МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

НН/факультет Інформаційних технологій
Кафедра Інформатики і прикладного програмного забезпечення
Спеціальність Інженерія програмного забезпечення
Форма навчання Денна

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Фігури Дениса Сергійовича
(прізвище, ім'я, по батькові здобувача)

на тему «Розробка програмного забезпечення музичного каталогу»
(повна назва теми)

за матеріалами праць провідних спеціалістів з розробки ПЗ та проектування БД
(повна назва бази дослідження)

науковий керівник к.е.н. доц. ________ Ткаліченко С.В.
(наук. ступінь, вчене звання) *(підпис)* *(прізвище, ініціали)*

Робота допущена до захисту в ЕК
Протокол засідання кафедри
від 11.06.2025 р. № 12
Завідувач кафедри ________________
(підпис)

д.т.н., професор Зеленський О.С.
Наук. ступень, вчене звання *Ініціали, прізвище*

Кривий Ріг – 2025

ЗГОДА здобувача вищої освіти
Державного університету економіки і технологій про перевірку кваліфікаційної роботи на прояви академічного плагіату та розміщення в Репозитарії Університету

Я, Фігура Денис Сергійович, підтримую політику Державного університету економіки і технологій з академічної доброчесності і відкритого доступу.

Засвідчую, що кваліфікаційна бакалаврська робота «Розробка програмного забезпечення музичного каталогу» виконана самостійно та не містить академічного плагіату. Я не надавав і не одержував недозволену допомогу під час підготовки цієї роботи. Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Державного університету економіки і технологій ознайомлений. Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення норм академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

Також я поінформований, що відповідно до «Положення про Репозитарій (електронну базу даних) Державного університету економіки і технологій» зазначена робота буде розміщена в Електронному архіві Університету (Репозитарії ДУЕТ). З умовами такого розміщення ознайомлений.

Дата　　　　　　　　　　підпис　　　　　　　　　　ініціали, прізвище

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ/факультет	Інформаційних технологій
Кафедра	Інформатики і прикладного програмного забезпечення
Спеціальність	Інженерія програмного забезпечення
Форма навчання	Денна

«ЗАТВЕРДЖУЮ»
Завідувач кафедри ____________ Зеленський О.С.
(підпис) (Прізвище, ініціали)
« 11 » червня 2025 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ

1. Тема роботи «Розробка програмного забезпечення музичного каталогу»

Керівник роботи к.е.н., доц. Такліченко С.В.
затверджені наказом закладу вищої освіти від «04» квітня 2025 р. № 222-ст

2. Строк подання здобувачем роботи до «09» червня 2025 р.

3. Зміст кваліфікаційної роботи, об’єкт, предмет та мета дослідження:

Розділ 1. Постановка задачі

Розділ 2. Розробка алгоритму розв’язання задачі

Розділ 3. Організація інформаційного забезпечення

Розділ 4. Розробка програмного забезпечення

Об’єкт дослідження: музичний каталог

Предмет дослідження: алгоритм каталогу

Мета кваліфікаційної роботи: *розробка програмного забезпечення музичного каталогу*

5. Дата видачі завдання «04» квітня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів МДР	Строк виконання етапів роботи	Відмітка керівника про виконання етапів (дата, підпис)
1	Підготовка розділу 1	04.04.2025-13.04.2025	
2	Підготовка розділу 2	14.04.2025-26.04.2025	
3.	Підготовка розділу 3	27.04.2025-15.05.2025	
4	Підготовка розділу 4	16.05.2025-08.06.2025	
5	Реєстрація завершеної кваліфікаційної роботи	09.06.2025	Реєстраційний №___ «09»червня 2025 р.
6	Отримання відгуку від наукового керівника	10.06.2025	
7	Подання кваліфікаційної роботи на перегляд завідувачу кафедри	11.06.2025	
8	Отримання зовнішньої рецензії	12.06.2025	
9	Попередній захист кваліфікаційної роботи на кафедрі	13.06.2025	
10	Підготовка до захисту в ЕК	16.06.2025-21.06.2025	

Завдання підготував науковий керівник ______________ __Ткаліченко С.В.___

(підпис) (прізвище та ініціали)

Завдання одержав ______________ ___Фігура Д.С.__

_

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

на кваліфікаційну бакалаврську роботу

«Розробка програмного забезпечення музичного каталогу»

Фігури Дениса Сергійовича

Кваліфікаційна бакалаврська робота на здобуття освітньо-кваліфікаційного рівня бакалавра зі спеціальності 121 «Інженерія програмного забезпечення» – Державний університет економіки і технологій – Кривий Ріг, 2025.

У бакалаврській дипломній роботі розроблено веб-застосунок для збереження та впорядкування персонального музичного каталогу з можливістю генерації рекомендацій на основі індивідуальних вподобань користувача. Клієнтська частина реалізована з використанням React, серверна – на ASP.NET Core. Для обміну даними використано REST API, а зберігання інформації забезпечено через MS SQL Server.

Ключові слова: КАТАЛОГ, РЕКОМЕНДАЦІЇ, REST API, ASP.NET CJRE, REACT, БАЗА ДАНИХ.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API	Інтерфейс прикладного програмування, набір засобів для взаємодії між програмними компонентами.
CRUD	базові операції обробки даних у базі.
HTTP	Протокол передачі гіпертексту.
JWT	Формат для передачі даних з перевіркою автентичності
ASP.NET Core	Кросплатформенний фреймворк для створення веб-застосунків від компанії Microsoft.
React	JavaScript-бібліотека для створення користувацьких інтерфейсів, розроблена компанією Meta.

ЗМІСТ

ВСТУП 8

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ 9

1.1 Характеристика задачі 9

1.2 Огляд існуючих музичних платформ 11

1.3. Вимоги до каталогу 18

РОЗДІЛ 2. РОЗРОБКА АЛГОРИТМУ РОЗВ’ЯЗАННЯ ЗАДАЧІ 22

РОЗДІЛ 3. ОРГАНІЗАЦІЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ 27

3.1. Вибір системи управління бази даних 27

3.2. Процес міграції даних 29

3.3. Структура бази даних 37

РОЗДІЛ 4. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 45

4.1. Авторизація та реєстрація 45

4.2. Каталог та рекомендації музики 50

ВИСНОВКИ 56

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 58

ВСТУП

Музика давно перестала бути лише елементом дозвілля. Вона перетворилася на інформаційний простір, де користувачі прагнуть зручного доступу до своїх уподобань. У той час, як великі стрімінгові платформи зосереджуються на монетизації, помітний масив функціоналу залишається поза їхньою увагою — особисті каталоги для впорядкування власних музичних колекцій. Саме тут виникає ідея створення програмного забезпечення, яке дає змогу зберігати музику та має алгоритм рекомендацій на основі уподобань користувача.

Мета бакалаврської дипломної роботи розробити веб-застосунок для збереження та управління персональним музичним каталогом. Особливу увагу зосереджено на простоті інтерфейсу, стабільній взаємодії між клієнтською та серверною частинами та можливості масштабування без зміни загальної архітектури.

Завдання дослідження:

1. Побудувати фронтенд частину застосунку за допомогою React.
2. Реалізувати серверну логіку на основі ASP.NET Core.
3. Налагодити REST API для зв’язку між клієнтом і сервером.
4. Забезпечити базові CRUD-операції для музичних елементів.
5. Створити систему фільтрації, сортування та пошуку в каталозі.
6. Запровадити систему збереження даних через інтеграцію з базою.
7. Підготувати середовище CI/CD для стабільного розгортання.

Об’єкт дослідження: процес створення веб-застосунку з клієнтською частиною на JavaScript-фреймворку та серверною логікою на .NET-платформі.

Предмет дослідження: технічна реалізація веб-каталогу для музичних композицій, включно з побудовою архітектури, розробкою інтерфейсів, налаштуванням API та логікою взаємодії з базою даних.

РОЗДІЛ 1
ПОСТАНОВКА ЗАДАЧІ

1.1 Характеристика задачі

Слухати музику — це звичка, що супроводжує людину в різних ситуаціях: у транспорті, під час роботи, у моменти відпочинку, сну чи концентрації. Для цього необхідно зберігати улюблені композиції, впорядковувати їх, повертатися до них у потрібний момент. У світі, де доступ до аудіоконтенту практично необмежений, зростає потреба в інструменті, який дозволяє не шукати кожного разу заново, а мати під рукою свою власну музичну базу. Наразі існують платформи з великим обсягом функцій, але, на жаль, доступ до них обмежений — цей функціонал найчастіше виявляється привілеями преміум-підписок. Базові можливості зазвичай зводяться до стримінгу, без можливості повноцінно керувати своїм архівом чи налаштовувати відображення за власним сценарієм.

Мета — створити застосунок, який вирішує дуже локальну, але чітку задачу: дати користувачеві можливість зручно вести персональний музичний каталог. Це не черговий стрімінговий сервіс і не платформа з платною підпискою. Це — місце, де користувач створює свою бібліотеку вручну або імпортує з інших джерел. Він фіксує свої музичні вподобання: від конкретного треку до цілих альбомів, додає виконавців, жанри, описи, і має змогу швидко повернутись до будь-якого збереженого елементу.

З технічного боку проект складатиметься з двох взаємозалежних частин: клієнтської (frontend) та серверної (backend). Фронтенд буде побудовано на React — сучасному JavaScript-фреймворку, який дозволяє створювати гнучкий та динамічний інтерфейс. Серверна частина реалізована на ASP.NET Core — платформі, що добре працює з REST API.

Привілеї платформи будуть заключатися у таких можливостях, які надаватимуться користувачу:

1) пошук музики;
2) редагування раніше створені записи;
3) видалення записів, які більше не актуальні;
4) переглядання повного списку записів у зручній формі;
5) переглядання рекомендацій, створених на основі його попередніх вподобать.

Окремо буде проведений аналіз на методи підтримки структури даних, щоб уникнути хаотичного переліку, і створити зручну навігацію по категоріям різних видів. Записи будуть згруповані за альбомами, жанрами або виконавцями.

На сервері має бути реалізовано REST API з метою забезпечення обробки HTTP-запитів від клієнта. Це включає в себе ендпоінти для всіх CRUD-операцій (CRUD — create, read, update, delete). Запити надсилаються у форматі JSON, сервер опрацьовує їх і повертає результат у тому ж форматі. Таким чином забезпечується незалежність між частинами застосунку: фронтенд може змінюватися окремо від бекенду, і навпаки.

Ще одна технічна необхідність — це робота з базою даних. У даному випадку застосовуватиметься Entity Framework Core як ORM-рішення для взаємодії з базою. Це дозволить описувати структуру таблиць мовою C#, не прописуючи вручну SQL-запити. Модель даних включатиме таблиці для композицій, виконавців, жанрів, альбомів, зв’язки між ними.

З точки зору користувача, завдання звучить просто: дати йому зручний інтерфейс для збереження музики. Але за цим стоїть багато технічних аспектів. Треба побудувати алгоритм взаємодії між компонентами, передбачити обробку помилок, валідувати дані, побудувати карту сайту, щоб сітко структурувати навігацію, формувати запити, підготувати графічне відображення відповідей сервера в разі помилок. Завдання ускладнюється ще й тим, що система повинна працювати при великому навантаженні. В планах розробити платформу, на якій одночасно можуть слухати музику приблизно 1000 користувачів.

