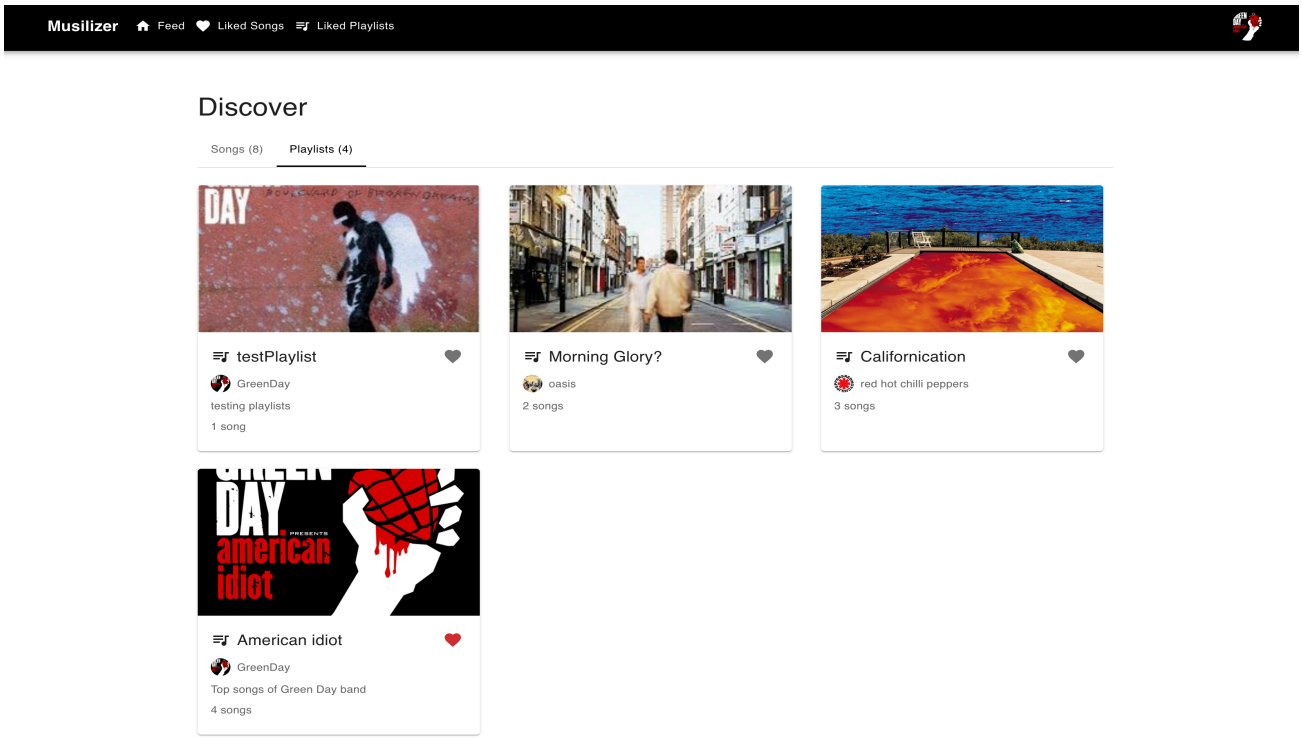

Рисунок 4.7 – Бібліотека створених музичних збірок

Кожна картка має схожі дані на наповнення карток треків, але головною відмінністю є показ кількості доданих до підбірки пісень. Схожим чином можна потрапити на сторінку з детальним описом кожної збірки та списком доданих до неї аудіофайлів. Усі аудіофайли у збірці відтворюються виключно у тому порядку який описаний у самому списку, тому користувач може сам обирати зручний йому порядок прослуховування та персоналізувати збірки. Сторінка огляду збірки зображена на рисунку 4.8.

