МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____Юрій КОНДРАТЕНКО

«____»_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ВЕБЗАСТОСУНОК ДЛЯ ПРОСЛУХОВУВАННЯ ТА ЗБЕРІГАННЯ АУДІОФАЙЛІВ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____ Даніл ТКАЧЕНКО

«___»_____ 2025 р.

Керівник канд. техн. наук, доцент _____ Олександр МЄЩАНІНОВ

«___»_____ 2025 р.

Миколаїв – 2025

Чорноморський національний університет імені Петра Могили

(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп’ютерні науки
Освітня програма	Комп’ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Юрій КОНДРАТЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Ткаченка Даніла Вікторовича

((прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Вебзастосунок для прослуховування та зберігання аудіофайлів».

Керівник роботи: Мєщанінов Олександр Павлович , професор кафедри ІІС, д-р пед. наук, професор.

Затверджена наказом ЧНУ ім. Петра Могили від «26» травня 2024 р. № 111.

2. Строк представлення кваліфікаційної роботи «___» _____ 2025 р.

3. Результатом роботи є створення інтегрованої вебплатформи для прослуховування, та збереження музичних композицій. Система повинна забезпечувати зручний інтерфейс для користувачів, можливість прослуховування

музики онлайн, завантаження треків, створення плейлистів, а також авторизацію та персоналізацію профілю. Початковими даними є потреби користувачів у поєднанні функцій музичного стримінгу та базової обробки аудіо у вебсередовищі.

4. Перелік питань, що підлягають розробці: аналіз сучасного стану задачі створення комплексних музичних вебплатформ з можливістю прослуховування та зберігання аудіо; огляд існуючих рішень, їх функціональних переваг та обмежень; визначення функціональних вимог до інтегрованої системи, яка об’єднує музичний стримінг і елементи цифрової аудіоробочої станції; проєктування архітектури системи з урахуванням розподілення на фронтенд та бекенд, вибір технологій для реалізації.

Керівник роботи _____ Олександр МЄЩАНІНОВ

(Особистий підпис) *(Власне ім’я ПРІЗВИЩЕ)*

Здобувач _____ Даніл ТКАЧЕНКО

(Особистий підпис) *(Власне ім’я ПРІЗВИЩЕ)*

Дата видачі завдання «18» грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Вебзастосунок для прослуховування та зберігання аудіофайлів

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	18.12.2024	20.12.2024	
2	Аналіз предметної області та постановка задачі	21.12.2024	30.01.2025	
3	Огляд аналогів і публікацій по темі роботи	31.01.2025	01.02.2025	
4	Пошук підходящих технологій для реалізації застосунку	02.02.2025	02.02.2025	
5	Реалізація інтерфейсу (фронтенд)	03.02.2025	28.03.2025	
6	Реалізація серверної частини (бекенд)	29.03.2025	25.04.2025	
7	Налаштування бази даних на докері	26.04.2025	15.05.2025	
8	Перший попередній захист КР на засіданні комісії кафедри	16.05.2025	16.05.2025	
9	Другий попередній захист КР на засіданні комісії кафедри	30.05.2025	30.05.2025	
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	11.05.2025	17.05.2025	

Керівник роботи _____ Олександр МЄЩАНІНОВ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Даніл ТКАЧЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану

«29» січня 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувача групи 401 ЧНУ ім. Петра Могили

Ткаченка Даніла Вікторовича

на тему: **“ВЕБЗАСТОСУНОК ДЛЯ ПРОСЛУХОВУВАННЯ ТА ЗБЕРІГАННЯ АУДІОФАЙЛІВ”**

Актуальність даної роботи полягає у необхідності створення зручного вебзастосунку для зберігання та прослуховування музичних композицій, що поєднує функції стримінгових сервісів і забезпечує доступ до музичного контенту без потреби встановлення додаткового програмного забезпечення.

Об’єктом роботи є процеси зберігання, організації та відтворення музичних файлів у вебсередовищі.

Предметом роботи є вебтехнології та а рішення, що забезпечують функціонування застосунку для потокового прослуховування музики.

Метою роботи є розробка інтерактивного вебзастосунку, що дозволяє зберігати, організовувати та прослуховувати музичні файли у браузері.

У результаті виконання роботи було сформовано вимоги до системи, створено архітектуру платформи з використанням Next.js та NestJS, реалізовано REST API для взаємодії клієнта з сервером, побудовано інтерфейс користувача для завантаження, зберігання та прослуховування аудіофайлів, а також забезпечено зберігання особистих музичних бібліотек.

Дана робота складається з чотирьох розділів. У першому розділі здійснено огляд предметної області, проаналізовано музичні вебсервіси та сформульовано вимоги. У другому розділі розглянуто принципи побудови вебархітектур, особливості роботи з аудіо у браузері та обґрунтовано вибір технологій. Третій розділ присвячено реалізації застосунку, модулів авторизації, управління файлами та взаємодії з базою даних. У четвертому розділі наведено результати тестування системи, приклади сценаріїв використання та оцінку продуктивності. Загальний

обсяг роботи – 65 сторінок. Кваліфікаційна робота містить 12 рисунків, додатки та список із 30 використаних джерел.

Ключові слова: вебплатформа, музичний стримінг, аудіофайли, Next.js, NestJS, REST API.

ABSTRACT

to the qualification work by the student of the group 401 of Petro Mohyla Black Sea National University

Tkachenka Danila

“ DEVELOPMENT OF A WEB PLATFORM FOR LISTENING AND SAVING MUSIC FILES”

A relevance of this study lies in the need to create a convenient web application for storing and listening to music tracks, combining the functionality of streaming services and ensuring access to audio content without the need to install additional software.

An object of the work is the processes of storing, organizing, and playing audio files in a web environment.

A subject of the work is the web technologies and architectural solutions that ensure the functioning of a music streaming application.

A purpose of the work is to develop an interactive web application that enables users to store, organize, and listen to music files directly in a browser.

As a result of the work, the system requirements were defined, the platform architecture was designed using Next.js and NestJS, REST API was implemented for client-server interaction, a user interface for uploading, storing, and playing audio files was developed, and the functionality for managing personal music libraries was provided.

This work consists of four sections. The first section analyzes the subject area, reviews existing music web services, and formulates the system requirements. The second section features of working with audio in a browser, and the rationale for the chosen technologies. The third section is devoted to the implementation of the application, including modules for authentication, file management. The fourth section presents the results of system testing, usage scenarios.

The overall scope of the work is 65 pages. The thesis contains 12 figures, appendices, and a list of 30 references.

Key words: web platform, music streaming, audio files, Next.js, NestJS, REST API.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API – Application Programming interface

DAW – Digital Audio Workstation

HTML – HyperText Markup Language

JS – JavaScript

MIDI – Musical Instrument Digital Interface

VS Code – Visual studio code

ВСТУП

Сучасний світ дедалі більше переходить до цифрових технологій у всіх сферах життя, і музична індустрія не є винятком. Замість традиційного використання фізичних носіїв або десктопного програмного забезпечення для прослуховування музики, користувачі все частіше звертаються до вебзастосунків, які забезпечують зручний та швидкий доступ до аудіоконтенту без потреби встановлення додаткових програм. Саме стримінгові сервіси стали основною платформою для споживання музики у глобальному масштабі.

Розробка таких систем вимагає глибокого розуміння сучасних вебтехнологій, принципів побудови клієнт-серверних архітектур, методів роботи з аудіофайлами у браузері та забезпечення високої продуктивності.

Актуальність даної роботи полягає в необхідності створення доступного та ефективного вебзастосунку для потокового відтворення музики, який поєднує у собі зручність використання, швидкодію та функціональність, притаманну популярним стримінговим сервісам. Такий інструмент є особливо цінним у сучасних умовах, коли користувачі очікують наявності музики «тут і зараз» без обмежень, завантажень і складної установки.

Метою цієї роботи є проєктування та реалізація вебплатформи, що дозволяє користувачам завантажувати, зберігати та прослуховувати власні аудіофайли в браузері. У процесі дослідження розглянуто існуючі рішення у даній галузі, проаналізовано технологічні аспекти побудови подібних систем, а також розроблено повноцінний програмний продукт з використанням сучасного стеку технологій.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної сфери

Цей проект був створений для зберігання, створення та прослуховування аудіофайлів. Його мета це створити простий у використанні браузерний додаток для потокового прослуховування музики, подібний до Spotify.

В основі функціонування подібних вебсервісів лежить обробка аудіоконтенту в режимі реального часу з використанням сучасних вебтехнологій. Технології HTML5 та JavaScript забезпечують можливість безперервного відтворення аудіофайлів у браузері без встановлення додаткових плагінів чи програм.

З технічного погляду, вебзастосунки для прослуховування музики включають клієнтську частину, яка відповідає за взаємодію з користувачем, і серверну частину, що оперує даними, файлами та користувацькою інформацією. Сучасні серверні технології дозволяють ефективно обробляти запити, здійснювати безпечне зберігання даних у базах даних, реалізовувати механізми автентифікації та доступу. Однією з ключових особливостей таких систем є масштабованість, що дає змогу обслуговувати тисячі одночасних користувачів без втрати продуктивності.

Окрему увагу варто приділити особливостям роботи з аудіо в браузерному середовищі. За допомогою Web Audio API, Media Source Extensions, File API та інших сучасних інтерфейсів розробники можуть реалізовувати функції попереднього завантаження, обробки, мікшування, зміни гучності, візуалізації звуку тощо. Це відкриває широкі можливості не лише для відтворення музики, а й для створення інструментів редагування чи генерації звуку у вебзастосунках.

1.2 Огляд наявних аналогів

1.2.1 Spotify

Spotify це шведський музичний стримінговий сервіс, який з'явився на світ у 2008 році і швидко став одним із лідерів на глобальному ринку цифрової музики [1]. Ця платформа дозволяє слухати мільйони треків онлайн, зберігати їх у своїй бібліотеці, створювати власні плейлисти та відкривати нову музику завдяки алгоритмам, які пропонують персоналізовані рекомендації. Ви можете отримати доступ до Spotify через веб-версію, мобільні додатки (iOS, Android) або десктопну програму.

На 2025 рік сервіс обслуговує понад 600 мільйонів користувачів щомісяця, з яких більше 220 мільйонів мають платну підписку. Spotify є яскравим прикладом SaaS-рішення, яке акцентує увагу на масштабованості, персоналізації контенту та високому рівні UX-дизайну.

Spotify надає користувачам вражаючий функціонал:

– пошук та прослуховування музики: ви отримуєте доступ до понад 100 мільйонів треків, включаючи нові релізи, класичні хіти, незалежних артистів і навіть подкасти;

– плейлисти та добірки: користувачі можуть створювати свої власні плейлисти, підписуватися на вже готові або насолоджуватися автоматичними добірками, що формуються на основі їхніх інтересів;

– рекомендації: завдяки технологіям машинного навчання Spotify пропонує персоналізовані рекомендації, такі як "Discover Weekly", "Release Radar" і "Your Mixes";

– офлайн режим: доступний у преміум-версії, ви можете завантажити музику та слухати її без інтернету;

– кросплатформеність: синхронізація між різними пристроями дозволяє вам почати прослуховування на одному девайсі і продовжити на іншому.

Переваги Spotify:

– найбільша медіатека серед стримінгових сервісів: Spotify постійно нарощує свою колекцію треків, охоплюючи багато музичних жанрів, країн і виконавців;

– рекомендаційна система: це одна з найкращих на ринку, адже вона аналізує, що ти слухаєш, твої вподобання та активність у соціальних мережах;

– інтерфейс та досвід користувача: у Spotify простий і зрозумілий інтерфейс, який підходить для всіх від новачків до справжніх музичних фанатів;

– можливість інтеграції з соціальними мережами: ти можеш ділитися музикою, бачити, що слухають твої друзі, і створювати спільні плейлисти.

Недоліки Spotify:

– відсутність інструментів для створення або редагування музики: Spotify не підходить для музикантів чи продюсерів, які хочуть працювати безпосередньо на платформі;

– якість звуку: хоча бітрейт достатній (до 320 кбіт/с у преміум-версії), Spotify ще не запровадив безстратне Hi-Fi аудіо, як деякі конкуренти, наприклад, Apple Music чи Tidal;

– комерційні обмеження для артистів: виплати виконавцям часто вважаються недостатніми, попри великий обсяг прослуховувань і прибутки компанії;

– рекламна модель у безкоштовній версії: якщо ти не маєш підписки, тобі регулярно показують рекламу, й є обмеження на пропуски треків та без офлайн-доступу.

Значення Spotify для ринку. Spotify змінив, як ми слухаємо музику в цифрову епоху. Його модель з безкоштовними і платними підписками, дружній інтерфейс і увага до персоналізації зробили його популярним серед музичних сервісів. Але варто зазначити, що платформа більше орієнтована на слухачів, ніж на музикантів. В рамках проєкту, що поєднує стримінг та редагування, Spotify є яскравим

прикладом для стримінгу, але при цьому не акцентує увагу на творчому процесі продюсування.

1.2.2 SoundCloud

SoundCloud це популярна музична платформа, яка з'явилася в 2007 році в Берліні[1]. Вона дозволяє користувачам завантажувати, слухати та коментувати музику. Це хороше місце для незалежних музикантів і продюсерів, які хочуть ділитися своїми творіннями без співпраці з великими лейблами. Платформа зручна для як слухачів, так і творців, об'єднуючи елементи соціальної мережі та стрімінгу. SoundCloud доступна в інтернеті, а також має мобільні додатки для iOS і Android. Вона інтегрується з деякими цифровими аудіостанціями, такими як FL Studio і GarageBand.