У рамках розробки музичного сервісу передбачено функціональність реєстрації користувачів. Тому буде розроблено збереження облікових записів за допомогою стандартної структури ASP.NET Identity. Кожен користувач має вказати адресу електронної пошти, ім’я користувача та пароль. Для всіх цих полів застосовуватиметься валідація на клієнтському та серверному рівнях. Електронна пошта має перевірятися на відповідність стандартному формату, ім’я користувача — на допустимість символів і унікальність у базі, а пароль — на мінімальну складність, включно з довжиною та використанням великих літер і цифр.

Користувачі мають змогу додавати музичні композиції до каталогу. Кожен запис включатиме набір атрибутів, які підлягають обов’язковій або умовній перевірці. Назва композиції та ім’я виконавця є обов’язковими полями і порожних значень бути не може. Композиція не може бути розташована анонімно. Для року випуску встановлюватиметься обмеження у вигляді допустимого діапазону, що забезпечує коректність історичних даних. Жанр обиратиметься зі списку або вводиться, з контролем довжини та символів. Поле опису композиції є необов’язковим, але в разі його заповнення застосовується обмеження за кількістю символів та заборона HTML-тегів.

Окрема частина взаємодії користувача з системою — це пошук музики. Введене значення очищуватиметься від зайвих пробілів, спеціальних символів і має переводиться до уніфікованого вигляду перед обробкою. Це розроблено з метою гарантувати відповідність запиту структурі даних та забезпечити коректне функціонування пошукового механізму.

1.2 Огляд існуючих музичних платформ

Однією з популярних платформ для прослуховування музики є сервіс YouTube Music (https://music.youtube.com/) — це потоковий музичний сервіс, розроблений компанією Google, який поєднує в собі можливості традиційного музичного програвача з доступом до відеоконтенту YouTube [9]. Платформа

орієнтована на широку аудиторію, яка шукає зручний спосіб прослуховування музики в поєднанні з відео. Сервіс надає доступ до мільйонів треків, кліпів, альбомів та персоналізованих рекомендацій. Платформа орієнтована як на мобільних користувачів, так і на десктопну версію, з підтримкою авторизації через Google-акаунт. Головна сторінка YouTube Music містить підбірки на основі історії прослуховувань. Це означає, що алгоритми програми формують індивідуальні добірки для кожного користувача (Рис. 1.1).

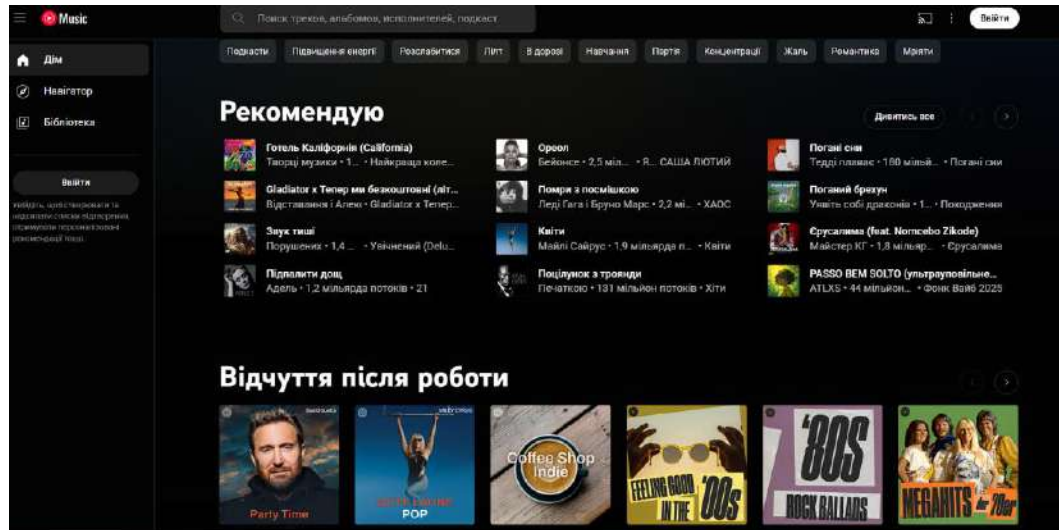

Рис. 1.1. Головна сторінка YouTube Music

У верхній частині інтерфейсу розташована пошукова панель, яка працює за принципами релевантності та машинного навчання. Вона дозволяє здійснювати пошук не тільки за назвою пісні чи іменем виконавця, а й за фрагментами тексту, ключовими словами або навіть настроєм/ Результати пошуку подаються у вигляді категорій: треки, альбоми, виконавці, кліпи (Рис. 1.2).

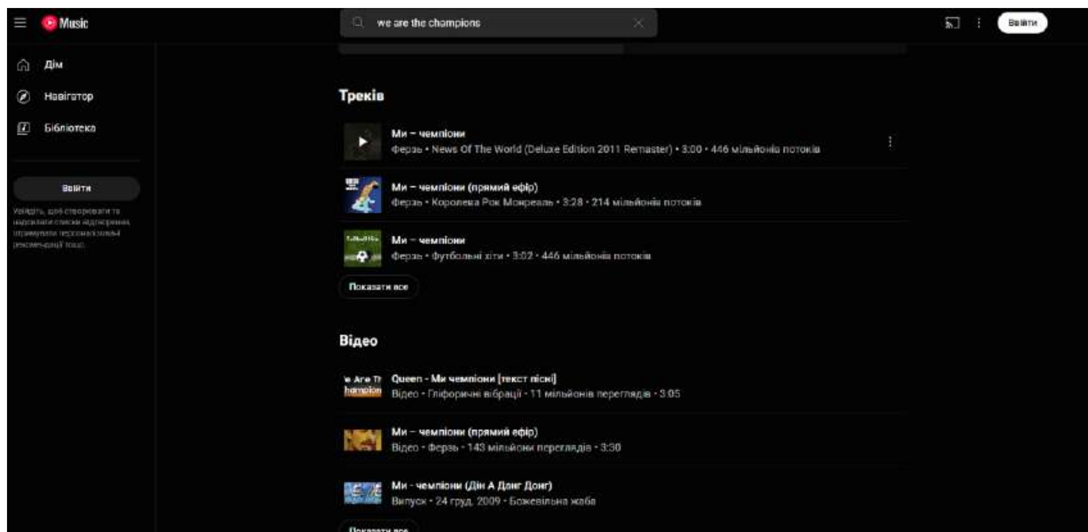

Рис. 1.2. Результати пошуку по словам з тексту треку

На сторінці обраного треку користувач може додати його до власної бібліотеки або до плейлиста (Рис. 1.3).

Рис. 1.3. Сторінка треку

YouTube Music має налаштовану систему рекомендацій. Вона базується на історії переглядів користувача в YouTube, попередніх прослуховуваннях, геолокації, часу доби та інших факторах. Це дозволяє платформі створювати щоденні мікси та персоналізовані добірки з високою точністю. Усі рекомендації оновлюються в режимі реального часу (Рис. 1.4).

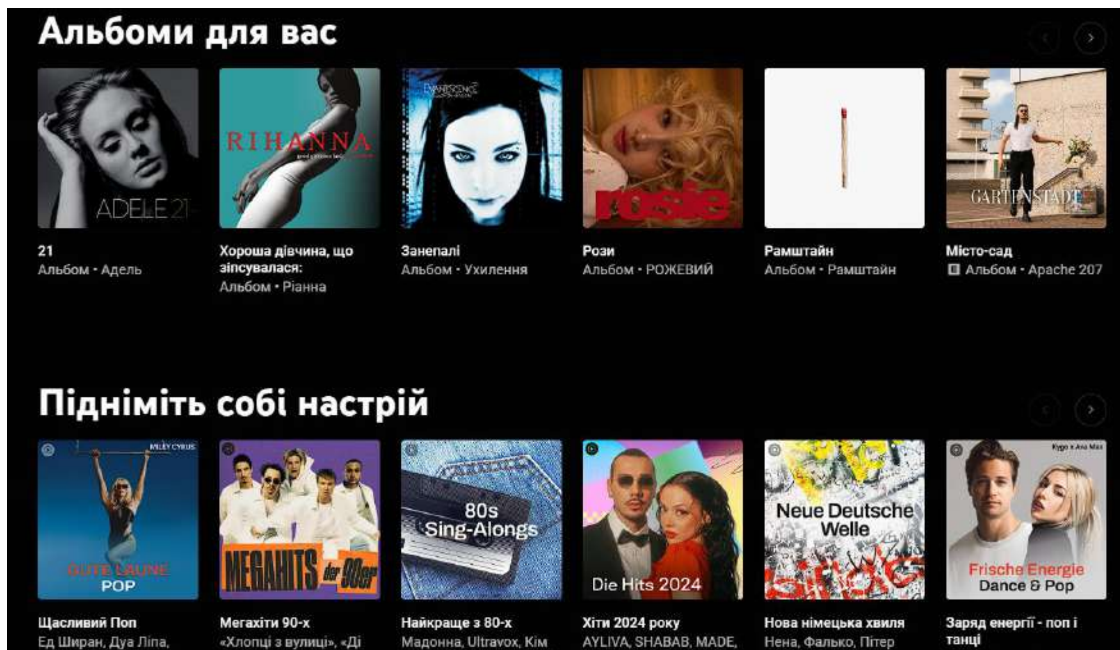

Рис. 1.4. Система рекомендацій

Користувачі можуть додавати пісні до власної бібліотеки, створювати та впорядковувати плейлисти, підписуватись на виконавців і отримувати повідомлення про нові релізи. Плейлисти можна робити публічними або приватними, а також ділитись ними з іншими користувачами платформи. Окремо реалізовано розділ “Нещодавно прослухане”, де зберігаються останні дії користувача (Рис. 1.5.).

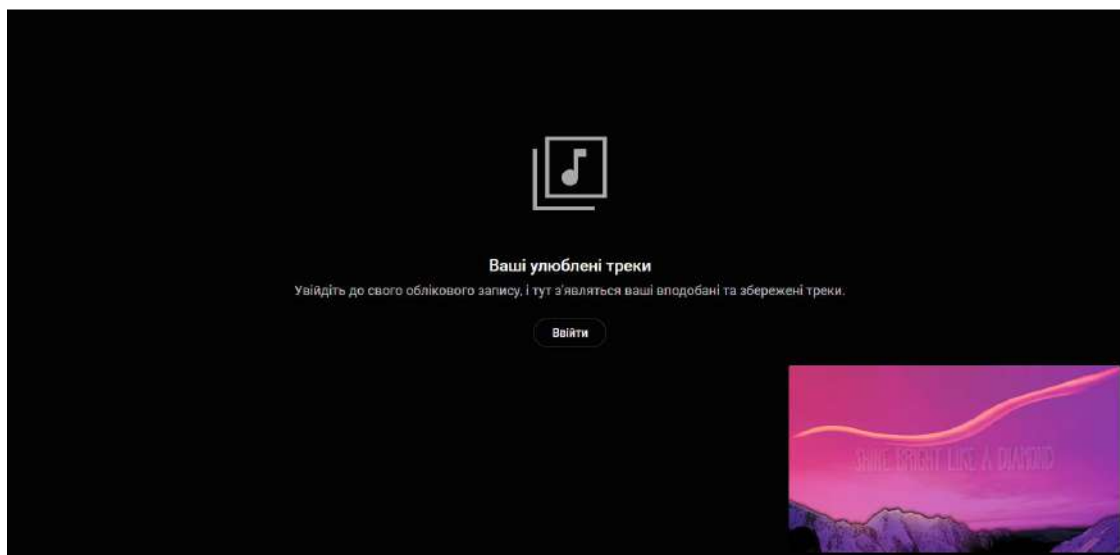

Рис. 1.5. Бібліотека музики

З технічної точки зору, платформа підтримує адаптивний стримінг — якість відтворення автоматично змінюється в залежності від пропускної здатності інтернет-з'єднання. Преміум-користувачі отримують доступ до

прослуховування без реклами, можливість фонової роботи програми та завантаження треків для офлайн-прослуховування.