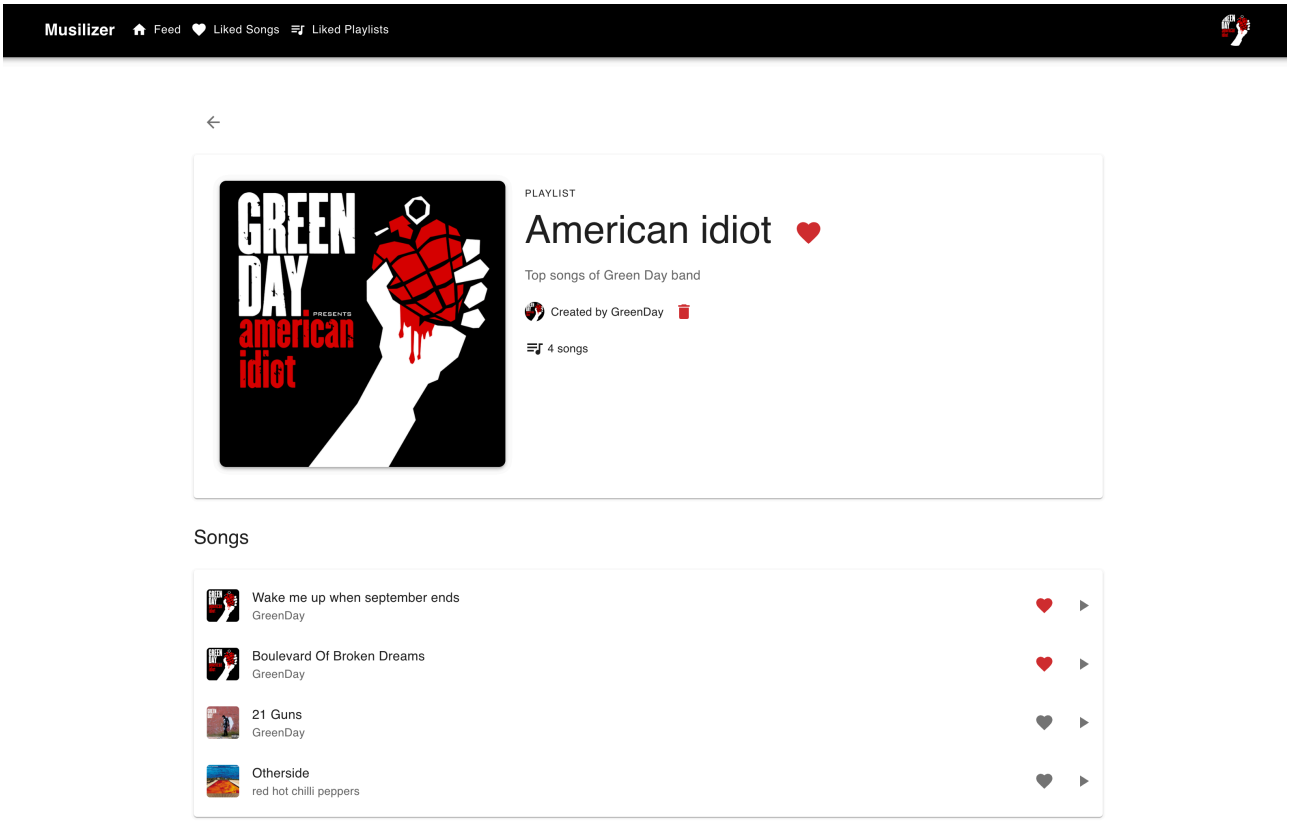

Рисунок 4.8 – Сторінка детального огляду музичної збірки

Також кожен користувач має можливість видаляти створені ним підбірки та завантажені музичні файли за допомогою кнопки видалення. Кнопка видалення відображається тільки у випадку якщо користувач є власником сутності.

4.2.3 Профіль користувача та списки збереженого

Кожен користувач платформи має змогу відвідати профіль іншого користувача чи власний. Це можна зробити натиснувши на кнопку перегляду свого профілю у випадаючому меню у правому верхньому краю екрану (рисунок 4.9).