Функції платформи:

– користувачі можуть завантажувати свої треки, додавати до них описи, жанри і теги. Є налаштування приватності для публічного або приватного прослуховування;

– можливість слухати музику онлайн, створювати плейлисти та підписуватись на виконавців;

– залишати коментарі до певних моментів у треку, це одна з особливостей SoundCloud;

– виконавці можуть переглядати статистику прослуховувань і лайків;

– SoundCloud також пропонує спосіб заробляти на музиці через програму для творців.

Переваги SoundCloud:

– простота для незалежних артистів: можна виставляти свою музику без лейблів;

– соціальна активність: лайки, коментарі та репости створюють відчуття спільноти;

– легкість в розповсюдженні: музиканти можуть вставляти SoundCloud-плеєри на свої сайти і блоги;

– слухачі можуть знаходити нові треки, які не завжди можна знайти на великих платформах.

Недоліки SoundCloud:

– обмежені інструменти для редагування звуку. Всі зміни потрібно робити в інших програмах;

– слабкі автоматичні рекомендації. Важче знаходити нову музику в порівнянні з Spotify чи YouTube Music;

– якість звуку у безкоштовній версії може не підходити для професійних музикантів;

– умови монетизації можуть бути менш вигідними в порівнянні з іншими сервісами.

SoundCloud у контексті комбінованих додатків. Для тих, хто хоче поєднати прослуховування музики з редагуванням треків, SoundCloud частково підходить. Він дозволяє поширення й прослуховування, але не має можливості редагування треків безпосередньо у браузері або на мобільних пристроях. Тож SoundCloud краще сприймати як платформу для спілкування артистів і слухачів, а не як повноцінне робоче середовище для створення музики.

1.2.3 BandLab

BandLab це безкоштовна платформа для створення та роботи з музикою, доступна як у веб-версії, так і через мобільні додатки. Започаткована в 2015 році, BandLab стала популярною завдяки своїй легкості у використанні та вбудованим інструментам, які допомагають музикантам створювати музику прямо у браузері або на мобільному пристрої [1].

Це не просто інструмент, а й соціальна мережа для музикантів. Ви можете співпрацювати з іншими в реальному часі, працюючи над проектами з будь-якого місця.

Можливості:

– платформа має потужний редактор, де можна створювати й мікшувати композиції з кількома аудіо- та MIDI-доріжками;

– є багато вбудованих інструментів, таких як синтезатори і семплери, а також колекція звукових ефектів;

– запрошуйте інших музикантів до свого проекту, щоб разом створювати та редагувати треки;

– створюйте музику на ходу, записуйте вокал або інструменти і діліться своїми роботами;

– всі проекти зберігаються в хмарі, так що можна легко перейти з одного пристрою на інший;

– діліться своїми треками, отримуйте відгуки, підписуйтеся на інших виконавців.

Переваги BandLab:

– всі основні функції доступні безкоштовно, що робить платформу хорошим вибором для новачків;

– можливість працювати на комп'ютері і мобільному пристрої;

– функції реальної колаборації виділяють BandLab серед інших;

– легко почати, навіть якщо ви не маєте досвіду;

– знайомтеся з музикантами з усього світу.

Недоліки BandLab:

– у порівнянні з професійними DAW, BandLab має менше складних інструментів;

– веб-версія та мобільний додаток можуть мати обмеження в якості звуку при великих проектах;

– досвідчені користувачі можуть вважати інтерфейс занадто спрощеним;

– відсутність підтримки деяких професійних плагінів може бути проблемою.

BandLab це зручна платформа, яка поєднує можливості онлайн-редактора і соціальної мережі для музикантів. Хоча вона має деякі обмеження в порівнянні з професійними DAW, все ж це чудовий інструмент для онлайн-музикантів, які хочуть співпрацювати та ділитися своєю творчістю.

1.2.4 Soundation

Soundation це онлайн-платформа для створення музики, яка дозволяє працювати з аудіо- і MIDI-треками прямо у веб-браузері. Вона з'явилася в 2010 році і пропонує простий інструмент для музикантів і тих, хто хоче експериментувати зі звуком без потреби встановлювати складні програми [1].

Ця платформа поєднує функції цифрової аудіо робочої станції (DAW) з можливістю спільної роботи та обміну музиками проектами, що робить її популярною серед початківців.

Функції:

– редактор композицій: Soundation має багатодоріжковий редактор, що підтримує аудіо- та MIDI-доріжки, так що можна створювати комплексні аранжування;

– віртуальні інструменти і ефекти: тут є набори віртуальних інструментів, такі як синтезатори та драм-машини, а також базові аудіоефекти, реверберація, еквалайзер і компресор;

– спільна робота: кілька користувачів можуть одночасно працювати над одним проектом, що заохочує спільне створення музики;

– хмарне зберігання: проекти автоматично зберігаються в хмарі, і доступ до них можливий з будь-якого пристрою з інтернетом;

– бібліотека звуків: Soundation пропонує велику колекцію лупів, семплів та пресетів, що допомагає швидко створити треки;

– укспорт: можна експортувати готові роботи в популярних форматах (WAV, MP3), що спрощує їх подальше використання;

– інтеграція з іншими сервісами: підтримка імпорту та експорту MIDI-файлів полегшує інтеграцію з іншими DAW.

Переваги Soundation:

– доступність: оскільки це веб-платформа, користувачам не потрібно встановлювати програми достатньо мати браузер і інтернет;

– спільна робота в реальному часі: музиканти можуть працювати разом, не зважаючи на те, де вони знаходяться;

– безкоштовна версія: платформа має безкоштовний план з основними функціями, що дозволяє ознайомитись із сервісом без витрат;

– велика бібліотека: колекція семплів і лупів допомагає швидко створювати якісні композиції;

– легка інтеграція: підтримка MIDI робить Soundation хорошим доповненням до професійних програм.

Недоліки Soundation

– обмежений функціонал: у порівнянні з десктопними DAW, тут менше інструментів і ефектів, що може обмежувати складні проекти;

– обмеження безкоштовної версії: у безкоштовному плані обмежене хмарне сховище та доступ до певних функцій;

– залежність від інтернету: потрібне стабільне з'єднання для комфортної роботи;

– відсутність плагінів: на відміну від професійних DAW, Soundation не підтримує VST та інші плагіни, що обмежує її функції.

Soundation це зручний інструмент для створення і редагування музики, що підходить для початківців. Він дозволяє швидко реалізувати ідеї без установки складних програм.

У проекті, який поєднує можливості потокового прослуховування музики і редагування, Soundation може стати прикладом веб-DAW з можливістю спільної роботи, навіть якщо вона має обмеження у порівнянні з професійними програмами.

1.2.5 Apple music

Apple Music це музичний стримінговий сервіс від компанії Apple, офіційно запущений у 2015 році. З моменту свого виходу він став одним із ключових гравців на ринку цифрового аудіо, конкуруючи зі Spotify, YouTube Music і Tidal. Apple Music дозволяє слухати мільйони треків онлайн, зберігати їх у бібліотеці, створювати плейлисти, отримувати персоналізовані рекомендації та користуватися потоковим радіо. Доступ до сервісу можливий через мобільні пристрої (iOS, Android), комп’ютери (macOS, Windows), а також через браузерну версію. [1]

Станом на 2025 рік Apple Music має понад 110 мільйонів активних підписників, і залишається ключовим елементом екосистеми Apple. Як SaaS-платформа, вона орієнтується на високу якість звуку, глибоку інтеграцію з пристроями Apple і преміальний користувацький досвід.

Apple Music пропонує великий набір функцій:

– прослуховування музики онлайн та офлайн: користувачі отримують доступ до понад 100 мільйонів треків із каталогів найбільших лейблів, інді-виконавців та ексклюзивних релізів;

– інтелектуальні рекомендації: система аналізує прослуховування користувача та пропонує добірки, плейлисти та новинки, які можуть зацікавити;

– плейлисти, радіо та Apple Music 1: користувач може створювати власні плейлисти, слухати авторські передачі та живі ефіри радіостанції Apple Music 1;

– безстратне аудіо та Dolby Atmos: одна з головних переваг сервісу підтримка Hi-Res Lossless і просторового звучання (Spatial Audio) без додаткових доплат;

– сімейний доступ і інтеграція з Siri: зручно використовувати сервіс у межах родини та керувати музикою голосом через Siri або інші пристрої Apple (HomePod, Apple Watch тощо).

Переваги Apple Music:

– висока якість звуку: сервіс підтримує Lossless Audio (до 24 біт / 192 кГц) та Dolby Atmos, що забезпечує кращу деталізацію у порівнянні зі Spotify;

– інтеграція з екосистемою Apple: ідеально працює з усіма продуктами компанії, включаючи iPhone, iPad, Mac, Apple Watch, Apple TV;

– рекомендації від редакторів: Apple Music, на відміну від Spotify, значною мірою спирається не лише на алгоритми, а й на ручні добірки від музичних редакторів та експертів;

– просторове аудіо (Spatial Audio): унікальний досвід прослуховування з ефектом присутності, доступний у великій кількості треків;

– розширені можливості для артистів через Apple Music for Artists: зручний інтерфейс аналітики та керування профілем виконавця.

Недоліки Apple Music

– менш ефективна алгоритмічна рекомендація: для деяких користувачів добірки можуть бути менш точними, ніж у Spotify;

– менш зручна соціальна інтеграція: функції на кшталт перегляду активності друзів чи спільних плейлистів реалізовані слабше;

– обмежена кросплатформеність: хоча існують додатки для Android і веб-версія, найкращий досвід доступний лише на пристроях Apple;

– відсутність безкоштовної версії: користування сервісом можливе лише за передплатою після пробного періоду.

Значення Apple Music для ринку. Apple Music є важливим гравцем у сфері цифрової музики. Його сила у високій якості звуку, ексклюзивах, редакційній підтримці та глибокій інтеграції з iOS і macOS. Хоча сервіс менш гнучкий у соціальному плані й має вужчу екосистему, ніж Spotify, він пропонує багатший аудіодосвід. У контексті проєкту, що зосереджений на стримінгу музики, Apple Music виступає прикладом преміального сервісу, орієнтованого на якість, естетику та інтеграцію з технікою Apple, але без функцій створення чи редагування треків.

1.3 Огляд публікацій

1. Chen, J., & Huang, S. (2020). Design and Implementation of a Cloud-Based Music Collaboration Platform. International Journal of Multimedia and Ubiquitous Engineering, 15(2), 67-78. У цій статті автори говорять про платформу для спільного створення музики в хмарі. Вони описують, як поєднання потокового аудіо, інструментів для редагування та комунікаційних функцій допомагає музикантам працювати разом у реальному часі. Також наведені приклади таких сервісів, як BandLab і Soundation, з акцентом на їхні плюси і мінуси. Важливо, що автори підкреслюють потребу в тому, щоб система була зручною і масштабованою для багатьох користувачів.
2. Kim, Y., & Lee, H. (2019). User Experience Evaluation of Music Streaming and Production Applications. Journal of Creative Music Systems, 4(1), 12-29. Ця публікація зосереджується на користувацькому досвіді в музичних додатках, які об'єднують функції прослуховування і створення музики. Дослідження аналізує популярні сервіси, як-от Spotify і FL Studio, як і нові платформи. Автори порівнюють інтерфейси, функціональність та їх адаптивність до різних пристроїв. Висновки вказують на важливість знайти баланс між складністю професійних інструментів і простотою для новачків, а також на те, як соціальні функції можуть залучити користувачів.

1.4 Постановка задачі

У наш час музика стала не просто способом розваги, а й важливим інструментом для самовираження, навчання, терапії та формування культурної ідентичності. Завдяки розвитку веб-технологій, все більше музичних сервісів переходять у браузер, що робить їх доступними для ширшої аудиторії як для професіоналів, так і для аматорів, які не мають спеціального програмного забезпечення чи потужного обладнання.

Популярні платформи, такі як Spotify, SoundCloud або YouTube Music, зробили прослуховування музики зручним і мобільним. Проте для багатьох користувачів важливо мати можливість не тільки слухати музику онлайн, але й зберігати улюблені треки, створювати власні плейлисти та організовувати колекції. Сучасні браузерні інструменти дозволяють реалізувати це без потреби в інсталяції додаткового програмного забезпечення або використання складних програм.

Універсальний вебсервіс для прослуховування та зберігання музики був би надзвичайно корисним для тих, хто прагне мати доступ до своєї колекції будь-де та будь-коли. Це також відкриває нові можливості для освітніх проєктів і творчих ініціатив, де важливий простий та зручний доступ до аудіоконтенту.

У часи, коли онлайн-освіта, дистанційна робота та цифрове дозвілля стрімко розвиваються, подібні інструменти викликають великий інтерес як у користувачів, так і в розробників.

Метою цієї кваліфікаційної роботи є створення вебзастосунку, який об'єднує функції музичного стримінгу та системи збереження треків. Застосунок повинен дозволяти користувачам завантажувати, прослуховувати, зберігати та організовувати музичні файли в зручному інтерфейсі прямо в браузері.

Об’єктом дослідження в цій роботі є інформаційна система, яка дозволяє створювати, редагувати та прослуховувати музичні композиції в вебсередовищі. Якщо говорити загальніше, це процеси, методи та технології, що забезпечують взаємодію користувача з музичним контентом через інтерфейс вебзастосунку.