Функціональність YouTube Music доступна через веберсію, Android- та iOS-додатки. Усі дії синхронізуються між пристроями через єдиний обліковий запис Google. Окремої уваги заслуговує інтеграція з сервісами Google Assistant, завдяки якій можна керувати голосом відтворення музики.

Spotify — одна з найпопулярніших музичних платформ у світі, яка зосереджена суто на стримінгу аудіо. Сервіс забезпечує доступ до понад 100 мільйонів треків і подкастів, а також надає користувачам можливості створення персоналізованих плейлистів, рекомендацій і спільного прослуховування. Основна стратегія платформи полягає в інтелектуальній адаптації контенту під уподобання слухача за допомогою штучного інтелекту та обробки великих обсягів даних (Рис. 1.6).

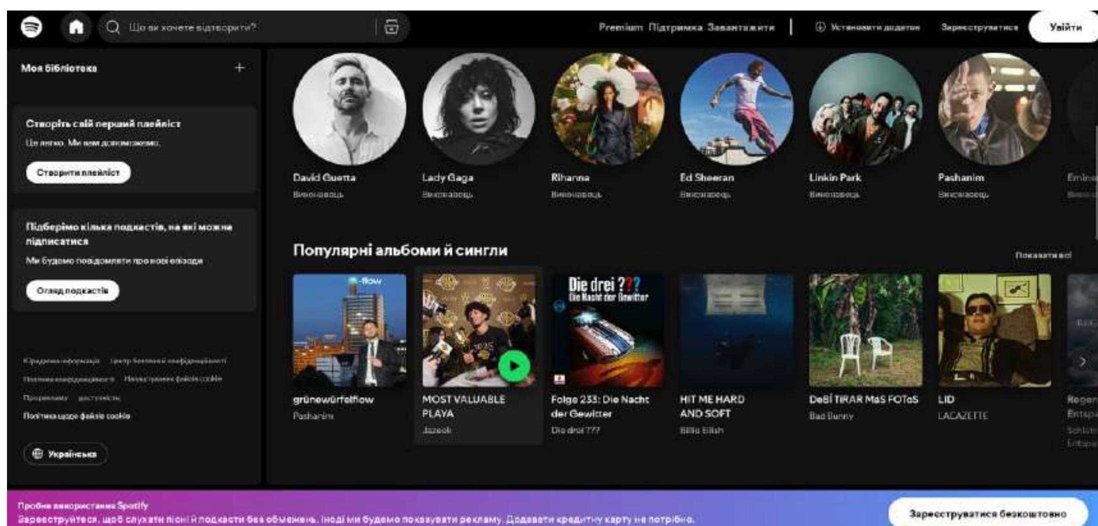

Рис. 1.6. Головна сторінка Spotify

Головна сторінка відображає персоналізовані добірки, нові релізи, популярні списки у країні користувача, а також мікси, сформовані на основі слухацьких звичок. Існує система підбору динамічних плейлистів типу “Daily Mix”, “Discover Weekly” і “Release Radar”, які оновлюються щодня або щотижня (Рис. 1.7).

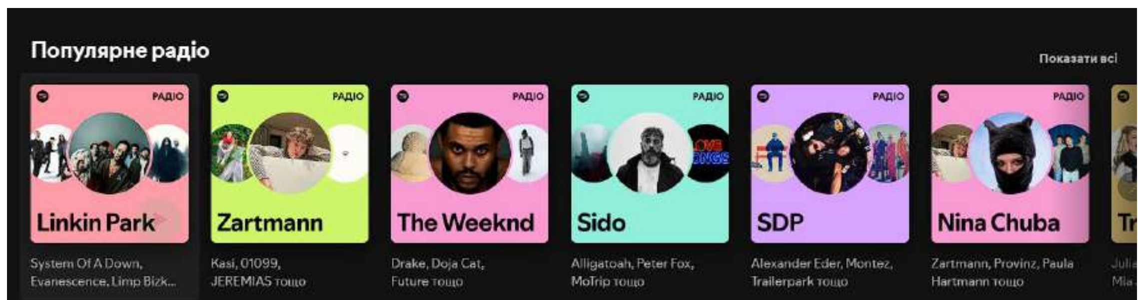

Рис. 1.7. Динамічний підбір радіо по різним критеріям

Функціональність пошуку в Spotify реалізована з використанням автозаповнення, підтримки жанрових і тематичних запитів, а також інтеграції з алгоритмами рекомендацій. Користувачі можуть шукати пісні, альбоми, виконавців, подкасти, конкретні плейлисти під настрій (Рис. 1.8).

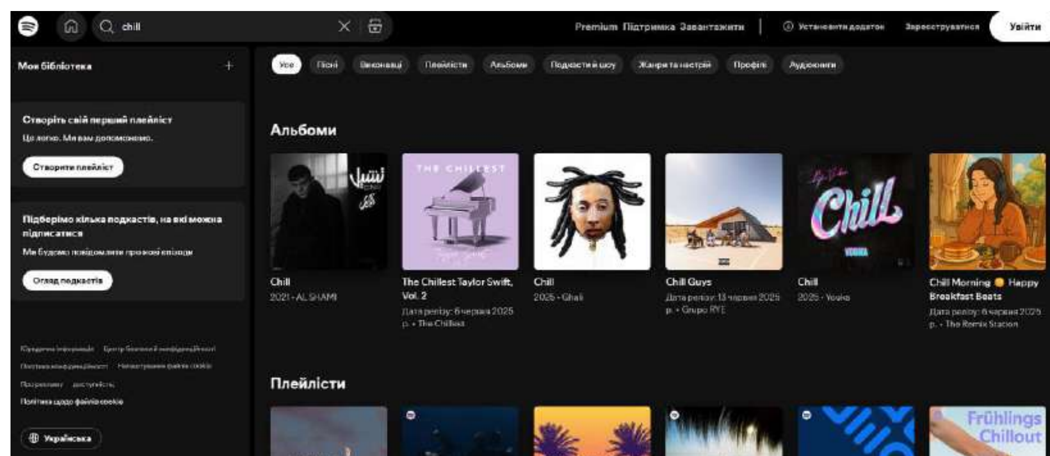

Рис. 1.8. Пошук по ключовому слові настрою

Spotify підтримує збереження музики до бібліотеки, додавання до власних або загальнодоступних плейлистів, та можливість позначати вподобані треки. Кожен користувач має персональну сторінку, де зберігаються його активності: останні прослуховування, обране, створені плейлисти та підписки на виконавців. Також присутній розділ “Нещодавно прослухане” для швидкого доступу до останніх треків.

Перевагою є підтримка спільного відтворення: Spotify дозволяє кільком користувачам створювати спільні плейлисти, а також синхронно слухати музику в режимі "Group Session", що робить платформу соціально-орієнтованою.

Сторінка «Радіо» в інтерфейсі Spotify реалізована як окрема функціональна область, яка дозволяє користувачеві слухати добірки композицій, згенеровані автоматично на основі конкретної пісні, виконавця або жанру. Цей механізм не є прямим аналогом традиційного радіомовлення, оскільки відсутні ефіри, ведучі або фіксована сітка відтворення. Натомість система формує музичний потік, що динамічно підлаштовується під обраний користувачем вихідний контент. Запуск радіо відбувається безпосередньо зі сторінки композиції або альбому — після натискання відповідної кнопки формується добірка треків, які мають подібне звучання або стилістичні риси (Рис. 1.9).

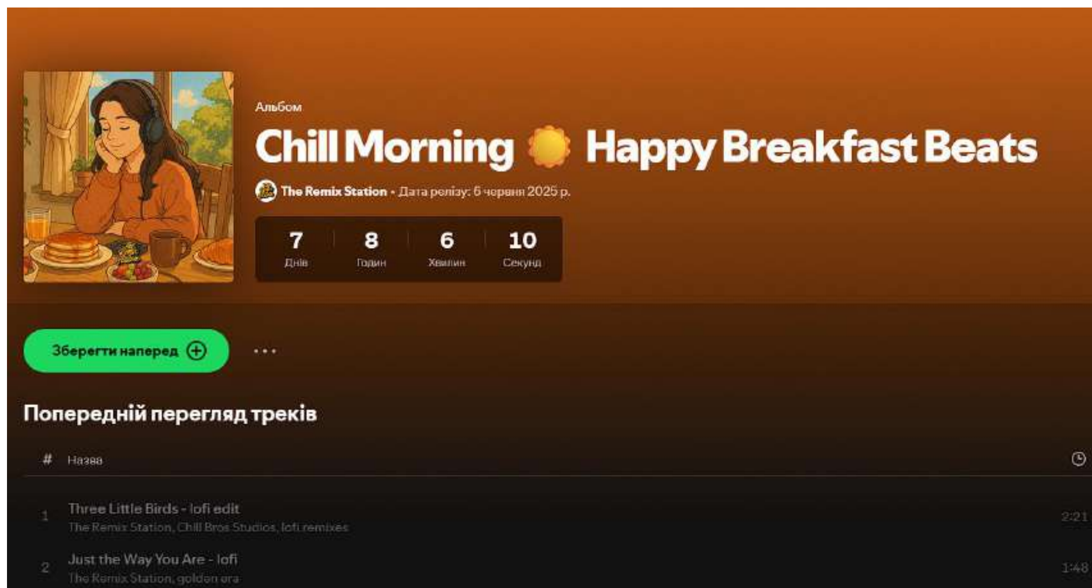

Рис. 1.9. Сторінка радіо “Chill Morning”

На сторінці відображається обкладинка радіо, яке створено на основі вибраного джерела, а також назва добірки. Під цими елементами розташовано перелік треків, для попереднього перегляду. Користувач має змогу пропускати пісні, додавати їх до власної бібліотеки або відмічати як уподобані. Такі дії фіксуються системою і надалі впливають на загальну логіку рекомендацій, хоча на саму структуру конкретної добірки вони впливають лише частково.

Особливістю цього функціонального модуля є відсутність фіксованого завершення — після відтворення останнього треку у списку програвач автоматично додає нові композиції. Таким чином забезпечується безперервність звучання, яка не потребує додаткових дій з боку користувача.

Існують також сторінки «Радіо по треку», яка знаходить подібну музику відносно обраного треку. Сторінка також надає можливість зберегти згенеровану добірку до бібліотеки, після чого вона стає доступною у вигляді звичайного плейлиста. Однак навіть після збереження радіо зберігає здатність оновлюватись із часом, включаючи нові треки за подібною логікою добору.

З технічної сторони, Spotify використовує адаптивне кодування для мінімізації затримок і збереження якості звуку на всіх рівнях підключення. Преміум-підписка забезпечує безрекламний доступ, режим офлайн, можливість перемотування треків, та аудіо з високим бітрейтом (до 320 кбіт/с).