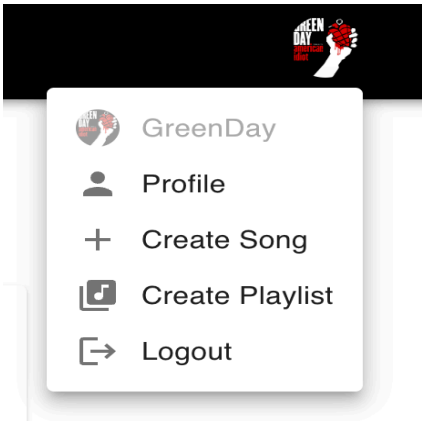

Рисунок 4.9 – Випадаюче меню користувача

Також перейти на профіль користувача є можливість натиснувши на ім’я автора під опублікованими ним аудіофайлів або музичних підбірок.

У профілі надається можливість отримати більш детальну інформацію про самого слухача чи автора, а також переглянути усі створені ним збірки і пісні за весь час використання на платформи. Профіль користувача зображено на рисунку 4.10.

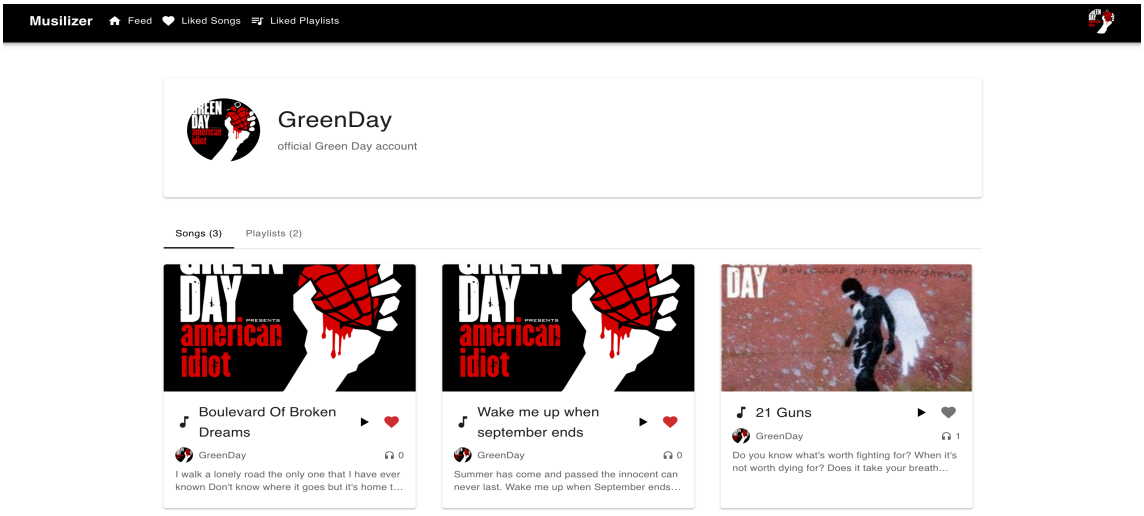

Рисунок 4.10 – Профіль користувача

На платформі існують списки збережених музичних файлів та музичних збірок для кожного користувача. Вони є приватними для кожного, тобто переглядати їх є можливість тільки у власника облікового запису.

Ці списки існують, щоб користувачі могли додавати до них сутності для майбутнього швидкого доступу та зручного користування.

Ці списки виглядають точно як бібліотеки на головному екрані, тільки вони є унікальними і приватними для кожного слухача. Їх вигляд зображено на рисунку 4.11.