Цей об’єкт охоплює широкий спектр аспектів: від технологій зберігання аудіофайлів та їх обробки на сервері до логіки рендерінгу інтерфейсу у браузері та взаємодії користувача з функціоналом системи.

У межах дослідження особлива увага приділяється механізмам взаємодії між клієнтською та серверною частинами, способам обробки мультимедійного контенту в реальному часі та принципам створення зручного й інтуїтивно зрозумілого інтерфейсу.

Таким чином, об’єкт включає в себе як технічну реалізацію (програмне середовище, фреймворки, API), так і логічну організацію роботи застосунку з точки зору кінцевого користувача.

Предметом цього дослідження є різні підходи та інструменти для створення вебзастосунку, який дозволяє редагувати та зберігати музичні треки прямо у браузері. Особливу увагу ми приділяємо технічним аспектам реалізації як клієнтської, так і серверної частин системи, їхній взаємодії, а також методам роботи з аудіофайлами на фронтенді та бекенді. У рамках дослідження ми розглядаємо, як побудувати зручний користувацький інтерфейс, обробляти звукові сигнали у браузері, а також організувати зберігання даних про музичні треки, проєкти та плейлисти користувача. Також ми аналізуємо технології та фреймворки, які забезпечують безперебійну та ефективну взаємодію між клієнтською та серверною частинами, включаючи передачу, обробку та збереження мультимедійного контенту в реальному часі. Усе це дозволяє нам дослідити можливості створення повноцінного онлайн-сервісу, що поєднує функціональність стримінгу та базового аудіоредактора в одному вебінтерфейсі.

Щоб досягти поставленої мети, потрібно вирішити кілька завдань:

– по-перше, необхідно проаналізувати наявні вебсервіси для стримінгу та редагування музики, щоб виявити їхні сильні та слабкі сторони;

– далі, слід спроєктувати архітектуру системи, використовуючи сучасний стек вебтехнологій зокрема, Next.js для клієнтської частини та NestJS для серверної логіки;

– також важливо реалізувати систему авторизації та особистих кабінетів для користувачів, де кожен зможе зберігати свої треки та проєкти;

– необхідно розробити інтерфейс для завантаження та редагування аудіо, який включатиме базові функції: обрізання, петлі (loop), регулювання гучності та додавання ефектів;

– крім того, потрібно забезпечити можливість збереження змін, експорту треків та відновлення проєктів;

– також варто створити інтерфейс для створення та прослуховування плейлистів, подібно до Spotify.

Висновки до розділу 1

У даному розділі було здійснено ґрунтовний аналіз предметної області, що дозволило чітко окреслити актуальність і доцільність обраної теми. На основі вивчення існуючих проблем та потреб користувачів сформульовано тему роботи, яка спрямована на створення інноваційного рішення в обраній сфері.

Огляд аналогічних програмних продуктів виявив як сильні сторони наявних рішень, так і їхні обмеження, що дає можливість обґрунтувати власний підхід до реалізації системи. Аналіз публікацій та технічної документації дозволив сформувати цілісне уявлення про сучасні підходи, технології та інструменти, які можуть бути використані під час розробки.

Тож, у цьому розділі закладено методологічну та теоретичну основу для подальшого проектування і реалізації програмного продукту відповідно до поставленої теми.

2 МОДЕЛІ, МЕТОДИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

1.1 Технології розробки системи

1.1.1 Visual Studio Code

Для створення вебзастосунку, який поєднує в собі функції музичного стримінгу та аудіоредагування, ми використовуємо середовище розробки Visual Studio Code (VS Code). Це сучасний, легкий і кросплатформений редактор коду, що забезпечує зручне середовище для розробки, тестування та підтримки вебпроєктів. Завдяки своїй гнучкості та безлічі розширень, VS Code став одним із найулюбленіших інструментів серед веброзробників [11].

У процесі реалізації проєкту VS Code слугує універсальним інструментом для написання як клієнтського коду на Next.js, так і серверної логіки на NestJS. Його переваги включають зручну навігацію по проєкту, автодоповнення, підтримку синтаксису TypeScript, інтеграцію з Git, а також можливість налаштування середовища відповідно до потреб розробника.

Рисунок 2.1 – Логотип VS Code [11]

Visual Studio Code пропонує потужний інструмент під назвою IntelliSense, який не лише підказує назви змінних та функцій, але й аналізує типи, імпорти та

залежності між модулями. Це допомагає уникнути багатьох помилок ще на етапі написання коду. Крім того, вбудовані засоби перевірки синтаксису підтримують чистоту та узгодженість усього проєкту, а розширення, такі як ESLint і Prettier, автоматично перевіряють, чи відповідає код прийнятому стилю.

Ще одна важлива перевага Visual Studio Code це вбудований термінал, який дозволяє запускати локальний сервер розробки, встановлювати пакети, виконувати збірку проєкту або запускати тести, не перемикаючись на інші додатки. Це значно підвищує ефективність роботи та пришвидшує процес розробки.

Інтеграція з системою контролю версій Git також реалізована на високому рівні. Розробник може створювати та перемикатися між гілками, робити коміти, переглядати історію змін і зручно вирішувати конфлікти прямо в редакторі. Це особливо важливо під час командної розробки або роботи з великими проєктами.

1.1.2 Docker

У процесі розробки та розгортання серверної частини вебзастосунку важливо забезпечити стабільність, портативність і зручність налаштування середовища. Для досягнення цих цілей у нашому проєкті ми використовуємо Docker платформу для контейнеризації, яка дозволяє запускати застосунки в ізольованих середовищах, що містять усе необхідне для їхньої роботи. [4] Це особливо важливо для бекенд-частини, реалізованої на основі NestJS, оскільки серверна логіка включає взаємодію з базою даних, обробку запитів, автентифікацію, збереження файлів та інші сервіси, кожен з яких потребує свого середовища виконання та налаштувань.

Docker дозволяє створити контейнер, який включає весь необхідний стек технологій: Node.js [10], встановлені залежності, змінні середовища, скрипти ініціалізації, а також, за потреби, зв’язок з іншими контейнерами, наприклад, з базою даних або сховищем файлів. Завдяки цьому контейнер із сервером NestJS можна легко запускати на будь-якій операційній системі, де встановлено Docker,

без додаткового налаштування середовища. Це значно спрощує як локальну розробку, так і деплой на хмарні платформи чи сервери.

Рисунок 2.2 – Логотип Docker [4]

Процес налаштування Docker починається зі створення Dockerfile це конфігураційний файл, який детально описує, які кроки потрібно виконати для створення образу. У цьому файлі ви вказуєте базовий образ Node.js, копіюєте код вашого застосунку, встановлюєте залежності за допомогою npm або yarn, а також задаєте команду для запуску сервера. Додатково створюється файл docker-compose.yml, який дозволяє запускати кілька сервісів одночасно, наприклад, сервер і базу даних PostgreSQL, в окремих, але взаємопов’язаних контейнерах.

Використання Docker гарантує, що ваш застосунок працюватиме однаково в усіх середовищах, зменшуючи ризик помилок, пов’язаних з різними версіями бібліотек, залежностей або операційних систем. Це також дозволяє швидко оновлювати сервер, масштабувати систему або змінювати налаштування без необхідності втручатися в основний код. Завдяки збереженню всіх налаштувань у вигляді конфігураційних файлів, система легко документується та підтримується.

1.1.3 NestJS

У процесі розробки серверної частини вебзастосунку для потокового відтворення та редагування музики ми використовуємо сучасний фреймворк NestJS. Цей фреймворк базується на Node.js і повністю написаний на TypeScript, що дозволяє поєднувати гнучкість JavaScript [5] із суворою типізацією, що, в свою чергу, підвищує стабільність і передбачуваність коду [2]. Завдяки своїй архітектурі, NestJS пропонує структурований підхід до створення бекенд-застосунків, схожий на принципи розробки в таких фреймворках, як Angular або Spring (Java), що робить його ідеальним для створення масштабованих, підтримуваних і модульних проєктів.

У контексті розробки цього застосунку NestJS відповідає за всі серверні процеси: маршрутизацію HTTP-запитів, автентифікацію користувачів, взаємодію з базою даних, завантаження та збереження аудіофайлів, а також реалізацію REST API, який використовує клієнтська частина, побудована на Next.js. Використання NestJS дозволяє чітко розділити логіку за допомогою контролерів, сервісів і модулів. Такий підхід робить проєкт легким для масштабування, як на рівні коду, так і на рівні команди розробників.

Фреймворк підтримує основи об’єктно-орієнтованого програмування, інверсію залежностей (Dependency Injection), модульність і розділення відповідальностей, що в результаті призводить до більш чистого та тестованого коду. Крім того, NestJS має глибоку інтеграцію з популярними ORM (Object-Relational Mapping) бібліотеками, такими як TypeORM і Prisma, що дозволяє швидко та ефективно працювати з базами даних. У цьому проєкті NestJS у поєднанні з Prisma використовується для створення моделей, міграцій та виконання SQL-запитів до бази даних PostgreSQL [10].

NestJS також підтримує створення мікросервісної архітектури та взаємодію через WebSocket, що відкриває можливості для реалізації функцій у реальному часі, наприклад, для колаборативного редагування треків або спільного

прослуховування. [2] Для захисту API використовуються стандартні механізми авторизації та автентифікації, такі як JWT, OAuth2 або сесії, які легко налаштовуються через вбудовані або сторонні бібліотеки.

У процесі розробки важливу роль відіграє система middleware, яка дозволяє обробляти запити ще до того, як вони потраплять до контролерів. Це може бути корисно для логування, перевірки прав доступу або кешування, також було використано Express [9]. Завдяки цим можливостям, серверна частина стає більш гнучкою, ефективною та масштабованою.

Приклад коду з проекту:

```
    public async create(dto: CreateTrackDto, image: 
Express.Multer.File, audio: Express.Multer.File) {
        const saveAudio = await 
this.fileService.createFile(FileType.AUDIO, audio);
        const saveImage = await 
this.fileService.createFile(FileType.IMAGE, image);

        const track = await this.prisma.track.create({
          data: {
            ...dto,
            popularity: 0,
            audio: saveAudio.filePath,
            image: saveImage.filePath,
            duration: saveAudio.duration},
        });
        return track;}
```

1.1.4 NextJS

Next.js це сучасний фреймворк для створення вебінтерфейсів на базі бібліотеки React, який активно використовується для розробки як простих SPA-застосунків (Single Page Application), так і складних масштабованих платформ. Коли мова йде про створення вебзастосунку для зберігання, відтворення та

редагування музичних треків у браузері, Next.js стає основною технологією для побудови клієнтської частини [3]. Його головна перевага полягає в універсальності, що дозволяє поєднувати серверний і клієнтський рендеринг, забезпечуючи високу продуктивність і хорошу індексацію сторінок, це важливо для будь-якого сучасного вебресурсу.

Next.js пропонує зручну структуру проєкту та автоматизує маршрутизацію, спираючись на файлову систему. Це значно полегшує навігацію в застосунку та зменшує обсяг шаблонного коду. У рамках цього проєкту клієнтська частина відповідає за візуалізацію інтерфейсу для користувача: відображення музичних треків, вбудованого аудіоплеєра, форм для завантаження та редагування файлів, а також модулів автентифікації, профілю користувача та інтерфейсів редагування треків. Усе це реалізовано за допомогою сучасних технологій, що робить процес зручним і інтуїтивно зрозумілим.

Однією з ключових переваг Next.js є його глибока інтеграція з API-інтерфейсами та можливість створення так званих API-роутів. Це дозволяє розміщувати легкі серверні обробники прямо у фронтенді, що особливо зручно для простих завдань наприклад, для перевірки автентифікації, обробки завантажених файлів або взаємодії з зовнішніми сервісами. Хоча основна логіка бекенду реалізована у NestJS, ця функціональність може бути використана для оптимізації деяких операцій, зменшення навантаження на основний сервер або прискорення взаємодії.

Крім того, Next.js має вбудовану підтримку TypeScript, що забезпечує типізацію на клієнті та підвищує надійність коду. У цьому проєкті ми використовуємо саме TypeScript, оскільки це сприяє кращому супроводу великого обсягу коду, покращує інтелектуальні підказки під час розробки та дозволяє уникнути багатьох помилок ще до компіляції.

Також важливою особливістю є зручна інтеграція Next.js з популярними UI-бібліотеками, такими як shadcn/ui, Tailwind CSS або Chakra UI. Це дозволяє створити адаптивний, сучасний і привабливий інтерфейс без зайвого навантаження

стилями чи кастомним CSS-кодом. У цьому застосунку інтерфейс користувача має вирішальне значення, адже користувач взаємодіє з елементами, які потребують інтуїтивності та швидкого реагування, такими як аудіоредактор, плеєр, таймлайн, кнопки ефектів тощо.

Next.js також пропонує гнучку систему маршрутизації, налаштування метаданих, динамічні маршрути, а також оптимізацію зображень, кешування і lazy-loading компонентів. [2] Це все позитивно впливає на загальну швидкість роботи та зручність користувацького досвіду. Особливо це важливо для вебзастосунку, який працює з мультимедійним контентом і має забезпечувати високу продуктивність у браузері.