Платформа доступна як через вебінтерфейс, так і через десктопні й мобільні застосунки для всіх основних операційних систем. Усі дії синхронізуються в режимі реального часу. Також реалізовано підтримку інтеграції з іншими сервісами: Discord, Google Maps, Instagram Stories, Android Auto, Apple CarPlay тощо.

1.3. Вимоги до каталогу

Каталог музичного контенту повинен відповідати низці функціональних та структурних вимог, які забезпечують зручність навігації, релевантність вмісту та узгодженість із загальною архітектурою системи. Його побудова має забезпечувати чітке групування елементів за категоріями, можливість динамічного оновлення та інтеграцію із системами рекомендацій.

Передусім, у каталозі повинна бути реалізована підтримка різнорівневої класифікації музичних елементів — за типами трек, альбом, виконавець, добірка, жанрами, тематиками або настроєм. Ця структура повинна залишатися видимою для користувача, водночас залишаючи простір для адаптивного сортування, залежно від запитів чи поведінкових параметрів. При перегляді конкретного виконавця користувач має отримувати доступ не лише до релізів, а й до пов’язаних артистів, створених плейлистів, схожих композицій.

Каталог повинен забезпечувати є швидкий пошук музики по ріщним критеріям. Пошукова система повинна підтримувати роботу з частковим введенням запитів, латиницею та кирилицею, мати механізми виправлення помилок і підказки. Результати повинні повертатися з урахуванням контексту. При введенні імені виконавця система повинна надавати пріоритет відповідним сторінкам артистів, а не випадковим трекам із подібним написанням.

Каталог повинен інтегруватися з системою рекомендацій: відображати персоналізовані блоки новинок, добірок, найбільш прослуховуваних треків, або нових релізів у межах жанрів, які були визначені як релевантні для конкретного користувача. Водночас для нового користувача, який не має ще сформованої історії взаємодії, система має відображати узагальнену структуру.

Окрему увагу необхідно приділити візуальній частині каталогу. Інтерфейс повинен бути однаково зручним як у вебверсії, так і в мобільному додатку. Обкладинки, фотографії виконавців, мініатюри альбомів мають бути достатньо чіткими і мати адаптивні розміри.

Ще одним функціональним елементом каталогу є можливість взаємодії з окремими елементами без потреби переходу на нові сторінки. Натискання на мініатюру композиції має викликати невелике контекстне меню, що дозволяє додати трек до черги, зберегти в добірку або позначити як уподобаний. Ці взаємодії мають бути реалізовані з мінімальними затримками, тому запити до серверної частини мають бути оптимізовані.

Також варто передбачити можливість асинхронного оновлення частини каталогу. При прокручуванні донизу має відбуватися дозавантаження нових елементів без повного перезавантаження сторінки.

У межах реалізації цих вимог має також підтримуватися модульна структура даних, що дозволяє повторно використовувати компоненти для інших частин застосунку. Список треків на сторінці виконавця має бути ідентичним за логікою відображення до списку у плейлисті чи добірці.

Вимоги до авторизації та реєстрації формуються з урахуванням необхідності забезпечення зручного доступу користувачів до платформи, підтримки декількох способів автентифікації, а також поетапного збору обов’язкових персональних даних.

Система повинна підтримувати авторизацію користувача через сторонні облікові записи. Зокрема, має бути реалізована можливість входу через акаунти Google, Facebook та Apple. Взаємодія із зовнішніми платформами повинна здійснюватися за допомогою безпечного протоколу OAuth2. Після підтвердження ідентичності користувач має бути перенаправлений до платформи із збереженням сесії.

Окрім сторонньої автентифікації, має бути реалізована класична форма авторизації за допомогою електронної пошти та паролю. При введенні неправильних даних користувач отримує відповідне повідомлення про помилку. Для цього варіанту необхідно забезпечити функцію відновлення доступу, побудованому на процесі відправлення одноразового посилання на вказану електронну адресу.

Реєстрація нових користувачів повинна бути поділена на три етапи, кожен з яких передбачає заповнення конкретного набору полів. На першому етапі користувач вводить адресу електронної пошти, яка перевіряється на правильність формату та унікальність у системі. На другому етапі створюється пароль, що повинен відповідати встановленим вимогам щодо мінімальної довжини та наявності різних типів символів. Третій етап передбачає заповнення персональної інформації: ім’я, дата народження та стать. Усі ці поля є обов’язковими для завершення реєстрації.

Валідація введених даних повинна здійснюватися як на стороні клієнта, так і на стороні сервера. Усі етапи взаємодії повинні відбуватися без перезавантаження сторінки, з використанням асинхронних запитів. Інтерфейс має бути адаптивним, забезпечуючи коректне відображення та роботу на пристроях з різними розмірами екранів.

Висновки до розділу 1

Підсумовуючи зміст першого розділу, можна зробити висновок: розробка сучасного музичного сервісу вимагає поєднання технічної ретельності, уваги до досвіду користувача та врахування характеру взаємодії з великим обсягом даних. На прикладі платформ YouTube Music та Spotify простежується певна закономірність — інтерфейс для прослуховування музики здається простим лише на перший погляд. Насправді ж за ним стоїть складна система пошуку, персоналізації, обробки історії прослуховувань, рекомендацій, відстеження вподобань, підписок та формування бібліотеки.

РОЗДІЛ 2
РОЗРОБКА АЛГОРИТМУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

Наступним етапом проектування музичної платформи є розробка алгоритмів, що описують логіку функціонування системи (Рис. 2.1.). Алгоритм побудови клієнтської частини базується на постійній синхронізації з сервером, контролі стану, наявності зворотного зв'язку з користувачем та адаптивності інтерфейсу. Це дозволяє зберігати цілісність роботи застосунку, забезпечуючи при цьому динамічний і зручний досвід взаємодії для кожного користувача.

Клієнтська частина взаємодіє з користувачем і формує враження від роботи сервісу. Завданням цього етапу є побудова послідовності дій, які забезпечують цілісну та зрозумілу взаємодію з основними елементами: реєстрація, авторизація, навігація по каталогу, прослуховування музики, додавання до особистої бібліотеки, перегляд історії прослуховувань.

Спочатку було розроблено структуру інтерфейсу. Визначено основні компоненти: головна сторінка, каталог, сторінка входу/реєстрації, плейлісти. Кожен компонент виконує окрему функцію, проте всі вони зв'язані між собою через маршрутизацію. Усі переходи відбуваються без перезавантаження сторінки — використовується модель односторінкового застосунку (SPA), що реалізується за допомогою фреймворку React.

При першому завантаженні застосунку клієнт перевіряє наявність токена доступу у локальному сховищі. Якщо він є, виконується запит до сервера для підтвердження сесії. У разі відсутності або недійсності токена користувач автоматично переадресовується на сторінку авторизації. Увесь цей процес є автоматизованим, а повідомлення про помилки формуються на основі відповіді сервера.

Перевірити наявність токена в локальному сховищі
Токен існує?
так
ні
Відправити запит на сервер для підтвердження сесії
Переадресувати на сторінку авторизації
Сесія дійсна?
так
ні
Завантажити головну сторінку
Переадресувати на сторінку авторизації

Авторизація
Ввести електронну пошту та пароль
Надіслати дані на сервер
Авторизація успішна?
так
ні
Отримати JWT-токен
Показати повідомлення про помилку
Зберегти токен у локальному сховищі
Завантажити головну сторінку

Вхід через сторонні сервіси
Вибрати Google/Facebook/Apple
Авторизуватися через SDK
Отримати токен стороннього сервісу
Надіслати токен серверу
Авторизація успішна?
так
ні
Отримати JWT-токен
Показати повідомлення про помилку
Зберегти токен
Завантажити головну сторінку

Головна сторінка
Відобразити меню навігації
Обробити вибір користувача (Каталог, Плейлисти, Пошук)
Вибір - Каталог?
так
Сформувати запит із фільтрами
Отримати дані з API
Відобразити композиції у вигляді сітки
Вибір - Плейлист?
так
Завантажити вибраний плейлист

Особиста бібліотека
Натиснути кнопку "Додати до бібліотеки"
Відправити запит на сервер
Запит успішний?
так
ні
Оновити стан бібліотеки на клієнті
Показати повідомлення про помилку
Відобразити оновлення у розділі "Збережене"

Рис. 2.1. Алгоритм роботу сайту

Процес авторизації реалізується через форму, яка приймає введені користувачем облікові дані. Верифікація виконується на стороні сервера, після чого у відповідь клієнт отримує JWT-токен. У разі помилки користувач бачить відповідне повідомлення, а поля форми підсвічуються для зручності. Для входу через сервіси Google, Facebook або Apple використовуються відповідні SDK, які після авторизації передають токен сервісу, а вже він ініціює або створює сесію користувача.

Після входу доступ до всіх основних функцій відкривається через головне меню. Користувач може перейти до каталогу, скористатись пошуком або обрати плейлист із запропонованих. Вибрані параметри формують запит до API, який повертає відповідну вибірку композицій. Візуалізація результатів реалізована через сіткове відображення, де кожен елемент є окремим компонентом з назвою, зображенням і кнопками керування.

Додавання до особистої бібліотеки реалізується через кнопку на картці композиції. При натисканні формується запит на сервер, який прив’язує обрану пісню до користувача. Відповідно, на клієнті оновлюється стан бібліотеки, і елемент відображається у розділі «Збережене».

Взаємодія між сервером і клієнтською частиною здійснюється через API на основі протоколу HTTP або його захищеної версії HTTPS. HTTP (HyperText Transfer Protocol) забезпечує передачу даних між клієнтом і сервером у вигляді запитів і відповідей [4]. Версія HTTPS додає до цього процесу шифрування, яке захищає інформацію від перехоплення та несанкціонованого доступу під час передачі. Це потрібно для забезпечення конфіденційності та цілісності даних користувача.

При взаємодії клієнт формує HTTP-запити, які містять метод GET або POST, адресу ресурсу і додаткові дані у вигляді заголовків і тіла запиту. Сервер отримує цей запит, обробляє його згідно з логікою системи, взаємодіє з базою даних, і формує відповідь у форматі JSON, що потім інтерпретується клієнтом.

HTTP-запити типів GET і POST виконують різні функції в процесі обміну даними між клієнтом і сервером. Запит GET призначений для отримання інформації з сервера без зміни його стану. Такий запит передає параметри у URL-адресі і використовується для завантаження даних, наприклад, списку музичних композицій або інформації про користувача. На відміну від GET, запит POST застосовується для передачі даних на сервер з метою створення або зміни ресурсів. У випадку реєстрації користувача або додавання пісні до особистої бібліотеки POST-запит містить у тілі інформацію, яку сервер обробляє та зберігає.

JSON (JavaScript Object Notation) — це формат обміну даними, який використовується для передачі структурованої інформації між клієнтом і сервером [8]. Він формується у текстовій формі, тому він легко читається і може вписувати дані, а також швидко парсити їх у програмному коді. JSON представляє інформацію у вигляді пар «ключ-значення», де ключі — це рядки, а значення можуть бути різних типів: числа, рядки, логічні значення, масиви чи вкладені об’єкти. JSON широко застосовується у веб-розробці для обміну інформацією.