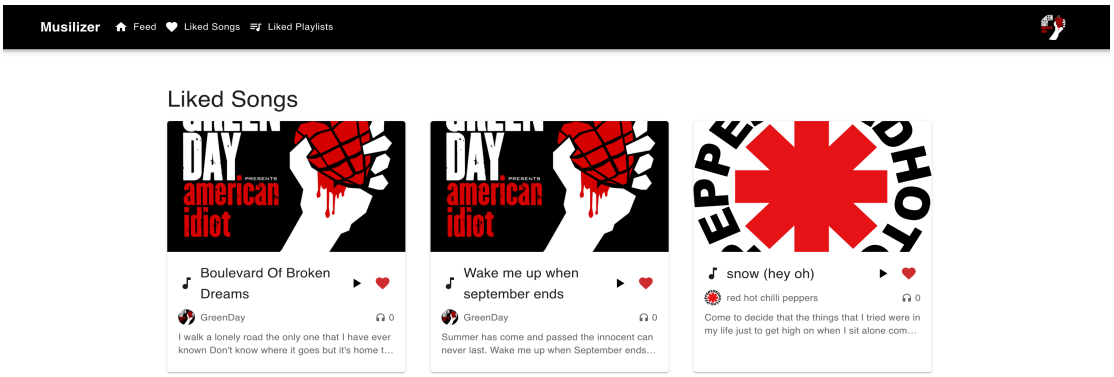

Рисунок 4.11 – Список збережених музичних файлів

Майже аналогічний інтерфейс розроблений для сторінки перегляду усі збережених музичних підбірок, тому великої потреби додавати ще один рисунок не виникає.

4.2.4 Завантаження музичних файлів та створення збірок

Для завантаження нового музичного файлу користувачеві треба скористатись випадаючим меню у правому верхньому кутку, де з’явиться відповідна для цього кнопка.

Користувач потрапить на сторінку з формою (рисунок 4.12), де йому потрібно заповнити усі необхідні поля.

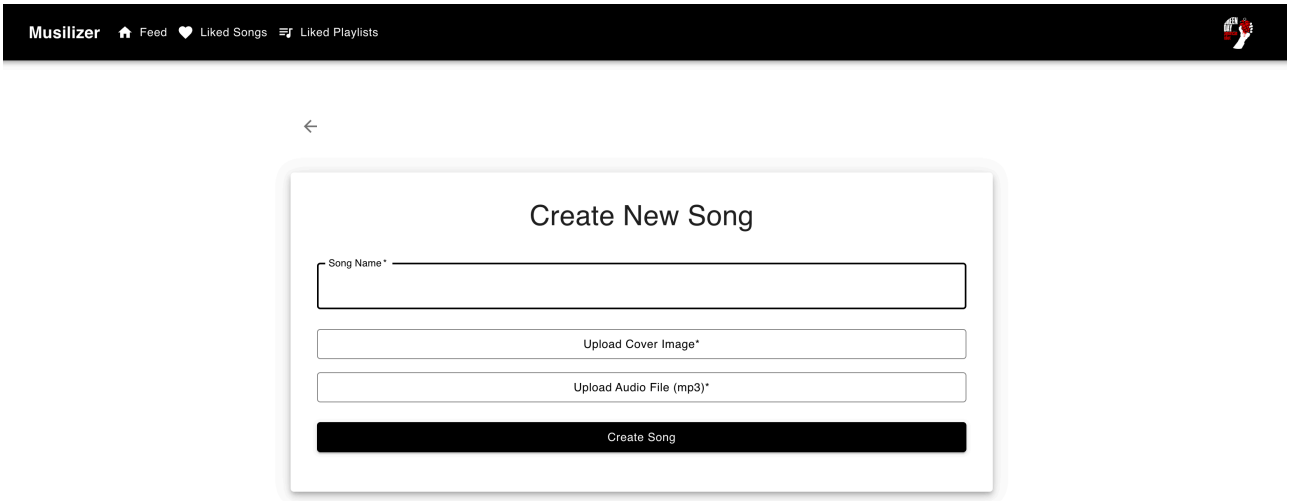

Рисунок 4.12 – Форма завантаження музичного файлу

У цій формі налічується 3 поля: перше тестове поле відповідає за присвоєння назви, інші два за прийом файлів обкладинки пісні та самого музичного файлу для прослуховування.

Одразу після заповнення форми та натискання на кнопку створення, платформа робить запит на серверну частину, яка починає зберігати інформацію до бази даних та робить запит на хмарний сервіс із штучним інтелектом для автоматичної генерації тексту для пісні. Цей процес займає декілька секунд після чого користувач отримує сповіщення про успішне створення та його одразу переадресовує на головну сторінку де з’являється його пісня першою у списку зі всіма завантаженими файлами та знегерованим текстом.