Приклад коду з проекту:

```
    const MainLayout: React.FC<ContainerProps> = ({
children }) => {
        const [navOpen, setNavOpen] = useState(false);

        const toggleNav = () => {
            setNavOpen(!navOpen);
        };
        return (
            <Box sx={{ display: 'flex', flexDirection:
'column', minHeight: '100vh' }}>
                <Header toggleNav={toggleNav} />
                <Navbar open={navOpen} onClose={() =>
setNavOpen(false)} />
                <Box component="main" sx={{ flexGrow: 1
}}>
                    {children}
                </Box>
                <Footer />
            </Box>);};

    export default MainLayout;
```

1.1.5 Prisma

Під час розробки вебплатформи для прослуховування та редагування музики важливо було створити надійний та зручний зв'язок із базою даних. Для цього ми обрали Prisma, сучасний ORM, що добре працює з JavaScript та TypeScript, і його часто використовують у вебпроєктах [13]. На відміну від традиційного написання SQL-запитів, Prisma дозволяє описувати структуру бази даних у формі схеми, що потім автоматично генерує потрібний код для роботи з базою.

Prisma стала основою для зв’язку між сервером (зробленим на NestJS) та реляційною базою даних PostgreSQL. З її допомогою ми описали основні сутності проєкту, як-от користувачі, треки, плейлисти, та інші об'єкти.

Всі ці моделі описуються у файлі схеми (schema.prisma), а потім Prisma автоматично генерує клієнтську бібліотеку, що дозволяє розробникам легко виконувати запити.

Рисунок 2.3 – Логотип Prisma [13]

З Prisma значно простіше розробляти серверну логіку. Наприклад, щоб створити новий трек чи плейлист, знайти всі треки користувача або оновити інформацію про проєкт, достатньо викликати потрібні методи клієнта з зрозумілим синтаксисом, які ще на етапі компіляції перевіряють правильність запитів. Це

зменшує ймовірність помилок під час програмування і спрощує супровід коду. Інтеграція з TypeScript ще більше підвищує стабільність системи.

Ще один важливий момент це система міграцій, яка спрощує управління змінами в базі даних. Немає потреби писати SQL-скрипти вручну: досить описати зміни в схемі, а Prisma згенерує потрібну міграцію, що працюватиме в будь-якому середовищі. Це дозволяє синхронізувати структуру бази між членами команди та уникнути конфліктів.

У цьому проєкті Prisma також допомогла реалізувати складні зв’язки між об'єктами, наприклад, багато-до-багатьох між треками та плейлистами, або один-до-багатьох між користувачем та його проєктами. Завдяки декларативному опису ці зв’язки легко реалізувати в коді, що дозволяє спростити створення API-методів і тестування.

Щодо швидкості роботи, Prisma показала себе на висоті навіть із великими обсягами даних. Завдяки автоматично генерованим запитам, система швидко реагує, особливо коли багато користувачів працюють одночасно з аудіофайлами.

Зрештою, завдяки Prisma вийшло реалізувати серверну частину платформи, що відповідає вимогам сучасних вебзастосунків: стабільність, швидкість, зручність. Цей ORM став важливим інструментом для роботи з даними, що підтверджує правильність його вибору для нашої платформи.

1.1.6 HTML і CSS

При створенні сучасного вебдодатку для прослуховування та редагування музики використовували різні технології фронтенд-розробки, де ключовими є HTML і CSS. Ці технології є основою для створення будь-якого вебінтерфейсу, незалежно від його складності.

HTML формує структуру і зміст сторінки. Він визначає порядок елементів, таких як заголовки, параграфи, кнопки тощо. У нашому проекті HTML допомагає створити логічну структуру інтерфейсу. Наприклад, основні компоненти, як

музичний плеєр і список треків, реалізовані за допомогою семантичних HTML-елементів. Це не тільки забезпечує коректне відображення у браузері, але й відповідає нормам доступності, що важливо для сучасного вебу.

Семантичний HTML (такі теги, як <header>, <nav>, <main>, <section>, <article>, <footer>) покращує SEO-індексування і забезпечує кросбраузерну сумісність, що робить код більш читабельним. Це також важливо для користувачів, які використовують програми для читання з екрану.

CSS, з іншого боку, відповідає за стиль і візуальне оформлення. За його допомогою ми реалізували адаптивний дизайн, візуальні ефекти та загальну естетику додатку. Стилі керують виглядом елементів: їх розмірами, кольорами і анімаціями.

Я використовув можливості CSS, такі як:

– flexbox і CSS Grid для компонування;
– CSS-змінні для управління кольорами, темами та шрифтами;
– анімації для плавних переходів станів інтерфейсу;
– Media Queries для підтримки різних пристроїв.

Особливу увагу я приділив інтерфейсу редактора аудіо з інтерактивними елементами. CSS допоміг створити їх таким чином, щоб вони були зрозумілі та зручні для користувачів. Ми використали кольорове кодування і підказки для полегшення взаємодії.

Розробка стилів проходила з урахуванням принципів модульності. Це дозволило кожному компоненту мати власні стилі, знизити ймовірність конфліктів і спростити оновлення коду. Також ми використовували утилітарні бібліотеки CSS, зокрема Tailwind CSS, щоб пришвидшити процес верстки.

Отже, HTML і CSS у цьому проекті стали основою для створення простого, адаптивного і привабливого інтерфейсу, що відповідає сучасним вимогам до вебзастосунків. Це забезпечує легкість у використанні та підтримці проекту.

Висновки до розділу 2

У цьому розділі було здійснено аналіз сучасних технологій, які можуть бути використані для розробки програмної системи. Розгляд таких фреймворків і інструментів, як NestJS (для побудови серверної частини), Next.js (для клієнтського інтерфейсу), Prisma (як ORM для взаємодії з базою даних) і Docker (для контейнеризації та розгортання), дозволив обґрунтувати вибір технологічного стеку на основі критеріїв продуктивності, масштабованості, зручності розробки та підтримки.

NestJS забезпечує модульну структуру та підтримку TypeScript, що сприяє написанню масштабованого та підтримуваного бекенду. Next.js поєднує переваги серверного рендерингу та клієнтської SPA-архітектури, що важливо для побудови зручного та швидкого інтерфейсу. Використання Prisma дозволяє суттєво спростити роботу з базою даних завдяки типізації та автоматичній генерації запитів. Docker, своєю чергою, забезпечує стабільне середовище виконання, полегшує розгортання та підвищує портативність розробленої системи.

У результаті аналізу зроблено висновок, що обрані технології повністю відповідають вимогам до функціональності, гнучкості та подальшого масштабування програмного продукту. Це дає змогу перейти до наступного етапу проєктування та реалізації системи на основі сформованого технологічного стеку.

3 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1 Опис вхідних даних та структури системи

При розробці інтерактивної веб-платформи для прослуховування музики, увагу було зосереджено на обробці даних та створенні надійної технічної бази. Це допомогло забезпечити стабільну роботу між клієнтською та серверною частинами, навіть при високих навантаженнях, а також пристосуватися до потреб різних користувачів: слухачів, музикантів і продюсерів.

Основні типи даних, що обробляються платформою, включають:

– аудіофайли (mp3, wav, flac): користувачі завантажують їх для прослуховування або редагування. Файли зберігаються в сховищі, а метадані – у базі даних за допомогою ORM Prisma;

– метаінформація треків: тут міститься назва, виконавець, жанр, дата публікації, тривалість, обкладинки тощо. Ці дані можна вводити вручну або отримувати з аудіофайлів автоматично;

– інформація про користувачів: це дані облікових записів, як-от ім’я, email, зашифрований пароль і список завантажених треків, плейлистів і проєктів, а також роль у системі (звичайний користувач, адміністратор або продюсер);

– API-запити: це вхідні запити з клієнтської частини (Next.js) до серверної (NestJS), наприклад, на отримання списку треків або зміна інформації про користувача.

Архітектура системи. Платформа має клієнт-серверну архітектуру з чітким розподілом обов’язків. Вся система складається з трьох частин: фронтенда, бекенда та бази даних.

Фронтенд, створений на Next.js, відповідає за інтерфейс, взаємодію з редактором, візуалізацію даних і маршрутизацію. Ми активно використовуємо HTML, CSS та JavaScript для створення UI-компонентів.

Бекенд на NestJS обробляє запити, автентифікацію, валідацію даних і керування медіавмістом. Він обслуговує REST API-запити та WebSocket-з’єднання для функцій спільного редагування.

Сховище даних реалізовано через базу даних PostgreSQL, із якою система працює через ORM Prisma. Це забезпечує безпечну та структуровану роботу з моделями, що описують користувачів, аудіотреки, редагування і інші елементи платформи.

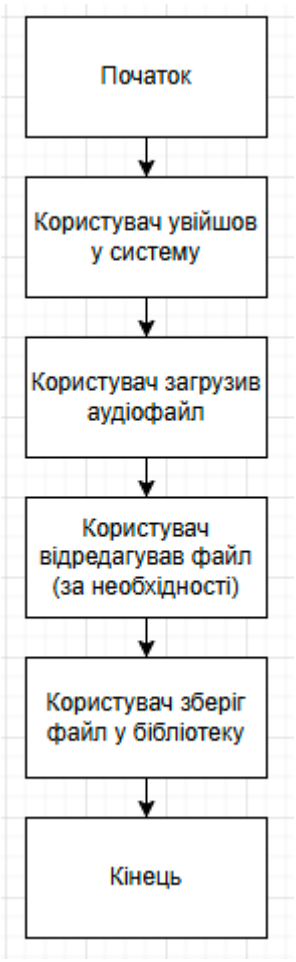

Рисунок 3.1 – Робота застосунку з точки зору користувача

3.2 Проектування і моделювання

Одним із важливих етапів розробки вебплатформи для прослуховування та редагування музики стало створення її візуальної частини. Фронтенд це той інтерфейс, який робить взаємодію з основними функціями платформи зручною. Це стосується відтворення треків, редагування композицій та управління профілем, плейлистами, проектами та аудіофайлами.

Коли розробляли інтерфейс, врахували сучасні вимоги до UX/UI дизайну. Це означає, що інтерфейс повинен бути зрозумілим, привабливим, швидким і адаптивним для різних пристроїв, від настільних комп'ютерів до мобільних телефонів. Платформа має бути зручною як для звичайних слухачів, так і для музичних продюсерів, які потребують потужного редактора з великою кількістю функцій.

Для клієнтської частини ми обрали фреймворк Next.js. Він поєднує React для створення інтерфейсів і серверний рендеринг, що допомагає з продуктивністю і SEO. Всі компоненти розробляються окремо, що дозволяє повторно використовувати код і швидко змінювати інтерфейс.

Для стилізації ми використали бібліотеку Tailwind CSS. Вона дозволяє швидко створювати адаптивні стилі без складних CSS-файлів. Завдяки цьому ми отримали сучасний та мінімалістичний дизайн, акцентуючи увагу на зручності: великі кнопки, темна тема, логічне групування функцій.

Основні екрани інтерфейсу включають:

Головну сторінку, де показуються нові треки, популярні композиції, рекомендації та нещодавно прослухані альбоми. Це стартова точка для користувача.

Аудіопрогравач, який завжди під рукою внизу екрану. Тут є кнопки для відтворення, перемотування, зміни гучності, інформація про трек і виконавця.

Профіль користувача, де зберігаються особисті дані, історія прослуховування, треки, проєкти і плейлисти. Також є можливість редагування профілю.

Музичний редактор це окрема сторінка з інтерфейсом, що має таймлайн, доріжки, панель інструментів і інші елементи. Для цього використані Canvas/Web Audio API разом із React.

Бібліотека, де можна переглядати, фільтрувати та редагувати треки, а також додавати їх до плейлистів. Є пошук, сортування за жанром, автором і настроєм.

Переходи між сторінками реалізовані через маршрутизацію Next.js, що забезпечує плавну навігацію.

На етапі моделювання інтерфейсу ми створили макети в Figma, що дозволило протестувати важливі сценарії до початку розробки. Було проведене внутрішнє юзабіліті-тестування, за результатами якого ми внесли деякі зміни для покращення взаємодії.

Головна сторінка вебплатформи виконана в простому сучасному стилі, з акцентом на зручність для користувачів і приємний вигляд. Верхня частина екрану займає героїчна секція з яскравим фоновим зображенням концерту, що створює атмосферу музики ще до початку роботи з сайтом.

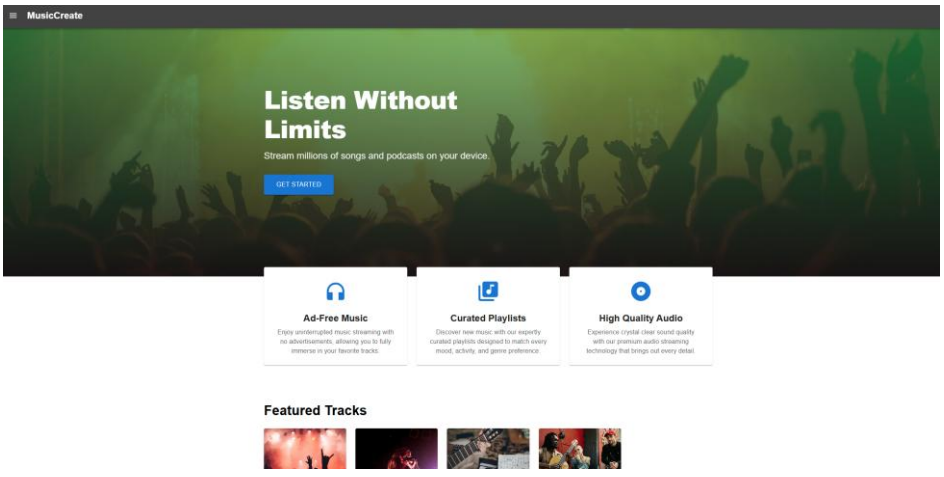

Рисунок 3.2 – Головна сторінка застосунку

В центрі слоган “Listen Without Limits”, який закликає користувача діяти. Під ним написано коротко про платформу: “Stream millions of songs and podcasts on your device.” Це чітко пояснює, що пропонує сервіс. Під цим є синя кнопка “Get Started”, яка запрошує користувача зануритися в платформу або зареєструватися.