JWT-токен (JSON Web Token) використовується для передачі інформації про користувача у вигляді цифрового підпису. Він складається з трьох частин: заголовка, корисних даних (payload) та підпису. Заголовок визначає тип токена і алгоритм підпису, у корисних даних зберігаються відомості про користувача та термін дії токена. Під час автентифікації користувача сервер приймає облікові дані, перевіряє їх і у випадку успішної верифікації створює JWT-токен. Цей токен повертається клієнту і надалі додається до заголовків HTTP-запитів для ідентифікації користувача і контролю доступу до ресурсів платформи. У разі надходження запитів на пошук музичних композицій, додавання їх до особистої бібліотеки або інших дій сервер обробляє інформацію, взаємодіє з базою даних і надсилає відповідь, яка містить необхідні дані або підтвердження виконання операції.

Висновки до розділу 2

У результаті розробки алгоритму клієнтської частини визначено послідовність дій, яка забезпечує інтерактивну та безперервну взаємодію користувача із сервісом. Встановлено, що основою функціонування є синхронізація зі сервером і підтримка актуального стану застосунку, що дозволяє швидко реагувати на зміни та забезпечує комфортний досвід користування. Особливу увагу приділено обробці авторизації і навігації, які реалізовані через SPA-модель з використанням React, усуваючи необхідність перезавантаження сторінок і сприяючи плавній роботі інтерфейсу.

Взаємодія з сервером побудована на основі протоколів HTTP і HTTPS із застосуванням методів GET і POST для отримання та відправлення даних. Відповіді сервера формуються у форматі JSON, що спрощує обробку інформації на стороні клієнта. Для підтримки безпеки сесій використовується JWT-токен, який дозволяє ідентифікувати користувача без повторної аутентифікації при кожному запиті.

Розроблений алгоритм враховує особливості взаємодії між компонентами клієнтської частини і сервером, забезпечуючи можливість масштабування системи і підтримку високої навантаженості. Виконана робота створює основу для подальшої реалізації функціоналу платформи, формуючи технічну базу для забезпечення зручності і надійності сервісу.

РОЗДІЛ 3
ОРГАНІЗАЦІЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Вибір системи управління бази даних

Розробка серверної частини музичної платформи передбачає визначення технологічного стеку, який забезпечує зберігання, обробку та доступ до даних. Основою для серверного коду служить платформа .NET, що відкриває можливості для створення додатків з підтримкою різних мов програмування та типів баз даних. Вибір системи управління базою даних впливає на організацію роботи з користувацькими обліковими записами, інформацією про музику, плейлисти та інші дані.

Платформа .NET являє собою середовище виконання та розробки, яке підтримує взаємодію з різноманітними базами даних через стандартизовані інтерфейси. Завдяки цьому можна обирати СУБД відповідно до потреб проекту, а також легко інтегрувати базу даних у загальну архітектуру додатка. У роботі з даними використовується Entity Framework — ORM технологія, що дозволяє оперувати з базою даних на рівні об’єктів, абстрагуючись від низькорівневих SQL-запитів.

Одним із основних інструментів розробника виступає середовище Visual Studio. Воно надає широкі можливості для створення, тестування і налагодження серверних застосунків. Середовище підтримує інтеграцію з базами даних, дозволяючи підключатись до СУБД, переглядати структуру таблиць, виконувати SQL-запити, а також створювати та виконувати міграції бази даних безпосередньо з інтерфейсу. Важливою особливістю Visual Studio є підтримка різних технологій для аутентифікації користувачів, розробки REST API, а також можливість швидко вбудовувати сторонні бібліотеки та інструменти.

Для організації бази даних обрано локальну систему Microsoft SQL Server. Ця СУБД має стабільний набір функцій для роботи з реляційними

даними, підтримує транзакції, індекси, різноманітні типи зв'язків між таблицями. Вона гарантує цілісність даних, надає розширені можливості для керування безпекою доступу, а також включає механізми резервного копіювання і відновлення. Використання Microsoft SQL Server в локальному середовищі відповідає умовам розробки, оскільки забезпечує швидкий доступ до даних і надійну роботу на рівні серверу.

Підключення клієнтської частини серверної системи до бази даних відбувається через стандартизовані протоколи TCP/IP. TCP/IP — це набір протоколів, який забезпечує передачу даних у комп'ютерних мережах [7]. Він складається з двох основних частин: протоколу керування передачею (TCP) і протоколу інтернету (IP). IP відповідає за адресацію і маршрутизацію пакетів даних між пристроями, а TCP забезпечує надійну передачу цих пакетів, контролюючи їх доставку, порядок та цілісність інформації. Завдяки TCP/IP комп'ютери можуть обмінюватися даними у різних мережах, утворюючи єдину глобальну мережу — Інтернет, а також локальні мережі.

Використання TCP/IP дозволяє здійснювати стабільну передачу інформації між сервером додатка і базою даних. У разі локального розгортання застосовується версія SQL Server Express, яка містить базовий набір функцій і забезпечує зберігання даних в межах обмежень за обсягом. Проте ці можливості повністю покривають потреби розробки і дозволяють ефективно працювати з даними на ранніх етапах.

Обрана конфігурація сприяє раціональному використанню ресурсів і надає зручний інтерфейс для розробника під час налаштування і взаємодії з базою даних, забезпечує необхідні можливості для виконання запитів, оновлення даних і керування користувачами.

Варто зазначити, що вибір Microsoft SQL Server пов'язаний з її можливістю зберігати різноманітні типи даних, включно з текстовою інформацією, числовими значеннями та мультимедійними метаданими. Ця система дає змогу створювати складні структури таблиць і встановлювати між

ними логічні зв’язки, що є необхідним для формування каталогу музики з урахуванням авторів, альбомів, жанрів та інших атрибутів.

Перевагою MS SQL Server виступає стек зручних інструментів для роботи з користувацькими обліковими записами. Система дозволяє створювати ролі і визначати права доступу для різних категорій користувачів. Це підвищує безпеку системи і контролює доступ до конфіденційної інформації. Крім того, доступна інтеграція з механізмами аутентифікації на сервері, що дозволяє реалізовувати комплексні моделі управління користувачами і сесіями.

У Visual Studio налагодження підключення до MS SQL Server здійснюється через спеціальний менеджер з’єднань, який зберігає конфігурацію і спрощує роботу з базою даних під час розробки. Цей інструмент дозволяє швидко перевіряти структуру таблиць, виконувати запити та керувати даними без необхідності виходити з середовища програмування. Завдяки цьому відпадає потреба у використанні додаткових сторонніх програм для роботи з базою даних.

Для передачі даних між сервером і клієнтом застосовується формат JSON. Він легко перетворюється у внутрішні об’єкти програми та підтримується більшістю мов програмування. JSON дозволяє компактно передавати складні об’єкти з різними типами даних, що робить його оптимальним вибором для обміну інформацією в межах архітектури веб-додатка. Завдяки цьому дані, які зберігаються в MS SQL Server, можуть бути представлені у зручному для клієнта вигляді.

3.2. Процес міграції даних

Міграції даних — це процес перенесення, оновлення або трансформації структури та вмісту бази даних у контексті розвитку програмного забезпечення. Під час розробки системи структура бази даних може змінюватися через додавання нових таблиць, зміну полів, виправлення

помилок або оптимізацію. Міграції дозволяють автоматизувати ці зміни, забезпечуючи послідовне оновлення схеми бази та збереження існуючих даних.

Застосування міграцій необхідне для підтримки узгодженості між різними версіями програмного продукту, особливо коли система розгортається в різних середовищах. Відсутність упорядкованого процесу міграцій може призвести до розбіжностей у структурах баз даних, помилок під час запуску застосунку або втрати даних.

Перша реалізована міграція бази даних має на меті створити структуру таблиць, необхідних для збереження даних користувачів і ролей, використовуючи стандартні компоненти ASP.NET Identity. Вона визначає таблиці, які будуть зберігати інформацію про користувачів, їх ролі, а також пов'язані з ними права та токени.

У методі Up прописано послідовність створення таблиць. Таблиця "AspNetUsers" зберігає основні дані про користувачів, включно з унікальним ідентифікатором, електронною поштою, паролем у хешованому вигляді та іншими параметрами безпеки. Таблиця "AspNetRoles" містить ролі, які можна призначати користувачам. Зв’язки між таблицями реалізовані через зовнішні ключі, що забезпечують цілісність даних.

```
migrationBuilder.CreateTable(
    name: "AspNetUsers",
    columns: table => new
    {
        Id = table.Column<string>(type: "nvarchar(450)", nullable: false),
        UserName = table.Column<string>(type: "nvarchar(256)", maxLength:
256, nullable: true),
        NormalizedUserName = table.Column<string>(type: "nvarchar(256)",
maxLength: 256, nullable: true),
        Email = table.Column<string>(type: "nvarchar(256)", maxLength: 256,
nullable: true),
        NormalizedEmail = table.Column<string>(type: "nvarchar(256)",
maxLength: 256, nullable: true),
        EmailConfirmed = table.Column<bool>(type: "bit", nullable: false),
        PasswordHash = table.Column<string>(type: "nvarchar(max)", nullable:
true),
```

```
            SecurityStamp = table.Column<string>(type: "nvarchar(max)", nullable: true),
            ConcurrencyStamp = table.Column<string>(type: "nvarchar(max)", nullable: true),
            PhoneNumber = table.Column<string>(type: "nvarchar(max)", nullable: true),
            PhoneNumberConfirmed = table.Column<bool>(type: "bit", nullable: false),
            TwoFactorEnabled = table.Column<bool>(type: "bit", nullable: false),
            LockoutEnd = table.Column<DateTimeOffset>(type: "datetimeoffset", nullable: true),
            LockoutEnabled = table.Column<bool>(type: "bit", nullable: false),
            AccessFailedCount = table.Column<int>(type: "int", nullable: false)
          },
          constraints: table =>
          {
            table.PrimaryKey("PK_AspNetUsers", x => x.Id);
          });
```

Метод Down забезпечує можливість скасування міграції, видаляючи створені таблиці. Це полегшує відкат змін у разі потреби.

Наступна міграція створює таблицю UserRecentlyListened, яка служить для збереження історії прослуханих користувачем композицій. Така таблиця дозволяє фіксувати інформацію про контент, який нещодавно був відтворений, що дозволяє створювати підбірки рекомендацій і допомагає персоналізувати досвід користувача.

У структурі таблиці передбачено унікальний ідентифікатор Id, який автоматично збільшується для кожного нового запису. Поле UserId пов’язане з таблицею користувачів і вказує на того, хто слухав музику. Для ідентифікації самого елемента, наприклад пісні чи плейлиста, передбачене поле ItemId. Поле ItemType використовується для розмежування типів контенту, що зберігається — це дозволяє системі працювати не лише з треками, а й з іншими видами медіа.

Дата і час додавання запису фіксуються у полі DateAdded. Це допомагає зберегти хронологію прослуховувань, що в подальшому дає можливість формувати актуальні списки відтворення або аналізувати активність користувачів.

Для запобігання дублюванню однакових записів для одного користувача використано унікальний індекс на комбінацію UserId, ItemId та ItemType. Це гарантує, що один і той самий елемент не буде зберігатися у списку недавно прослуханих повторно.

```
migrationBuilder.CreateTable(
    name: "UserRecentlyListened",
    columns: table => new
    {
        Id = table.Column<int>(type: "int", nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        UserId = table.Column<string>(type: "nvarchar(450)", nullable: false),
        ItemId = table.Column<string>(type: "nvarchar(450)", nullable: false),
        ItemType = table.Column<string>(type: "nvarchar(450)", nullable: false),
        DateAdded = table.Column<DateTime>(type: "datetime2", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_UserRecentlyListened", x => x.Id);
        table.ForeignKey(
            name: "FK_UserRecentlyListened_AspNetUsers_UserId",
            column: x => x.UserId,
            principalTable: "AspNetUsers",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });

migrationBuilder.CreateIndex(
    name: "IX_UserRecentlyListened_UserId_ItemId_ItemType",
    table: "UserRecentlyListened",
    columns: new[] { "UserId", "ItemId", "ItemType" },
    unique: true);
}
```

У разі скасування цієї міграції метод Down видаляє створену таблицю, повертаючи структуру бази даних до попереднього стану.