Процес створення музичної збірки відрізняється тільки відсутністю запиту на генерацію тексту та завантаженням аудіофайлу. Форма приймає назву, опис та зображення для обкладинки (рисунок 4.13).

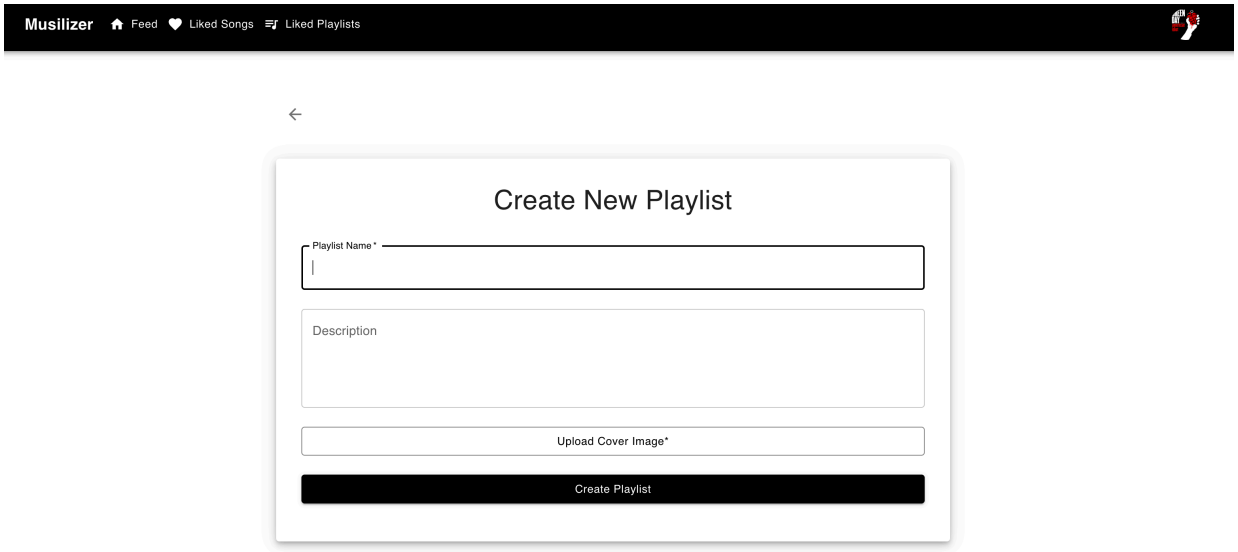

Рисунок 4.13 – Форма завантаження музичної збірки

Одразу після натискання на кнопку створення користувачу надходить повідомлення про успішне завантаження та переадресовує на головну сторінку, де він може знайти створений плейлист. Наповнювати плейлист надається можливість описаним раніше шляхом, зі сторінок детального огляду музичних файлів.

ВИСНОВКИ

В результаті виконання роботи було розроблено та впроваджено веб-платформу для прослуховування музики, що включає хмарне сховище аудіофайлів та інтеграцію голосового штучного інтелекту для обробки метаданих.

Було проведено детальний аналіз галузі дослідження та існуючих платформ, таких як Spotify та YouTube Music. Це визначило ключові вимоги до функціональності системи, зокрема: керування персональною музичною бібліотекою, обробка завантажених композицій, зручна авторизація та інтуїтивно зрозумілий інтерфейс користувача.

Для реалізації веб-застосунку було обрано сучасний технологічний стек: Nest.js для створення серверного компонента; React.js у поєднанні з Material UI для створення адаптивного клієнтського інтерфейсу; Sequelize як ORM для роботи з базою даних PostgreSQL; та зовнішні сервіси, такі як AWS RDS (для зберігання даних), AWS S3 (для зберігання аудіофайлів) та AssemblyAI (для обробки мовлення).

Впроваджена система дозволяє реєстрацію та авторизацію користувачів, завантаження музики, перегляд та прослуховування аудіофайлів, автоматичну генерацію транскрипцій на основі вмісту файлу та зберігання бази даних. Платформа забезпечує асинхронну обробку запитів та оптимальну роботу з великою кількістю одночасних користувачів.