Далі йде блок з трьома картками, які показують основні переваги застосунку:

– Ad-Free Music слухайте музику без реклами;
– Curated Playlists чудово підібрані плейлисти на різний випадок;
– High Quality Audio якісний звук, який відтворює кожну деталь.

Ці елементи створюють позитивне перше враження про платформу.

Трохи нижче розташований блок “Featured Tracks”, сітка з обкладинками альбомів і треків, що представляють популярний контент. Він допомагає швидко ознайомитися з найкращими композиціями на сайті. Кожен трек подається як окрема картка з зображенням і можливими інтерактивними кнопками.

Навігаційна панель вгорі. Ліворуч розміщено логотип “MusicCreate”, а праворуч кнопки входу, реєстрації або профілю. Меню виклику зліва вказує на наявність додаткових функцій.

Загалом дизайн сторінки простий, контрастний та привабливий. Це дозволяє швидко орієнтуватися, заохочує користувача до дії і демонструє цінність платформи з перших секунд.

Сторінка з файлами користувача. Сторінка для перегляду та пошуку треків у застосунку MusicCreate виглядає зручно і просто. Вона допомагає швидко знайти музику, яка вам до вподоби, у знайомому стилі, як і головна сторінка, з акцентом на легкість використання.

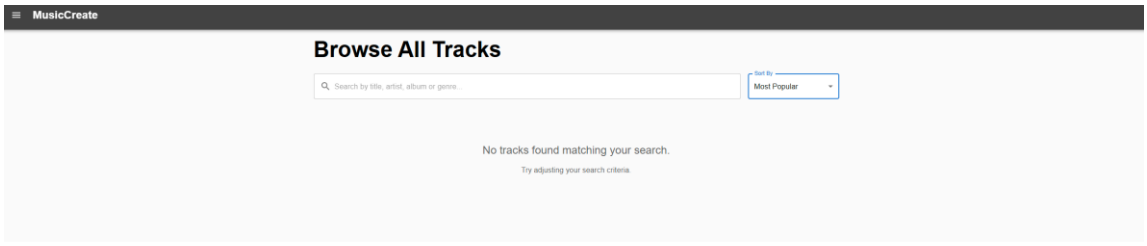

Рисунок 3.3 – Файли користувача

На верху написано “Browse All Tracks”, щоб ви відразу знали, що знаходитесь у загальному каталозі треків. Під заголовком є рядок для пошуку, де можна знайти пісні за назвою, виконавцем, альбомом чи жанром. Це справді спрощує процес, особливо якщо в бібліотеці багато контенту.

Праворуч від пошукового поля є список сортировки, що дозволяє впорядкувати результати. На даний момент доступна опція “Most Popular”, що допомагає знайти найпопулярніші треки відразу. Це буде зручно новим користувачам, які хочуть одразу дізнатися про топові композиції.

Якщо за вашими критеріями нічого не знайдено, на екрані з’явиться повідомлення “No tracks found matching your search” з ідеєю “Try adjusting your search criteria”, що підказує, як можна змінити пошук. Це робить користування простішим і допомагає не заплутатись у інтерфейсі.

Також був створений фільтр за популярністю та алфавітом.

Сторінка для завантаження нового аудіофайлу.

Ця сторінка дозволяє додати новий трек у додатку MusicCreate. Вона має формат покрокового майстра, який ділить весь процес на три етапи:

Першим є інформація про трек. Тут користувач заповнює основні дані:

– title, назва треку (обов'язкове поле);

– artist, ім’я виконавця (обов’язкове поле);

– genre, вибір жанру зі списку;
– description, необов’язковий опис треку;
– release date, вибір дати релізу через календар або введення вручну.

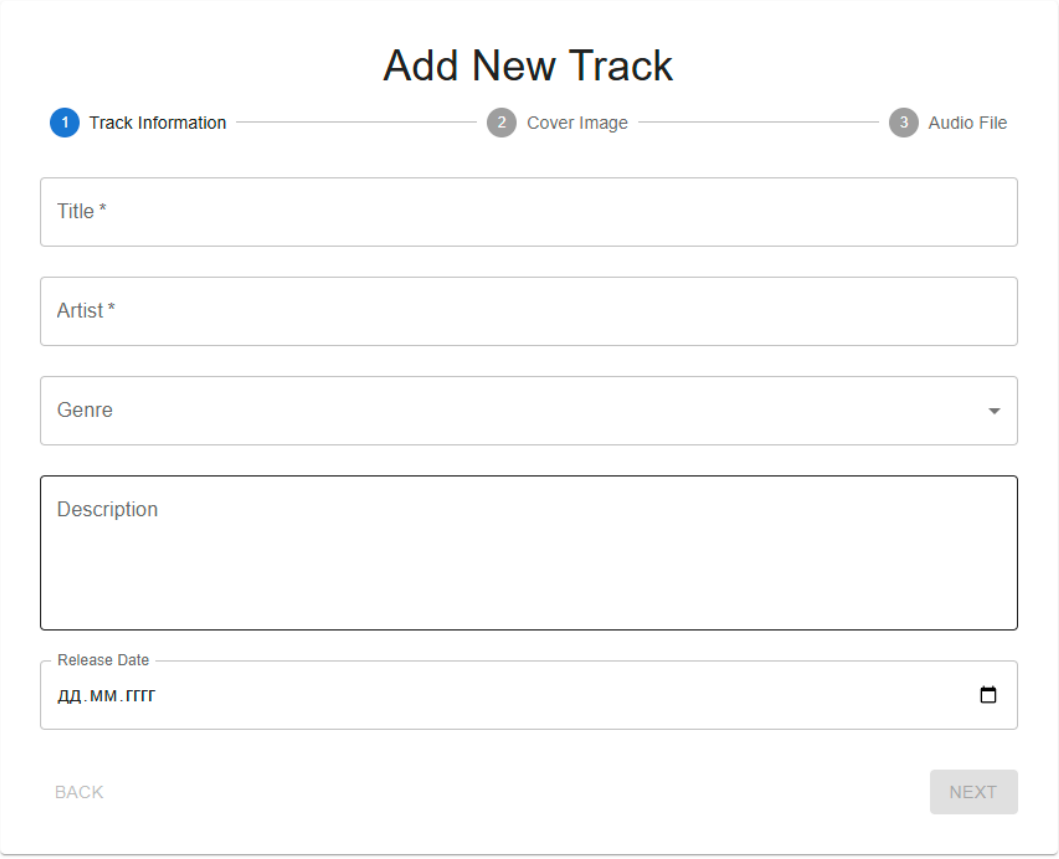

Рисунок 3.4 – Перший етап створення файлу

Всі поля згруповані, і є підказки, які роблять користування простішим. Під полями видно активну вкладку (синім кольором), а кнопка NEXT переходить до наступного кроку завантаження обкладинки.

На другому етапі додавання треку користувач має можливість завантажити обкладинку для свого треку або альбому. Це зображення стане візуальним представленням треку у бібліотеці, під час пошуку, у списках відтворення та на самій сторінці треку. Користувач може або перетягнути файл у спеціально відведену зону, або вибрати його через файловий менеджер.

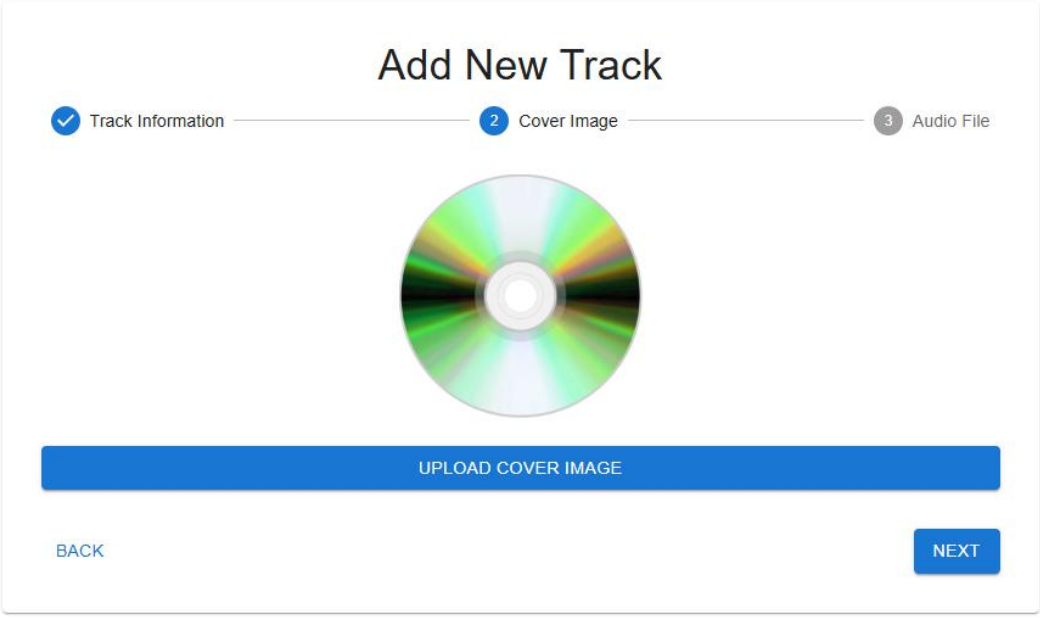

Рисунок 3.5 – Другий етап створення файлу

Після завантаження зазвичай з’являється попередній перегляд обкладинки, щоб впевнитися, що файл завантажено правильно. Очікується, що зображення будуть у форматах JPG, PNG або WebP, з обмеженням по розміру, наприклад, до 5 мегабайт. Користувач також може повернутися до попереднього кроку або перейти до наступного, тобто завантаження аудіофайлу. Додатково можуть бути рекомендації щодо бажаного розміру (наприклад, 600×600 пікселів), а також опціональна можливість автоматичної генерації обкладинки, якщо користувач не додав власну.

На третьому етапі завантажується сам музичний трек. Це фінальний і найважливіший момент, коли користувач обирає аудіофайл у форматі MP3, WAV або подібному.

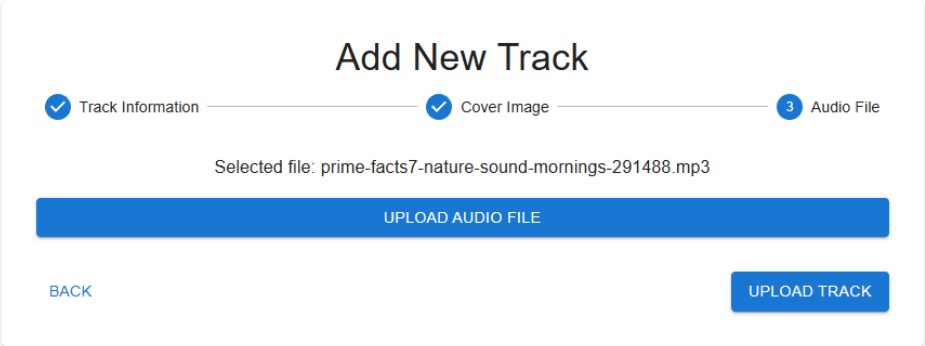

Рисунок 3.6 – Третій етап створення файлу

Система перевіряє, чи дійсно файл є аудіофайлом, чи не перевищує допустимий розмір, і чи не є порожнім. Якщо реалізовано розширені функції, можна також прослухати трек або зчитати ID3-теги для автоматичного заповнення полів перед публікацією. Коли все готово, користувач натискає кнопку завершення ("Submit" або "Finish"), після чого трек зберігається на сервері, і користувача перенаправляють до бібліотеки або на сторінку створеного треку. Цей процес завершує публікацію, роблячи трек доступним для інших користувачів платформи.

Рисунок 3.7 – Алгоритм створення файлу

Висновки до розділу 3

У цьому розділі було представлено реалізований програмний продукт відповідно до раніше визначених вимог і обраного технологічного стеку. Продемонстровано загальну архітектуру системи, її основні компоненти та логіку взаємодії між клієнтською та серверною частинами.

Описано ключові функціональні можливості проєкту, інтерфейс користувача та особливості реалізації основних сценаріїв використання. Завдяки застосуванню сучасних технологій, NestJS для бекенду, Next.js для фронтенду, Prisma для взаємодії з базою даних та Docker для контейнеризації, вдалося створити гнучку, масштабовану та зручну у використанні систему.

Проведений аналіз реалізованого функціоналу показав, що створене рішення відповідає поставленим цілям та завданням, а також є зручним для подальшогорозширення або адаптації під конкретні потреби користувачів.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1 Головна сторінка

Головна сторінка вебплатформи відіграє важливу роль у тому, як користувач взаємодіє з системою. Вона повинна бути не тільки технічно правильною, але й візуально привабливою. Ця сторінка має привертати увагу користувачів, показувати переваги платформи та пропонувати простий у використанні інтерфейс для доступу до основних функцій.

Основна мета створення цієї сторінки - зробити зручну і адаптивну вітрину, яка відображає суть сервісу, що допомагає створювати та слухати музику. Сторінка зроблена у вигляді функціонального React-компонента на базі Next.js. Код використовує JSX та компоненти з бібліотеки MUI, такі как Container, Typography, Button, Grid і Card, що полегшує масштабування та оформлення інтерфейсу. Фони, кольори, відступи, шрифти та інші стилістичні елементи підібрані в дусі мінімалізму, контрасту та функціональності, які стали основою дизайну проєкту.