Наступна міграція вводить таблицю UserSubscriptions, яка призначена для зберігання інформації про підписки користувачів на певні елементи

контенту платформи. Така структура дозволяє відслідковувати інтереси аудиторії, забезпечуючи зручний доступ до вподобаних або важливих для користувача об’єктів.

У таблиці передбачено автоматично інкрементований ідентифікатор Id, що унікально ідентифікує кожен запис. Поле UserId пов’язане з користувачами системи і вказує, хто оформив підписку. Поля ItemId та ItemType визначають конкретний елемент, на який оформлено підписку, причому ItemType дозволяє розрізняти різні види контенту, наприклад, артиста, плейлист або альбом.

Для відмітки часу оформлення підписки використовується поле CreatedAt. Це допомагає вести хронологію підписок і може стати у нагоді для подальшої аналітики або реалізації функції сповіщень.

Унікальний індекс покриває комбінацію UserId, ItemId та ItemType, гарантує відсутність дублювань, тобто один користувач не зможе підписатися на один і той самий елемент більше одного разу. Це спрощує роботу з базою та підтримує цілісність даних.

```
migrationBuilder.CreateTable(
    name: "UserSubscriptions",
    columns: table => new
    {
        Id = table.Column<int>(type: "int", nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        UserId = table.Column<string>(type: "nvarchar(450)", nullable: false),
        ItemId = table.Column<string>(type: "nvarchar(450)", nullable: false),
        ItemType = table.Column<string>(type: "nvarchar(450)", nullable: false),
        CreatedAt = table.Column<DateTime>(type: "datetime2", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_UserSubscriptions", x => x.Id);
        table.ForeignKey(
            name: "FK_UserSubscriptions_AspNetUsers_UserId",
            column: x => x.UserId,
            principalTable: "AspNetUsers",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
```

```
});
```

Метод Down у разі потреби видаляє таблицю UserSubscriptions, повертаючи схему бази даних до попереднього стану без слідів цієї міграції.

Впровадження таблиці для збереження підписок відкриває додаткові можливості для персоналізації платформи. Користувачі отримують змогу слідкувати за оновленнями улюблених виконавців або плейлистів, а система — відслідковувати популярність контенту й пропонувати релевантні рекомендації.

```
modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRole", b =>
    {
        b.Property<string>("Id")
            .HasColumnType("nvarchar(450)");

        b.Property<string>("ConcurrencyStamp")
            .IsConcurrencyToken()
            .HasColumnType("nvarchar(max)");

        b.Property<string>("Name")
            .HasMaxLength(256)
            .HasColumnType("nvarchar(256)");

        b.Property<string>("NormalizedName")
            .HasMaxLength(256)
            .HasColumnType("nvarchar(256)");

        b.HasKey("Id");

        b.HasIndex("NormalizedName")
            .IsUnique()
            .HasDatabaseName("RoleNameIndex")
            .HasFilter("[NormalizedName] IS NOT NULL");

        b.ToTable("AspNetRoles", (string)null);
    });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<string>", b =>
    {
        b.Property<int>("Id")
            .ValueGeneratedOnAdd()
            .HasColumnType("int");
```

```
SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"));

                b.Property<string>("ClaimType")
                    .HasColumnType("nvarchar(max)");

                b.Property<string>("ClaimValue")
                    .HasColumnType("nvarchar(max)");

                b.Property<string>("RoleId")
                    .IsRequired()
                    .HasColumnType("nvarchar(450)");

                b.HasKey("Id");

                b.HasIndex("RoleId");

                b.ToTable("AspNetRoleClaims", (string)null);
            });

        modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<str
ing>", b =>
            {
                b.Property<int>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("int");


SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"));

                b.Property<string>("ClaimType")
                    .HasColumnType("nvarchar(max)");

                b.Property<string>("ClaimValue")
                    .HasColumnType("nvarchar(max)");

                b.Property<string>("UserId")
                    .IsRequired()
                    .HasColumnType("nvarchar(450)");

                b.HasKey("Id");

                b.HasIndex("UserId");
```

```
            b.ToTable("AspNetUserClaims", (string)null);
        });

        modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<str
ing>", b =>
        {
            b.Property<string>("LoginProvider")
                .HasColumnType("nvarchar(450)");

            b.Property<string>("ProviderKey")
                .HasColumnType("nvarchar(450)");

            b.Property<string>("ProviderDisplayName")
                .HasColumnType("nvarchar(max)");

            b.Property<string>("UserId")
                .IsRequired()
                .HasColumnType("nvarchar(450)");

            b.HasKey("LoginProvider", "ProviderKey");

            b.HasIndex("UserId");

            b.ToTable("AspNetUserLogins", (string)null);
        });

        modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<stri
ng>", b =>
        {
            b.Property<string>("UserId")
                .HasColumnType("nvarchar(450)");

            b.Property<string>("RoleId")
                .HasColumnType("nvarchar(450)");

            b.HasKey("UserId", "RoleId");

            b.HasIndex("RoleId");

            b.ToTable("AspNetUserRoles", (string)null);
        });

        modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<str
ing>", b =>
        {
```

```
            b.Property<string>("UserId")
                .HasColumnType("nvarchar(450)");

            b.Property<string>("LoginProvider")
                .HasColumnType("nvarchar(450)");

            b.Property<string>("Name")
                .HasColumnType("nvarchar(450)");

            b.Property<string>("Value")
                .HasColumnType("nvarchar(max)");

            b.HasKey("UserId", "LoginProvider", "Name");

            b.ToTable("AspNetUserTokens", (string)null);
        });
```

Ця міграція — це знімок поточного стану моделі бази даних у Entity Framework Core, який відображає структуру таблиць і зв’язків на момент її створення. Вона автоматично генерується, щоб зафіксувати, як виглядає вся схема бази даних, включно з таблицями для аутентифікації та ролей, а також вашими власними таблицями.

У ній описані стандартні таблиці Identity, такі як користувачі, ролі, їхні права і логіни, з відповідними властивостями та обмеженнями. Крім того, є опис вашої таблиці UserRecentlyListened, яка зберігає інформацію про те, які елементи користувач нещодавно прослухав, з унікальним індексом по користувачу, айді елемента та типу елемента. Також присутня таблиця UserSubscriptions, що зберігає підписки користувачів на різний контент і має подібну унікальну індексацію та зв’язок з користувачем.

У міграції визначені всі ключі та зовнішні зв’язки, а також вказано, що при видаленні користувача пов’язані записи у цих таблицях видаляються автоматично завдяки каскадним правилам. Це дозволяє підтримувати цілісність даних.

3.3. Структура бази даних

База даних складається з кількох основних сутностей, кожна з яких реалізована через таблиці з відповідними полями.

Таблиця AspNetUsers зберігає дані користувачів (табл. 3.1.). Вона містить ідентифікатор користувача, ім’я, електронну пошту, пароль, статус підтвердження пошти, інформацію про блокування, телефон, двофакторну автентифікацію та інші атрибути, які потрібні для ідентифікації і безпеки користувачів.

Таблиця 3.1

AspNetUsers

Поле	Тип	Опис
1	2	3
Id	nvarchar(450)	Унікальний ідентифікатор користувача (PK)
UserName	nvarchar(256)	Ім'я користувача
NormalizedUserName	nvarchar(256)	Нормалізоване ім'я користувача
Email	nvarchar(256)	Email користувача
NormalizedEmail	nvarchar(256)	Нормалізований Email
EmailConfirmed	bit	Підтвердження Email
PasswordHash	nvarchar(max)	Хеш пароля
SecurityStamp	nvarchar(max)	Маркер безпеки
ConcurrencyStamp	nvarchar(max)	Токен для контролю версій
PhoneNumber	nvarchar(max)	Номер телефону

Продовження табл. 3.1

1	2	3
PhoneNumberConfirmed	bit	Підтвердження номера телефону
TwoFactorEnabled	bit	Двофакторна автентифікація
LockoutEnd	datetimeoffset?	Кінець блокування акаунта
LockoutEnabled	bit	Чи дозволено блокування
AccessFailedCount	int	Лічильник невдалих спроб входу

Таблиця AspNetRoles відповідає за ролі, які можуть бути призначені користувачам (табл. 3.2). Вона зберігає унікальний ідентифікатор ролі, її назву, а також нормалізовану назву для пошуку. Ролі можуть мати пов'язані права, що зберігаються у таблиці AspNetRoleClaims.

Таблиця 3.2

AspNetRoles

Поле	Тип	Опис
Id	nvarchar(450)	Унікальний ідентифікатор ролі (PK)
Name	nvarchar(256)	Назва ролі
NormalizedName	nvarchar(256)	Нормалізована назва ролі
ConcurrencyStamp	nvarchar(max)	Токен для контролю версій

Для зв’язку користувачів із ролями використовується таблиця AspNetUserRoles (табл. 3.3), що містить пару з ідентифікаторів користувача і ролі.

AspNetUserClaims (права користувача) зберігає додаткові твердження (claims), які асоціюються з окремими користувачами (табл. 3.4). Claim — це пара "ключ–значення", вона описує додаткову інформацію про користувача, таку як роль, вік, доступ до певного ресурсу. Ці дані використовуються для авторизації в системі. Кожен запис пов’язаний із конкретним користувачем через поле UserId.

Таблиця 3.3

AspNetUserRoles

Поле	Тип	Опис
UserId	nvarchar(450)	Ідентифікатор користувача (PK, FK)
RoleId	nvarchar(450)	Ідентифікатор ролі (PK, FK)

Таблиця 3.4

AspNetUserClaims

Поле	Тип	Опис
Id	int	Унікальний ідентифікатор (PK)
UserId	nvarchar(450)	Ідентифікатор користувача (FK)
ClaimType	nvarchar(max)	Тип права
ClaimValue	nvarchar(max)	Значення права

Таблиця AspNetRoleClaims зберігає твердження (claims), які прив’язані не до конкретного користувача, а до ролі (табл. 3.5). Роль — це узагальнений рівень доступу або набір прав, який можна присвоїти багатьом користувачам (наприклад, "admin", "moderator", "subscriber").

Таблиця 3.5

AspNetRoleClaims

Поле	Тип	Опис
Id	int	Унікальний ідентифікатор (PK)
RoleId	nvarchar(450)	Ідентифікатор ролі (FK)
ClaimType	nvarchar(max)	Тип права
ClaimValue	nvarchar(max)	Значення права

AspNetUserLogins (дані зовнішніх логінів) використовується для зберігання інформації про зовнішні методи входу користувачів у систему (табл. 3.6).