Отримана веб-платформа являє собою сучасне, функціональне та розширюване рішення для потокової передачі медіа. Вона підходить для різних сценаріїв використання, як в освітніх проектах, так і в комерційних ініціативах, спрямованих на користувачів, які цінують контроль над власним музичним контентом та зручний онлайн-доступ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Михайлова І.Ю., Фундамент Д. Д. Дослідження використання штучного інтелекту на музичних платформах. IV Міжнародна наукова конференція «Інноваційна наука: пошук відповідей на виклики сучасності». Київ, Україна 30 травня 2025 року. Київ, 2025. С. 438-439.

2. TypeScript Documentation. TypeScript: website. URL: https://www.typescriptlang.org/docs/ (Дата звернення: 15.05.2025).

3. Node.js Documentation. Node.js: website. URL: https://nodejs.org/en/docs (Дата звернення: 10.05.2025).

4. React Documentation. React: website. URL: https://reactjs.org/docs/getting-started.html (Дата звернення: 10.05.2025).

5. NestJS Documentation. NestJS: website. URL: https://docs.nestjs.com/ (Дата звернення: 07.05.2025).

6. Sequelize Documentation. Sequelize: website. URL: https://sequelize.org/docs/v6/ (Дата звернення: 13.05.2025).

7. AWS S3 Documentation. Amazon Web Services: website. URL: https://docs.aws.amazon.com/s3/ (Дата звернення: 11.05.2025).

8. AWS RDS Documentation. Amazon Web Services: website. URL: https://docs.aws.amazon.com/rds/ (Дата звернення: 12.05.2025).

9. PostgreSQL Documentation. PostgreSQL Global Development Group: website. URL: https://www.postgresql.org/docs/ (Дата звернення: 10.05.2025).

10. AssemblyAI API Documentation. AssemblyAI: website. URL: https://www.assemblyai.com/docs/ (Дата звернення: 10.05.2025).

11. Material UI Documentation. MUI: website. URL: https://mui.com/material-ui/getting-started/ (Дата звернення: 17.05.2025).

12. Dotenv Documentation. dotenv: website. URL: https://www.npmjs.com/package/dotenv (Дата звернення: 15.05.2025).

13. Vite Documentation. Vite: website. URL: https://vitejs.dev/guide/ (Дата звернення: 13.05.2025).

14. Axios Documentation. Axios: website. URL: https://axios-http.com/docs/intro (Дата звернення: 08.05.2025).

15. ESLint Documentation. ESLint: website. URL: https://eslint.org/docs/latest/ (Дата звернення: 13.05.2025).

16. Prettier Documentation. Prettier: website. URL: https://prettier.io/docs/en/index.html (Дата звернення: 13.05.2025).

17. JWT.IO Documentation. Auth0: website. URL: https://jwt.io/introduction (Дата звернення: 08.05.2025).

18. Node.js Best Practices. GitHub: website. URL: https://github.com/goldbergyoni/nodebestpractices (Дата звернення: 12.05.2025).

19. Modern Fullstack Development with NestJS, React, and PostgreSQL. FreeCodeCamp. URL: https://www.freecodecamp.org/news/nestjs-react-fullstack-app/ (Дата звернення: 08.05.2025).

20. Prisma vs Sequelize: Choosing the Right ORM. Prisma Blog. URL: https://www.prisma.io/blog/prisma-vs-sequelize (Дата звернення: 14.05.2025).

ДОДАТОК А

Реалізація завантаження музичного файлу і генерації тексту за допомогою ШІ

Програмні засоби:

- мова програмування TypeScript;
- фреймворк NestJs;
- бібліотека AssemblyAI.