Програмна реалізація на фронтенд частині:

```
    import React from 'react';
    import {
      Box,
      Container,
      Typography,
      Button,
      Grid,
      Card,
      CardContent,
      useTheme
    } from '@mui/material';
    import { styled } from '@mui/material/styles';
    import HeadphonesIcon from'@mui/icons-
material/Headphones';
    import LibraryMusicIcon from '@mui/icons-
material/LibraryMusic';

    import AlbumIcon from '@mui/icons-material/Album';
```

```
    import TrackList from
'@/component/TrackListComponent';

    import tracks from '@/data/tracks';
    import MainLayout from '@/layouts/MainLayout';

    const HeroSection = styled(Box)(({ theme }) => ({
      position: 'relative',
      height: '65vh',
      minHeight: 400,
      maxHeight: 700,
      display: 'flex',
      alignItems: 'center',
      backgroundImage: 'linear-gradient(180deg,
rgba(29,185,84,0.6) 0%, rgba(25,20,20,0.9) 100%),
url(https://images.pexels.com/photos/1105666/pexels-photo-
1105666.jpeg)',
      backgroundSize: 'cover',
      backgroundPosition: 'center',
      color: theme.palette.common.white,
    }));

    const FeatureCard = styled(Card)(({ theme }) => ({
      height: '100%',
      backgroundColor: theme.palette.background.paper,
      transition: 'transform 0.3s ease',
      '&:hover': {
        transform: 'translateY(-8px)',
      },
    }));

    const StatsWrapper = styled(Box)(({ theme }) => ({
      background: 'linear-gradient(to right, #3a1c71,
#d76d77, #ffaf7b)',
      padding: theme.spacing(8, 0),
      marginTop: theme.spacing(6),
      marginBottom: theme.spacing(6),
    }));

    function HomePage() {
      const theme = useTheme();
      const featuredTracks = tracks.filter((track: any) =>
track.popularity > 92).slice(0, 6);
```

```
        return (
          <MainLayout>
              <Box>
                  <HeroSection>
                      <Container maxWidth="lg">
                          <Grid container spacing={2}>
                              <Grid item xs={12} md={7}>
                                  <Typography
                                      variant="h1"
                                      component="h1"
                                      sx={{
                                      fontWeight: 900,
                                      fontSize: { xs:
'2.5rem', sm: '3.5rem', md: '4rem' },
                                      mb: 2
                                      }}
                                  >
                                      Listen Without Limits
                                  </Typography>
                                  <Typography
                                      variant="h5"
                                      sx={{
                                      mb: 4,
                                      opacity: 0.9,
                                      maxWidth: 600
                                      }}
                                  >
                                      Stream millions of
songs and podcasts on your device.
                                  </Typography>
                                  <Button
                                      variant="contained"
                                      size="large"
                                      sx={{
                                      px: 4,
                                      py: 1.5,
                                      fontSize: '1rem'
                                      }}
                                  >
                                      Get Started
                                  </Button>
                              </Grid>
                          </Grid>
```

```
                    </Container>
                </HeroSection>

                <Container maxWidth="lg" sx={{ mt: -6,
position: 'relative', zIndex: 1 }}>
                    <Grid container spacing={3}>
                        <Grid item xs={12} md={4}>
                            <FeatureCard>
                                <CardContent sx={{ p: 4,
textAlign: 'center' }}>
                                    <HeadphonesIcon sx={{
fontSize: 60, color: theme.palette.primary.main, mb: 2 }}
/>
                                    <Typography
variant="h5" component="h2" gutterBottom fontWeight={600}>
                                        Ad-Free Music
                                    </Typography>
                                    <Typography
variant="body1" color="text.secondary">
                                        Enjoy
uninterrupted music streaming with no advertisements,
allowing you to fully immerse in your favorite tracks.
                                    </Typography>
                                </CardContent>
                            </FeatureCard>
                        </Grid>

                        <Grid item xs={12} md={4}>
                            <FeatureCard>
                                <CardContent sx={{ p: 4,
textAlign: 'center' }}>
                                    <LibraryMusicIcon
sx={{ fontSize: 60, color: theme.palette.primary.main, mb:
2 }} />
                                    <Typography
variant="h5" component="h2" gutterBottom fontWeight={600}>
                                        Curated Playlists
                                    </Typography>
                                    <Typography
variant="body1" color="text.secondary">
                                        Discover new music
with our expertly curated playlists designed to match
    every mood, activity, and genre preference.
```

```
                                    </Typography>
                                </CardContent>
                            </FeatureCard>
                        </Grid>

                        <Grid item xs={12} md={4}>
                            <FeatureCard>
                                <CardContent sx={{ p: 4,
textAlign: 'center' }}>
                                    <AlbumIcon sx={{
fontSize: 60, color: theme.palette.primary.main, mb: 2 }}
/>
                                    <Typography
variant="h5" component="h2" gutterBottom fontWeight={600}>
                                        High Quality Audio
                                    </Typography>
                                    <Typography
variant="body1" color="text.secondary">
                                        Experience crystal
clear sound quality with our premium audio streaming
technology that brings out every detail.
                                    </Typography>
                                </CardContent>
                            </FeatureCard>
                        </Grid>
                    </Grid>
                </Container>

                <Box sx={{ py: 8 }}>
                    <TrackList tracks={featuredTracks}
title="Featured Tracks" />
                </Box>

                <StatsWrapper>
                    <Container maxWidth="lg">
                        <Grid container spacing={4}
justifyContent="center" textAlign="center">
                            <Grid item xs={12} sm={4}>
                                <Typography variant="h2"
fontWeight={900} sx={{ mb: 1 }}>
                                    15M+
                                </Typography>
                                <Typography variant="h6">
```

```
                                        Active Users
                                    </Typography>
                                </Grid>
                                <Grid item xs={12} sm={4}>
                                    <Typography variant="h2"
fontWeight={900} sx={{ mb: 1 }}>
                                        50M+
                                    </Typography>
                                    <Typography variant="h6">
                                        Tracks Available
                                    </Typography>
                                </Grid>
                                <Grid item xs={12} sm={4}>
                                    <Typography variant="h2"
fontWeight={900} sx={{ mb: 1 }}>
                                        100+
                                    </Typography>
                                    <Typography variant="h6">
                                        Countries Served
                                    </Typography>
                                </Grid>
                            </Grid>
                        </Container>
                    </StatsWrapper>

                    <Container maxWidth="lg" sx={{ py: 8 }}>
                        <Grid container spacing={6}
alignItems="center">
                            <Grid item xs={12} md={6}>
                                <Typography variant="h3"
component="h2" gutterBottom fontWeight={700}>
                                    Take Your Music Everywhere
                                </Typography>
                                <Typography variant="body1"
paragraph sx={{ mb: 3 }}>
                                    Listen on your phone,
tablet, desktop, and more. Our platform is designed to
seamlessly follow you wherever you go, ensuring your
favorite tunes are always just a click away.
                                </Typography>
                                <Typography variant="body1"
paragraph>
```

```
                                    With offline listening,
you can download your music and take it with you even when
there's no internet connection. Perfect for travel,
commutes, or anywhere your day takes you.
                                </Typography>
                                <Button
                                    variant="outlined"
                                    color="primary"
                                    size="large"
                                    sx={{ mt: 2 }}
                                >
                                    Learn More
                                </Button>
                            </Grid>
                            <Grid item xs={12} md={6}>
                                <Box
                                    component="img"

src="https://images.pexels.com/photos/1337753/
    pexels-photo-1337753.jpeg"
                                    alt="Person listening to
music"
                                    sx={{
                                        width: '100%',
                                        height: 'auto',
                                        borderRadius: 2,
                                        boxShadow: 3,
                                    }}
                                />
                            </Grid>
                        </Grid>
                    </Container>
                </Box>
            </MainLayout>
        );
    }

    export default HomePage;
```

4.2 Сторінка логіну та реєстрації

Авторизація – це важлива частина нашої платформи. Вона дозволяє користувачам безпечно отримувати доступ до своїх особистих функцій, як-от створення, збереження та редагування музичних треків. Процес авторизації легко знайти через посилання в заголовку сайту. Коли натискаєш на іконку користувача, потрапляєш на сторінку входу, де треба ввести логін (електронну пошту) та пароль.

З технічної сторони, авторизація реалізована як компонент React, який використовує хуки для збереження введених даних. Коли натискаєш кнопку Login, запускається асинхронна функція, яка відправляє ці дані на сервер, створений за допомогою NestJS. Сервер перевіряє, чи вірні облікові дані. Якщо все ок, він повертає токен доступу (JWT), який зберігається в браузері користувача - зазвичай у localStorage або як httpOnly cookie, залежно від налаштувань безпеки.

Якщо пароль або електронна пошта неправильні, інтерфейс покаже помилку, але без перезавантаження сторінки. Це забезпечує швидку і зручну роботу з формою, без зайвих переходів чи оновлень.

Сторінка логіну та реєстрації на фронтенд частині:

```
    'use client';

    import React, { useState } from 'react';
    import { Box, Tabs, Tab, TextField, Button, Card,
CardContent, Typography } from '@mui/material';

    function Auth() {
      const [tab, setTab] = useState(0);

      const handleChange = (event: React.SyntheticEvent,
newValue: number) => {
        setTab(newValue);
      };

      return (
        <Box
          sx={{
            width: 400,
```

```
            margin: '100px auto',
          }}
        >
          <Card>
            <CardContent>
              <Tabs   value={tab}   onChange={handleChange}
variant="fullWidth" centered>
                <Tab label="Login" />
                <Tab label="Register" />
              </Tabs>

              {tab === 0 && (
                <Box   sx={{   mt:   3,   display:   'flex',
flexDirection: 'column', gap: 2 }}>
                  <TextField   label="Email"   type="email"
fullWidth />
                  <TextField              label="Password"
type="password" fullWidth />
                  <Button              variant="contained"
color="primary" fullWidth>
                    Login
                  </Button>
                </Box>
              )}

              {tab === 1 && (
                <Box   sx={{   mt:   3,   display:   'flex',
flexDirection: 'column', gap: 2 }}>
                  <TextField label="Name" fullWidth />
                  <TextField   label="Email"   type="email"
fullWidth />
                  <TextField              label="Password"
type="password" fullWidth />
                  <TextField    label="Confirm    Password"
type="password" fullWidth />
                  <Button              variant="contained"
color="primary" fullWidth>
                    Register
                  </Button>
                </Box>
              )}
            </CardContent>
          </Card>
```

```
            </Box>
        );
    }

    export default Auth;
```

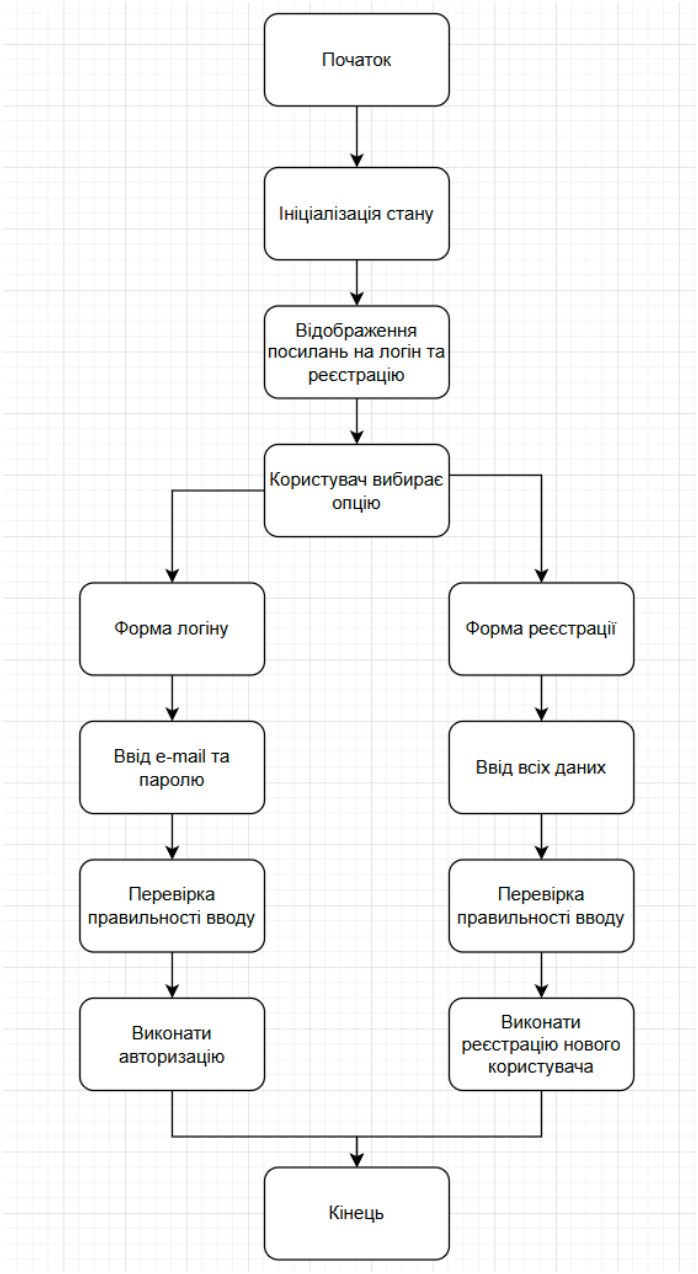

Рисунок 4.1 – Схема роботи логіну на реєстрації

4.3 Схема Prisma

Схема Prisma має три основні моделі: User, Track і Comment, які відповідають користувачам, музичним трекам і коментарям. Ось короткий опис кожної з них і як вони взаємодіють.