Таблиця 3.6

AspNetUserLogins

Поле	Тип	Опис
LoginProvider	nvarchar(450)	Провайдер логіну (PK)
ProviderKey	nvarchar(450)	Ключ провайдера (PK)
ProviderDisplayName	nvarchar(max)	Відображуване ім’я провайдера
UserId	nvarchar(450)	Ідентифікатор користувача (FK)

В таблиці зберігаються назви провайдерів, ключі авторизації та відповідність цим обліковкам у системі. Це дозволяє користувачам входити в застосунок без створення окремого логіна і пароля — через наявний обліковий запис у зовнішньому сервісі.

AspNetUserTokens (токени для сесій) призначена для зберігання токенів, які система видає користувачам (табл. 3.7).

Таблиця 3.7

AspNetUserTokens

Поле	Тип	Опис
UserId	nvarchar(450)	Ідентифікатор користувача (PK, FK)
LoginProvider	nvarchar(450)	Провайдер логіну (PK)
Name	nvarchar(450)	Назва токена (PK)
Value	nvarchar(max)	Значення токена

Токени можуть використовуватись для збереження сесій, підтвердження email, скидання пароля або дій, які вимагають додаткового рівня перевірки користувача. Записи в таблиці містять назву провайдера, ім'я токена, його значення та ідентифікатор користувача. Це дозволяє системі відслідковувати активні токени та забезпечити більш безпечну взаємодію з обліковим записом.

Власні таблиці розроблені для розширення функціоналу. Таблиця UserRecentlyListened містить записи про те, що користувач нещодавно слухав (табл. 3.8).

Таблиця 3.8

UserRecentlyListened

Поле	Тип	Опис
Id	int	Унікальний ідентифікатор запису (PK, автоінкремент)
UserId	nvarchar(450)	Ідентифікатор користувача (FK)
ItemId	nvarchar(450)	Ідентифікатор елемента, що прослухали
ItemType	nvarchar(450)	Тип елемента (наприклад, трек, альбом)
DateAdded	datetime2	Дата і час додавання до списку

Кожен запис має унікальний ідентифікатор, посилання на користувача, ідентифікатор елемента, тип цього елемента та дату додавання. Унікальний індекс не дозволяє дублювати записи для однієї й тієї ж комбінації користувач-елемент-тип.

Таблиця UserSubscriptions зберігає підписки користувачів на різні елементи (табл. 3.9). Вона також має ідентифікатор, посилання на користувача, ідентифікатор елемента, тип елемента і дату створення підписки. Для комбінації користувач — елемент — тип підписка унікальна, що виключає дублікати.

Таблиця 3.9

UserSubscriptions

Поле	Тип	Опис
Id	int	Унікальний ідентифікатор запису (PK, автоінкремент)
UserId	nvarchar(450)	Ідентифікатор користувача (FK)
ItemId	nvarchar(450)	Ідентифікатор підписаного елемента
ItemType	nvarchar(450)	Тип елемента, на який оформлена підписка
CreatedAt	datetime2	Дата і час створення підписки

В обох власних таблицях встановлені зовнішні ключі, які забезпечують цілісність даних: записи прив’язуються до існуючих користувачів, а видалення користувача спричиняє видалення пов’язаних з ним записів.

У цій базі даних центральною сутністю є таблиця AspNetUsers, яка містить основну інформацію про користувачів системи. Вона має численні зв’язки з іншими таблицями, що забезпечують базову ідентифікацію та автентифікацію, розширений функціонал. Таблиця AspNetUserClaims зберігає

claims — додаткові атрибути або дозволи, які призначаються окремим користувачам. Таблиця AspNetUserLogins використовується для підтримки зовнішніх систем входу, таких як Google або Facebook, дозволяючи одному користувачу мати кілька записів для різних платформ. AspNetUserTokens містить токени, що використовуються для підтримки автентифікації, наприклад, у процесі відновлення паролю або підтвердження дій.

Таблиця AspNetUserRoles реалізує зв’язок багато-до-багатьох між користувачами та ролями. Один користувач може мати кілька ролей, наприклад "admin", "user", а кожна роль може бути призначена багатьом користувачам. У свою чергу, таблиця AspNetRoles представляє самі ролі, а AspNetRoleClaims зберігає claims, які належать до кожної ролі — таким чином, ролі можуть нести в собі набір дозволів, що автоматично застосовуються до всіх користувачів із цією роллю.

Поза межами механізмів автентифікації, існують дві окремі сутності, які описують дії користувачів у контексті контенту: UserRecentlyListened та UserSubscriptions. Вони обидві мають зв’язок один-до-багатьох з таблицею AspNetUsers через поле UserId. Кожен користувач може мати багато прослуханих елементів (записаних у UserRecentlyListened) та багато підписок (у UserSubscriptions). Обидві таблиці містять поля ItemId та ItemType, які дозволяють зберігати інформацію про різні типи контенту — альбоми, виконавців або подкасти. Унікальний індекс на комбінацію UserId, ItemId і ItemType забезпечує, щоб одна й та сама підписка або прослуховування не були записані двічі для одного користувача.

Висновки до розділу 3

У третьому розділі було побудовано структуру бази даних, яка лежить в основі функціонування веб-застосунку. Було проаналізовано склад таблиць, їхні поля та зв’язки між ними. Особливу увагу приділено таблицям, пов’язаним із системою аутентифікації, які забезпечують зберігання

інформації про користувачів, їхні ролі, зовнішні логіни та токени доступу. Це дозволяє системі надійно і гнучко керувати правами доступу та автентифікацією користувачів.

Окремо розглянуто сутності, що відображають активність користувачів — збереження історії прослуховувань та підписок. Вони реалізовані у вигляді окремих таблиць, які мають прямі зв’язки з користувачами, що дозволяє формувати персоналізований контент та реалізовувати рекомендаційні функції.

РОЗДІЛ 4
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Авторизація та реєстрація

При першому відвідуванні сайту, користувач опиняється на сайті авторизації. Функціонал авторизації та реєстрації користувачів реалізований як окрема частина інтерфейсу веб-застосунку та забезпечує можливість входу або створення нового облікового запису через кілька доступних способів [A]. Розділ має форму входу та може переадресувати на багатокроковий процес реєстрації нового користувача.

Інтерфейс авторизації (Рис. 4.1) розміщений на одній сторінці та містить логотип каталогу у верхній частині. Нижче розташовані кнопки для входу через зовнішні служби Google, Facebook та Apple. Ці варіанти дозволяють користувачеві здійснити авторизацію через сторонній акаунт без введення пароля вручну в кілька кроків.

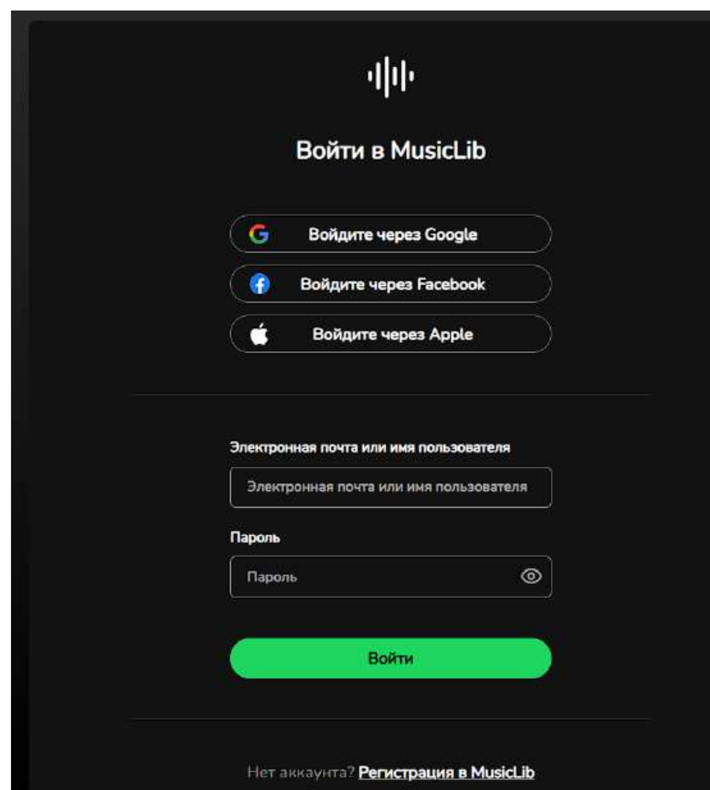

Рис. 4.1. Інтерфейс форми авторизації

Нижче кнопок входу через зовнішні ресурси розміщене текстове поле для введення електронної пошти користувача, а під ним — поле для введення пароля. Обидва поля супроводжуються відповідними плейсхолдерами для пояснення очікуваного формату даних. Під формою авторизації знаходиться кнопка "Увійти". Унизу розташовано текстове посилання "Реєстрація", що відкриває відповідну форму створення нового облікового запису. У разі помилок входу, коли користувач не ввів дані, ввів невалідні дані або ввів дані не авторизованого користувача, з'являється червоний текст під полями, пояснюючий зміст помилки (Рис. 4.2.).

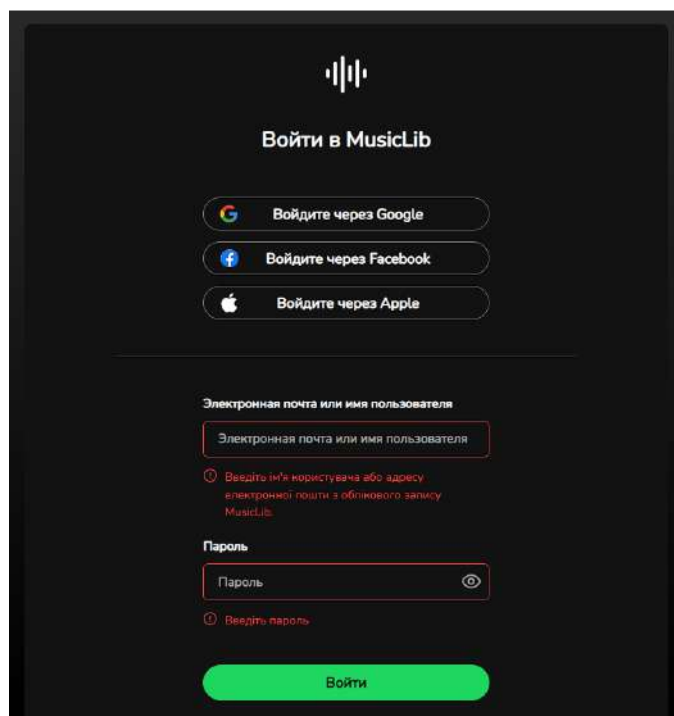

Рис. 4.2.Спроба авторизуватися без даних

Реєстрація реалізована як послідовність з трьох етапів. Кожен крок спрямований на поступове введення обов'язкових даних для створення нового облікового запису.

Перший екран форми реєстрації (Рис. 4.3.) містить поле для введення електронної пошти, кнопки авторизації через Google, Facebook та Apple, а також кнопку "Далі" для переходу до наступного кроку. Після введення пошти система перевіряє її формат. У разі помилки (Рис. 4.4.) з'являється повідомлення червоного кольору з текстом:

“Адреса електронної пошти недійсна. Переконайтеся, що вона вказана в такому форматі: example@email.com.”