Assembly.service.ts:

```
import { Injectable } from '@nestjs/common';
import { AssemblyServiceInterface } from '../../ports/assembly-service.interface';
import { ConfigService } from '@nestjs/config';
import { AssemblyAI } from 'assemblyai';
import { formatSongText } from '../../common/format-song-text.util';

@Injectable()
export class AssemblyService implements AssemblyServiceInterface {
  private assemblyClient: AssemblyAI;

  constructor(private readonly configService: ConfigService) {
    this.assemblyClient = new AssemblyAI({
      apiKey: this.configService.get('assembly.apiKey'),
    });
  }

  async songToText(songUrl: string): Promise<string> {
    const transcript = await this.assemblyClient.transcripts.transcribe({
      audio: songUrl,
      speech_model: 'slam-1',
    });

    return formatSongText(transcript.text);
  }
}
```

Create-song.usecase.ts

```
import { Injectable, Inject } from '@nestjs/common';
import { UseCase } from 'src/core/shared-kernel/interfaces/use-case';
import { CreateSongRequest } from '../data/request/create-song.request';
import { CreateSongResponse } from '../data/response/create-song.response';
import {
  SongRepository,
  SongRepositoryType,
} from '../../ports/song.repository';
import { FileObjectName } from 'src/core/shared-kernel/secondary-adapters/s3/data/enum/file-object-name.enum';
import {
```

```
  S3ServiceInterface,
  S3ServiceInterfaceType,
} from 'src/core/shared-kernel/ports/s3-service.interface';
import { Song } from '../data/song.dto';
import { mockData } from 'src/core/shared-kernel/data/constants/mock-data.constant';
import {
  AssemblyServiceInterface,
  AssemblyServiceInterfaceType,
} from 'src/core/shared-kernel/ports/assembly-service.interface';

interface CreateSongArguments {
  audio: Express.Multer.File;
  params: CreateSongRequest;
  userId: number;
  cover?: Express.Multer.File;
}

@Injectable()
export default class CreateSongUseCase
  implements UseCase<CreateSongArguments, CreateSongResponse>
{
  constructor(
    @Inject(SongRepositoryType)
    private readonly songRepository: SongRepository,
    @Inject(S3ServiceInterfaceType)
    private readonly s3Service: S3ServiceInterface,
    @Inject(AssemblyServiceInterfaceType)
    private readonly assemblyService: AssemblyServiceInterface,
  ) {}

  public async execute({
    audio,
    cover,
    params,
    userId,
  }: CreateSongArguments): Promise<CreateSongResponse> {
    let song: Song = null;
    try {
      const formattedAudioName = this.s3Service.formatFileName(
        audio.originalname,
        FileObjectName.song,
      );

      const audioUrl = await this.s3Service.uploadFile(
        audio,
        formattedAudioName,
        [FileObjectName.song],
        true,
      );

      let coverUrl = mockData.cover;

      if (cover) {
        const formattedCoverFileName = this.s3Service.formatFileName(
          cover.originalname,
          FileObjectName.cover,
        );

        coverUrl = await this.s3Service.uploadFile(
          cover,
          formattedCoverFileName,
          [FileObjectName.cover],
          true,
        );
      }

      // send audio to assembly to get text
      const text = await this.assemblyService.songToText(audioUrl);

      song = await this.songRepository.create({
        name: params.name,
        cover_url: coverUrl,
        user_id: userId,
```

```
        audio: audioUrl,
        description: null,
        text: text || null,
      });
    } catch (err) {
      throw new Error('Error creating song');
    }

    return {
      id: song.id,
      name: song.name,
      cover_url: song.cover_url,
      user_id: song.user_id,
      audio: song.audio,
      description: song.description,
      text: song.text,
      listens: song.listens,
      created_at: song.created_at,
      deleted_at: song.deleted_at,
    };
  }
}
```

ДОДАТОК Б

Дослідження використання штучного інтелекту на музичних платформах

Апробація

Київ – 2025

ДОСЛІДЖЕННЯ ВИКОРИСТАННЯ ШТУЧНОГО ІНТЕЛЕКТУ НА МУЗИЧНИХ ПЛАТФОРМАХ

Фундамент Даніїл Дмитрович
Студент 4-го курсу, кафедра цифрових технологій в енергетиці
Інститут атомної та теплової енергетики
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Україна

Науковий керівник: Михайлова Ірина Юріївна
доцент кафедри цифрових технологій в енергетиці
Інститут атомної та теплової енергетики
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Україна