Модель User представляє зареєстрованого користувача. Вона має унікальний email, а також необов'язкове ім'я і хешований пароль. Це дозволяє користувачам авторизуватися в системі.

Модель Track описує музичний трек. Кожен трек має свій унікальний ідентифікатор, назву, ім'я виконавця, опис (за замовчуванням – “No description”), рейтинг популярності, посилання на обкладинку, аудіофайл, дату релізу, тривалість і жанр. До кожного трека можуть бути прив'язані кілька коментарів завдяки зв'язку один-до-багатьох з моделлю Comment.

Модель Comment містить унікальний ідентифікатор, ім'я користувача, текст коментаря та зовнішній ключ trackId, який показує, до якого треку цей коментар належить. Зв'язок зав'язується через поле track, що вказує на модель Track.

Дані зберігаються в PostgreSQL, а URL до бази даних написаний у змінній DATABASE_URL. Для роботи з базою даних використовують prisma-client-js.

Вигляд схеми у проекті:

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model Comment {
  id       Int    @id @default(autoincrement())
  username String
  text     String
  trackId  Int
```

```
  track     Track  @relation(fields: [trackId], references:
[id])
}

model Track {
  id          Int       @id @default(autoincrement())
  title       String
  artist      String
  description String    @default("No description")
  popularity  Int
  image       String
  audio       String
  releaseDate String
  duration    String
  genre       String
  comments    Comment[]
}

model User {
  id       Int     @id @default(autoincrement())
  email    String  @unique
  name     String?
  password String
}
```

4.4 Сторінка додавання аудіофайлу

Сторінка зроблена у вигляді багатоетапної форми, де кожен крок відповідає певним даним про трек:

– інформація про трек, назва, артист, жанр, опис, дата релізу;

– обкладинка, завантаження картинки;

– аудіофайл, завантаження звукового файлу.

Цей інтерфейс працює з Stepper, що дозволяє переходити між етапами, зберігаючи введені дані в useState.

Сторінка додавання файлу:

```
import React, { useState } from 'react';
import {
Box,
```

```
Container,
Typography,
Stepper,
Step,
StepLabel,
Button,
TextField,
Paper,
FormControl,
InputLabel,
Select,
MenuItem,
Stack,
} from '@mui/material';
import { styled } from '@mui/material/styles';
import MainLayout from '@/layouts/MainLayout';
import axios from 'axios';
import router from 'next/router';

const Input = styled('input')({
display: 'none',
});

const steps = ['Track Information', 'Cover Image', 'Audio
File'];

type FormDataType = {
title: string;
artist: string;
genre: string;
description: string;
releaseDate: string;
image: File | null;
audio: File | null;
};

function AddTrackPage() {
const [activeStep, setActiveStep] = useState(0);
const [formData, setFormData] = useState<FormDataType>({
title: '',
artist: '',
genre: '',
description: '',
```

```
releaseDate: '',
image: null,
audio: null,
});

const [coverImagePreview, setCoverImagePreview] =
useState('');
const [audioFileName, setAudioFileName] = useState('');

const handleNext = () => {
setActiveStep((prevStep) => prevStep + 1);
};

const handleBack = () => {
setActiveStep((prevStep) => prevStep - 1);
};

const handleInputChange = (
e: any
) => {
const { name, value } = e.target;

setFormData((prev) => ({
...prev,
[name as keyof FormDataType]: value,
}));
};


const handleImageUpload = (e: any) => {
const file = e.target.files[0];
if (file) {
setFormData((prev) => ({
...prev,
image: file,
}));
setCoverImagePreview(URL.createObjectURL(file));
}
};

const handleAudioUpload = (e: any) => {
const file = e.target.files[0];
if (file) {
```

```
setFormData((prev) => ({
...prev,
audio: file,
}));
setAudioFileName(file.name);
}
};

const handleSubmit = (e: any) => {
e.preventDefault();

const data = new FormData();

Object.keys(formData).forEach((key) => {
const value = formData[key as keyof FormDataType];
if (value instanceof File) {
data.append(key, value);
} else if (value !== null && value !== undefined) {
data.append(key, String(value));
}
});
axios.post('http://localhost:5000/tracks', data)
.then((resp) => {
console.log('Response:', resp);
router.push('/tracks');
})
.catch((e) => {
console.error('Error in submitting form:', e);
if (e.response) {
console.error('Error Response:', e.response);
}
});

};

const getStepContent = (step: any) => {
switch (step) {
case 0:
return (
<Stack spacing={3}>
<TextField
fullWidth
label="Title"
```

```
    name="title"
    value={formData.title}
    onChange={handleInputChange}
    required
    />
    <TextField
    fullWidth
    label="Artist"
    name="artist"
    value={formData.artist}
    onChange={handleInputChange}
    required
    />
    <FormControl fullWidth>
    <InputLabel>Genre</InputLabel>
    <Select
    name="genre"
    value={formData.genre}
    label="Genre"
    onChange={handleInputChange}
    >
    <MenuItem value="Pop">Pop</MenuItem>
    <MenuItem value="Rock">Rock</MenuItem>
    <MenuItem value="Hip Hop">Hip Hop</MenuItem>
    <MenuItem value="Electronic">Electronic</MenuItem>
    <MenuItem value="Classical">Classical</MenuItem>
    <MenuItem value="Jazz">Jazz</MenuItem>
    </Select>
    </FormControl>
    <TextField
    fullWidth
    label="Description"
    name="description"
    value={formData.description}
    onChange={handleInputChange}
    multiline
    rows={4}
    />
    <TextField
    fullWidth
    label="Release Date"
    name="releaseDate"
    type="date"
```

```
value={formData.releaseDate}
onChange={handleInputChange}
InputLabelProps={{ shrink: true }}
/>
</Stack>
);

case 1:
return (
<Box sx={{ textAlign: 'center' }}>
{coverImagePreview && (
<Box
component="img"
src={coverImagePreview}
alt="Cover preview"
sx={{
width: 200,
height: 200,
objectFit: 'cover',
borderRadius: 1,
mb: 2,
}}
/>
)}
<label htmlFor="cover-image">
<Input
accept="image/*"
id="cover-image"
type="file"
onChange={handleImageUpload}
/>
<Button variant="contained" component="span" fullWidth>
Upload Cover Image
</Button>
</label>
</Box>
);

case 2:
return (
<Box sx={{ textAlign: 'center' }}>
{audioFileName && (
<Typography variant="body1" sx={{ mb: 2 }}>
```

```
Selected file: {audioFileName}
</Typography>
)}
<label htmlFor="audio-file">
<Input
accept="audio/*"
id="audio-file"
type="file"
onChange={handleAudioUpload}
/>
<Button variant="contained" component="span" fullWidth>
Upload Audio File
</Button>
</label>
</Box>
);

default:
return 'Unknown step';
}
};

return (
<MainLayout>
<Container maxWidth="md" sx={{ py: 12 }}>
<Paper sx={{ p: 4 }}>
<Typography variant="h4" component="h1" gutterBottom
align="center">
Add New Track
</Typography>

<Stepper activeStep={activeStep} sx={{ mb: 4 }}>
{steps.map((label) => (
<Step key={label}>
<StepLabel>{label}</StepLabel>
</Step>
))}
</Stepper>

<form onSubmit={handleSubmit}>
{getStepContent(activeStep)}
```

```
<Box sx={{ display: 'flex', justifyContent: 'space-
between', mt: 4 }}>
<Button
disabled={activeStep === 0}
onClick={handleBack}
>
Back
</Button>
<Box>
{activeStep === steps.length - 1 ? (
<Button
variant="contained"
type="submit"
disabled={!formData.title || !formData.artist ||
!formData.image || !formData.audio}
>
Upload Track
</Button>
) : (
<Button
variant="contained"
onClick={handleNext}
disabled={activeStep === 0 && (!formData.title ||
!formData.artist)}
>
Next
</Button>
)}
</Box>
</Box>
</form>
</Paper>
</Container>
</MainLayout>
);
}

export default AddTrackPage;
```

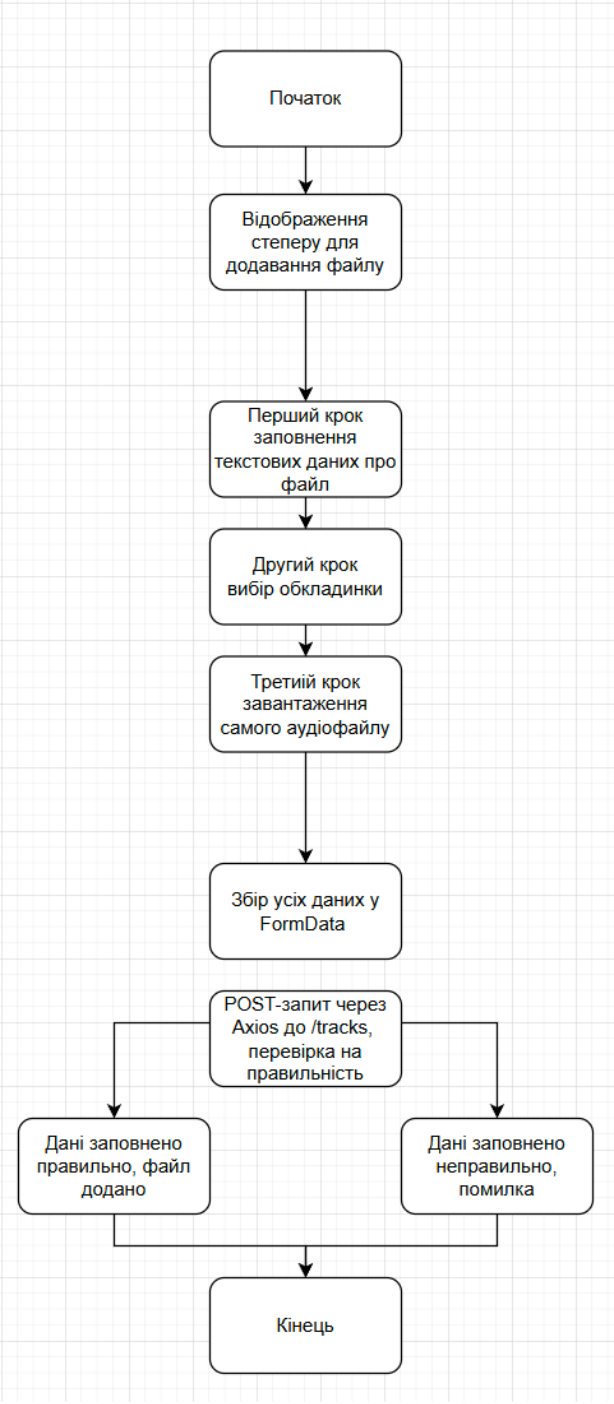

Рисунок 4.2 – Схема додавання файлу

4.5 Сторінка з аудіофайлами

TrackList – це зручний елемент, який може приймати:

– список треків (tracks);

– необов'язковий заголовок (title).

Він показує всі треки у вигляді сітки, де кожен трек представлений окремою карткою, яку створює інший компонент – TrackCard.

```
import React from 'react';
import {
  Grid,
  Box,
  Container,
  Typography
} from '@mui/material';
import TrackCard from './TrackCard';
function TrackList({ tracks, title }: any) {
  return (
    <Box sx={{ py: 4 }}>
      <Container maxWidth="lg">
        {title && (
          <Typography
            variant="h4"
            component="h2"
            gutterBottom
            sx={{
              fontWeight: 700,
              mb: 3
            }}
          >
            {title}
          </Typography>
        )}
        <Grid container spacing={3}>
          {tracks.map((track: any) => (
            <Grid item key={track.id} xs={6} sm={4} md={3}
lg={2.4}>
              <TrackCard track={track} />
            </Grid>))}
        </Grid>
      </Container>
    </Box>
  );
} export default TrackList;
```

Висновки до розділу 4

У цьому розділі було детально розглянуто основні програмні компоненти реалізованого застосунку. Надано фрагменти ключового коду, які відповідають за основну бізнес-логіку, взаємодію з базою даних, обробку запитів користувача та відображення інформації на клієнтській стороні.

Для кращого розуміння структури та внутрішніх процесів системи було використано блок-схеми, які відображають послідовність виконання основних алгоритмів і взаємодію між модулями. Це дозволило візуально представити логіку роботи застосунку, зокрема: обробку запитів API, автентифікацію користувачів, обробку медіаданих та взаємодію з базою даних через Prisma.

Проведений аналіз підтвердив, що структура коду є логічною, модульною та відповідає принципам чистої архітектури. Код легко розширюється, тестується та підтримується, що є важливою умовою для подальшого розвитку проєкту.

ВИСНОВКИ

У процесі створення веб-застосунку для зберігання та редагування музичних файлів, який поєднує в собі функції платформ, подібних до Spotify і FL Studio, було досягнуто значного прогресу як у бекенді, так і у фронтенді. Завдяки використанню технологій Next.js і NestJS, нам вдалося забезпечити ефективну роботу як на серверній, так і на клієнтській частинах застосунку.

Однією з ключових задач було створення зручного інтерфейсу для завантаження, зберігання, редагування та обробки музичних файлів у реальному часі. Використання React та Next.js дозволило створити інтуїтивно зрозумілий інтерфейс, що робить взаємодію користувачів з контентом максимально комфортною.

Також важливою частиною стала безпека та захист даних, зокрема забезпечення збереження інтелектуальної власності та запобігання несанкціонованому доступу до музичних файлів. Враховуючи значущість цих аспектів, були впроваджені відповідні заходи для захисту інформації.

Під час реалізації проекту ми також врахували питання масштабування, щоб забезпечити доступ до великих обсягів музичних файлів навіть під час високих навантажень. Завдяки NestJS, було оптимізовано серверні запити та взаємодію з базою даних і хмарними сервісами зберігання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1) Сайт вікіпедії URL: https://www.wikipedia.com.ua/
2) Документація NestJS URL: https://docs.nestjs.com
3) Документація NextJS URL: https://nextjs.org/docs
4) Документація Docker URL: https://docs.docker.com
5) Офіційна документація JavaScript URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript
6) Документація TypeScript URL: https://www.typescriptlang.org/docs
7) Документація ReactJS URL: https://react.dev
8) Node.js URL: https://nodejs.org/en/docs
9) Express.js URL: https://expressjs.com
10) PostgreSQL URL: https://www.postgresql.org/docs
11) Visual Studio Code URL: https://code.visualstudio.com
12) MySQL URL: https://dev.mysql.com/doc
13) Prisma ORM URL: https://www.prisma.io/docs
14) Redis URL: https://redis.io/docs
15) Webpack URL: https://webpack.js.org
16) Babel URL: https://babeljs.io/docs
17) Git URL: https://git-scm.com/doc
18) GitHub Docs URL: https://docs.github.com
19) CI/CD з GitHub Actions URL: https://docs.github.com/en/actions
20) Jest (тестування) URL: https://jestjs.io/docs
21) Cypress (E2E-тести) URL: https://docs.cypress.io
22) REST API Design URL: https://restfulapi.net
23) GraphQL URL: https://graphql.org/learn
24) Swagger (OpenAPI) URL: https://swagger.io/docs
25) Visual Studio Code Docs URL: https://code.visualstudio.com/docs
26) Stack Overflow URL: https://stackoverflow.com

27) FreeCodeCamp URL: https://www.freecodecamp.org

28) W3Schools (базові вебтехнології) URL: https://www.w3schools.com

29) Книга Robert C. Martin – "Clean Code: A Handbook of Agile Software Craftsmanship"

30) Книга Steve McConnell – "Code Complete" (2nd Edition)

Додаток А
Код застосунку

Файл user.servise.ts:

```
import { ConflictException, Injectable } from
'@nestjs/common';
import { PrismaService } from
'src/prisma/prisma.service';
import { CreateUserDto } from './dto/user.dto';

@Injectable()
export class UserService {
    constructor(private prisma: PrismaService){}

    async create(userDto: CreateUserDto){
        const user = await
this.prisma.user.findUnique({
            where: {
                email: userDto.email,
            }
        });

        if (user) throw new ConflictException('email
duplicated');

        const newUser = await
this.prisma.user.create({
            data: {
                ...userDto
            }
        })
    }
}
```

Файл file.service.ts:

```
import * as fs from 'fs';
import * as path from 'path';
import * as uuid from 'uuid';
import * as mm from 'music-metadata';
import { HttpException, HttpStatus, Injectable } from
'@nestjs/common';
```

```
export enum FileType{
    AUDIO = 'audio',
    IMAGE = 'image'
}

@Injectable()
export class FileService {
  async createFile(type: FileType, file): Promise<{
filePath: string, duration?: string }> {
    try {
      const fileExtension =
file.originalname.split('.').pop();
      const fileName = uuid.v4() + '.' + fileExtension;

      const filePath = path.resolve(__dirname, '..',
'static', type);

      if (!fs.existsSync(filePath)) {
        await fs.promises.mkdir(filePath, { recursive:
true });
      }

      await
fs.promises.writeFile(path.resolve(filePath, fileName),
file.buffer);

      if(type === FileType.AUDIO){
        const fullAudioPath = path.resolve(filePath,
fileName);
        const metadata = await
mm.parseFile(fullAudioPath);
        const durationInSeconds =
metadata.format.duration ?? 0;
        const formattedDuration =
this.formatDuration(durationInSeconds);

        return { filePath: type + '/' + fileName,
duration: formattedDuration };
      }

      return { filePath: type + '/' + fileName }
    } catch (e) {
```

```
            throw new HttpException(e.message,
    HttpStatus.INTERNAL_SERVER_ERROR);
          }
        }

        private formatDuration(seconds: number): string {
          const minutes = Math.floor(seconds / 60);
          const remainingSeconds = Math.floor(seconds % 60);
          return
    `${minutes}:${remainingSeconds.toString().padStart(2,
    '0')}`;
        }

        removeFile(fileName: string) {
        }
    }
```

Файл track.service.ts:

```
import { Injectable } from "@nestjs/common";
import { FileService, FileType } from
"src/file/file.service";
import { CreateCommentDto } from "./dto/create-comment-
dto";
import { CreateTrackDto } from "./dto/create-track-dto";
import { PrismaService } from "src/prisma/prisma.service";

@Injectable()
export class TrackService {
  constructor(
    private readonly prisma: PrismaService,
    private readonly fileService: FileService
  ) {}

  public async create(dto: CreateTrackDto, image:
Express.Multer.File, audio: Express.Multer.File) {
    const saveAudio = await
this.fileService.createFile(FileType.AUDIO, audio);
    const saveImage = await
this.fileService.createFile(FileType.IMAGE, image);
```

```
    const track = await this.prisma.track.create({
      data: {
        ...dto,
        popularity: 0,
        audio: saveAudio.filePath,
        image: saveImage.filePath,
        duration: saveAudio.duration
      },
    });

    return track;
  }

  public async getAll(count = 10, offset = 0) {
    const tracks = await this.prisma.track.findMany({
      skip: Number(offset),
      take: Number(count),
    });
    return tracks;
  }

  public async getOne(id: number) {
    const track = await this.prisma.track.findUnique({
      where: { id: Number(id) },
      include: {
        comments: true,
      },
    });
    return track;
  }

  public async delete(id: number) {
    const track = await this.prisma.track.delete({
      where: { id },
    });
    return track.id;
  }

  public async addComment(dto: CreateCommentDto) {
    const comment = await this.prisma.comment.create({
```

```
      data: {
        username: dto.username,
        text: dto.text,
        track: {
          connect: { id: dto.trackId },
        },
      },
    });

    return comment;
  }

  public async listen(id: number) {
    await this.prisma.track.update({
      where: { id },
      data: {
        popularity: {
          increment: 1,
        },
      },
    });
  }

  public async search(query: string) {

    const tracks = await this.prisma.track.findMany({
      where: {
        title: {
          contains: query,
          mode: 'insensitive',
        },
      },
    });

    return tracks;
  }
}
```

Файл AudioPlayer.tsx:

```
import React, { useEffect } from 'react';
import { useActions } from '@/hooks/useAction';
import { useTypedSelector } from '@/hooks/useTypeSelector';
import {
  Box,
  IconButton,
  Paper,
  Slider,
  Stack,
  styled,
  Typography,
} from '@mui/material';
import SkipPreviousIcon from '@mui/icons-material/SkipPrevious';
import SkipNextIcon from '@mui/icons-material/SkipNext';
import VolumeUpIcon from '@mui/icons-material/VolumeUp';
import PlayCircleIcon from '@mui/icons-material/PlayCircle';
import PauseCircleIcon from '@mui/icons-material/PauseCircle';
import CloseIcon from '@mui/icons-material/Close';

const PlayerWrapper = styled(Paper)(({ theme }) => ({
  position: 'fixed',
  bottom: 0,
  left: 0,
  right: 0,
  padding: theme.spacing(1, 2),
  backgroundColor: '#181818',
  borderTop: '1px solid rgba(255, 255, 255, 0.1)',
  display: 'flex',
  alignItems: 'center',
  zIndex: 1000,
}));

const TrackInfo = styled(Box)({
  display: 'flex',
  alignItems: 'center',
  maxWidth: '30%',
});

const Controls = styled(Box)({
```

```
  display: 'flex',
  flexDirection: 'column',
  alignItems: 'center',
  justifyContent: 'center',
  flex: 1,
});

const VolumeControl = styled(Box)({
  display: 'flex',
  alignItems: 'center',
  maxWidth: '30%',
  justifyContent: 'flex-end',
});

const TrackImage = styled('img')({
  width: 56,
  height: 56,
  objectFit: 'cover',
  marginRight: 12,
  borderRadius: 4,
});

let audio: HTMLAudioElement | null = null;

export function AudioPlayer() {
    const { pause, volume, active, duration, currentTime }
= useTypedSelector(
        (state) => state.player
    );
    const {
        pauseTrack,
        playTrack,
        setVolume,
        setCurrentTime,
        setDuration,
        setActiveTrack
    } = useActions();

    useEffect(() => {
        if (!audio) {
            audio = new Audio();
        }
```

```
        if (active) {
            if (audio.src !== 'http://localhost:5000/' +
active.audio) {
                audio.src = 'http://localhost:5000/' +
active.audio;
            }

            audio.onloadedmetadata = () => {
                setDuration(Math.ceil(audio!.duration));
            };
            audio.ontimeupdate = () => {

setCurrentTime(Math.ceil(audio!.currentTime));
            };

            audio.volume = volume / 100;

            if (!pause) {
                audio.play();
            }
        }
    }, [active]);

    useEffect(() => {
        if (!audio) return;

        if (pause) {
            audio.pause();
        } else {
            audio.play();
        }
    }, [pause]);

    const play = () => {
        if (pause) {
            playTrack();
            audio!.play();
        } else {
            pauseTrack();
            audio!.pause();
        }
    };
```

```
    const closePlayer = () => {
        audio?.pause();
        setActiveTrack(null);
    };

    const changeVolume = (_: Event, value: number | number[])
=> {
        if (!audio) return;
        const newVolume = typeof value === 'number' ? value
: value[0];
        audio.volume = newVolume / 100;
        setVolume(newVolume);
    };

    const changeCurrentTime = (_: Event, value: number |
number[]) => {
        if (!audio) return;
        const newTime = typeof value === 'number' ? value :
value[0];
        audio.currentTime = newTime;
        setCurrentTime(newTime);
    };

    const formatTime = (seconds: number) =>
        new Date(seconds * 1000).toISOString().substr(14,
5);

    if (!active) return null;

    return (
        <PlayerWrapper elevation={3}>
            <IconButton
                onClick={closePlayer}
                sx={{
                    position: 'absolute',
                    top: 8,
                    right: 8,
                    color: '#fff'
                }}
                >
                <CloseIcon />
            </IconButton>
            <TrackInfo>
```

```
                <TrackImage
                    src={'http://localhost:5000/'          +
active.image}
                    alt={active.title}
                />
                <Box>
                    <Typography  variant="subtitle2"  noWrap
sx={{ color: '#fff' }}>
                        {active.title}
                    </Typography>
                    <Typography   variant="caption"   noWrap
sx={{ color: '#ccc' }}>
                        {active.artist}
                    </Typography>
                </Box>
            </TrackInfo>

            <Controls>
                <Stack      direction="row"      spacing={1}
alignItems="center" sx={{ mb: 1 }}>
                    <IconButton sx={{ color: '#fff' }}>
                        <SkipPreviousIcon />
                    </IconButton>
                    <IconButton sx={{ color: 'primary.main'
}} onClick={play}>
                        {pause ? (
                            <PlayCircleIcon sx={{ fontSize:
40 }} />
                        ) : (
                            <PauseCircleIcon          sx={{
fontSize: 40 }} />
                        )
                    }
                    </IconButton>
                    <IconButton sx={{ color: '#fff' }}>
                        <SkipNextIcon />
                    </IconButton>
                </Stack>

                <Stack      direction="row"      spacing={2}
alignItems="center" sx={{ width: '100%', maxWidth: 500 }}>
                    <Typography   variant="caption"   sx={{
color: '#fff' }}>
```

```
                        {formatTime(currentTime)}
                    </Typography>
                    <Slider
                        size="small"
                        value={currentTime}
                        max={duration}
                        onChange={changeCurrentTime}
                        sx={{ color: 'primary.main' }}
                    />
                    <Typography   variant="caption"   sx={{
color: '#fff' }}>
                        {formatTime(duration)}
                    </Typography>
                </Stack>
            </Controls>

            <VolumeControl>
                <VolumeUpIcon sx={{ color: '#fff' }} />
                <Slider
                    size="small"
                    value={volume}
                    onChange={changeVolume}
                    sx={{   width:   100,   ml:   1,   color:
'primary.main' }}
                />
            </VolumeControl>
        </PlayerWrapper>
    );
}
```

Файл MainLayout.tsx:

```
import Footer from "@/component/layer/Footer";
import Header from "@/component/layer/Header";
import Navbar from "@/component/layer/Navbar";
import { Box } from "@mui/material";
import React, { ReactNode, useState } from "react";

interface ContainerProps {
    children?: ReactNode;
}

const MainLayout: React.FC<ContainerProps> = ({ children })
=> {
```

```
    const [navOpen, setNavOpen] = useState(false);

    const toggleNav = () => {
        setNavOpen(!navOpen);
    };

    return (
        <Box sx={{ display: 'flex', flexDirection: 'column',
minHeight: '100vh' }}>
            <Header toggleNav={toggleNav} />
            <Navbar    open={navOpen}    onClose={()    =>
setNavOpen(false)} />
            <Box component="main" sx={{ flexGrow: 1 }}>
                {children}
            </Box>
            <Footer />
        </Box>
    );
};

export default MainLayout;
```