Із революцією у сфері технологій – появою штучного інтелекту, відбулись нововведення та зміни в усіх сферах життя людей. Штучний інтелект – це метод змусити комп'ютер чи програмне забезпечення «мислити» як людський мозок. Це досягається шляхом вивчення закономірностей роботи людського мозку та аналізу когнітивних процесів [1]. Музична індустрія не стала винятком, інтеграція штучного інтелекту (ШІ) додала безпрецедентні можливості для створення, персоналізації та розповсюдження контенту. Завдяки ньому платформи отримали можливість працювати та аналізувати неймовірні обсяги даних, що дозволило їм досягти більшої індивідуалізації користувацького досвіду та опанувати більш інноваційні методи виробництва музики.

Нововведені та редаговані алгоритми із використанням штучного алгоритму повністю змінили способи взаємодії користувачів з музичними платформами. Якщо проаналізувати найпопулярніші платформи, такі як Spotify або Youtube Music, можна із впевненістю сказати що вони теж використовують штучний інтелект. Spotify пропонував своїм користувачам генерацію щоденних музичних збірок на основі їх смаків, аналізуючи жанри, гурти та пісні вони прослуховували останнім часом. Слухачам дуже сподобалась така можливість, адже це допомогло їм відкрити для себе нові горизонти у музиці, до яких самостійно вони б не дійшли. Youtube Music використав штучний інтелект у схожому ключі, але трішки іншим методом. Замість того, щоб генерувати щоденні музичні збірки, вони обрали генерувати підбірки пісень на основі обраної для прослуховування. Саме таким способом, користувач може відкрити для

себе нові гурти із схожим стилем виконання та розвинути свій кругозір у музичній сфері. Такий рівень персоналізації за допомогою штучного інтелекту підвищує залученість і задоволеність користувачів, сприяючи більш захоплюючому досвіду прослуховування.

Окрім персоналізації платформ під потреби і бажання користувачів, деякі платформи пішли далі та запропонували користувачам створювати власні пісні не витрачаючи купи зусиль. Такі продукти як MuseNet від OpenAI, Sona та Beatoven AI пропонують користувачам можливість створювати власні, оригінальні музичні твори за допомого штучного інтелекту. Моделі що використовуються на цих платформах навчались на неймовірно великих базах даних з існуючими композиціями, що допомогло їм генерувати дійсно якісні та складні твори. Також ці сервіси пропонують генерування обкладинок на основі опису. Користувачеві достатньо повністю та детально описати яку він хоче згенерувати композицію, надати текст до неї та просто очікувати на результат, а продукт згенерує музичний твір у різноманітних жанрах, стилях та відкриє можливість проявити свою творчість.

Деякі платформи зайшли ще далі, наприклад Endel. Endel – це звуковий-оздоровча платформа на основі штучного інтелекту, розроблена берлінською компанією Endel Sound GmbH. Вона генерує персоналізовані звукові ландшафти – так звані «функціональні звукові середовища» – розроблені для покращення концентрації уваги, розслаблення, сну та загального самопочуття. Ці звукові ландшафти адаптуються в реальному часі на основі різних вхідних даних, таких як: час, погода, частота серцебиття та місцезнаходження [2].

Висновки. Штучний інтелект пропонує багатообіцяючі можливості для інновацій у музичній сфері та на музичних платформах. Провівши аналіз використання штучного інтелекту у різних видах платформ було визначено, що деякі з них намагаються покращити досвід користувачів додаючи більше персоналізації використовуючи його, інші навпаки націлені на проявлення творчості користувачем та навіть на оздоровчі процеси.

Список використаних джерел:

1. Штучний інтелект: історія, види та складові. Веб-сайт. URL: https://gigacloud.ua/articles/shho-take-shtuchnyj-intelekt-istoriya-vydy-ta-skladovi/. (дата звернення: 22.05.2025)
2. Endel. Endel Mobile app. Sound wellness anywhere. Веб-сайт. URL: https://endel.io/ (дата звернення: 22.05.2025)