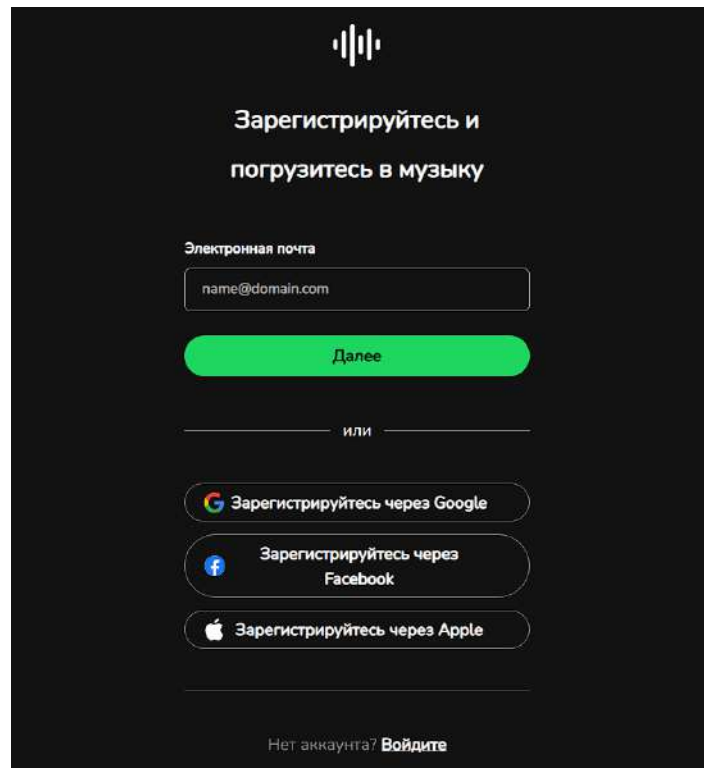

Рис. 4.3. Етап введення електронної пошти у формі реєстрації

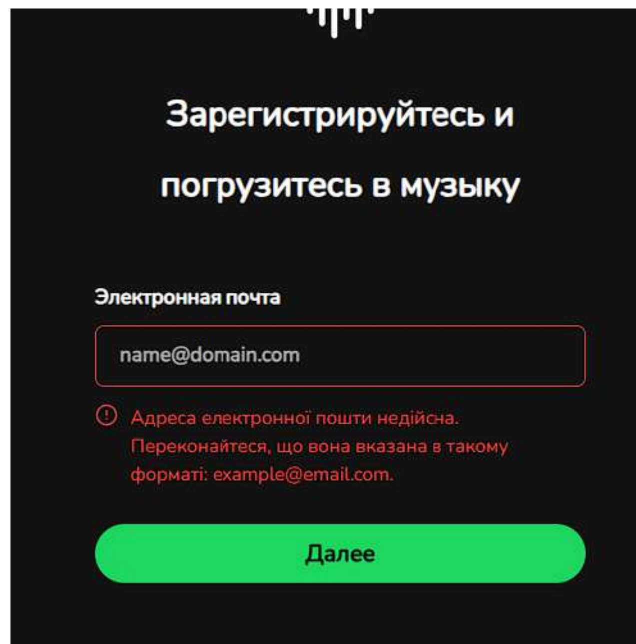

Рис. 4.4. Помилкове введення електронної пошти у формі реєстрації

Другий крок (Рис. 4.5.) передбачає створення пароля. Система в режимі реального часу перевіряє відповідність пароля встановленим вимогам: не менше 10 символів, наявність хоча б однієї цифри або спеціального символу,

а також щонайменше однієї літери (Рис. 4.6.). Для кожної з вимог відображається індикатор виконання. Якщо вимога виконана, поруч з нею з'являється зелена галочка, якщо ні — вона підсвічується червоним.

Рис. 4.5. Етап створення пароля із валідацією вимог

Рис. 4.6. Динамічна перевірка дотримання вимог

Третій етап (Рис. 4.7.) містить поля для введення імені користувача, дати народження та вибору статі. Кожне поле супроводжується відповідним заголовком. У разі помилки під полем з'являється текст червоного кольору з

поясненням, що саме потрібно виправити (Рис. 4.8.). Це дозволяє користувачу завершити реєстрацію лише після повного і коректного заповнення форми.

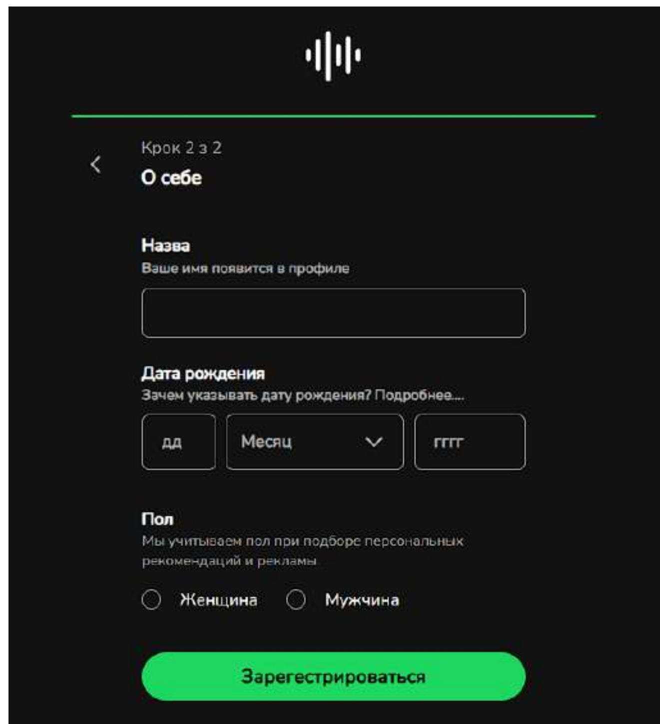

Рис. 4.7. Форма введення особистих даних

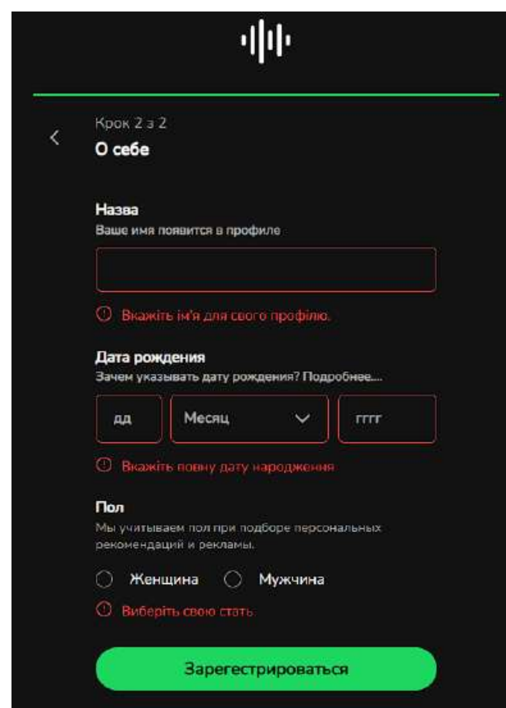

Рис. 4.8. Пимилкове введення особистих даних

Після успішної реєстрацій користувач редиректиться на сторінку авторизації. Там йому необхідно увійти у свій обліковий запис, щоб почати користуватися каталогом музики.

4.2. Каталог та рекомендації музики

На головному екрані каталогу (Рис. 4.9) розташовано кілька рекомендованих плейлистів, які формуються на основі вподобань користувача, нещодавніх прослуховувань та популярного контенту. Кожен блок рекомендацій має заголовок і складається з плиток, що представляють окремі плейлисти. Плитки містять обкладинку, назву плейлиста, а також короткий опис.

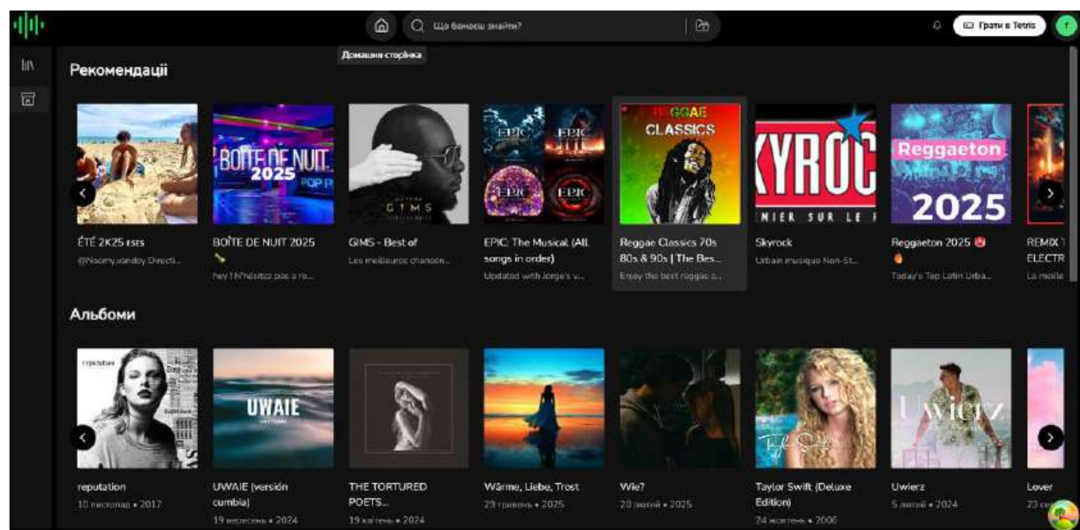

Рис. 4.9. Головна сторінка музичного каталогу

При натисканні на будь-який плейлист відкривається його повна сторінка (Рис. 4.10), де представлена велика обкладинка, список треків, кнопка відтворення, а також можливість додати плейлист до вподобаних. У правому верхньому куті сторінки присутній елемент інтерфейсу "серце", натискання на який додає плейлист до обраного користувача (Рис. 4.11.).

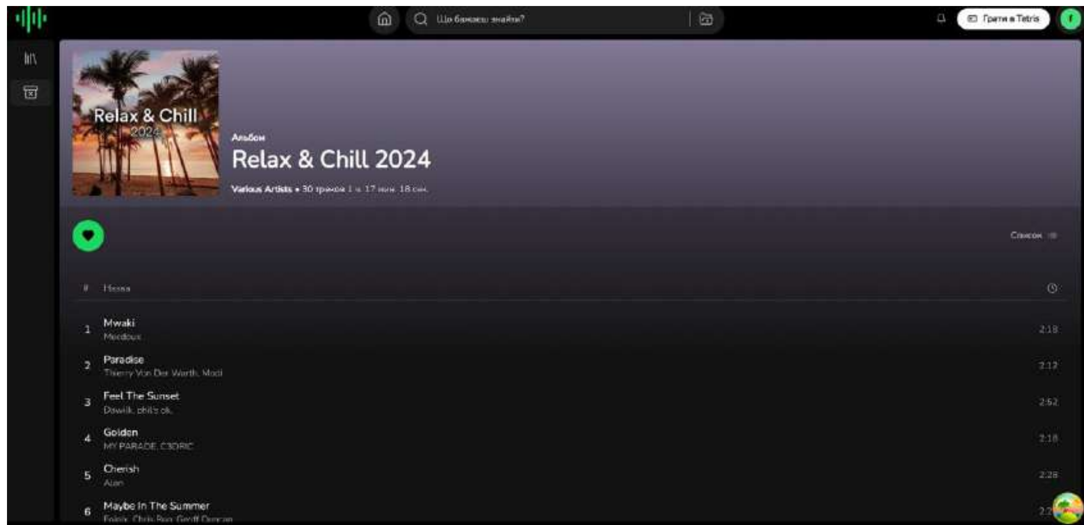

Рис. 4.10. Сторінка перегляду обраного плейлиста

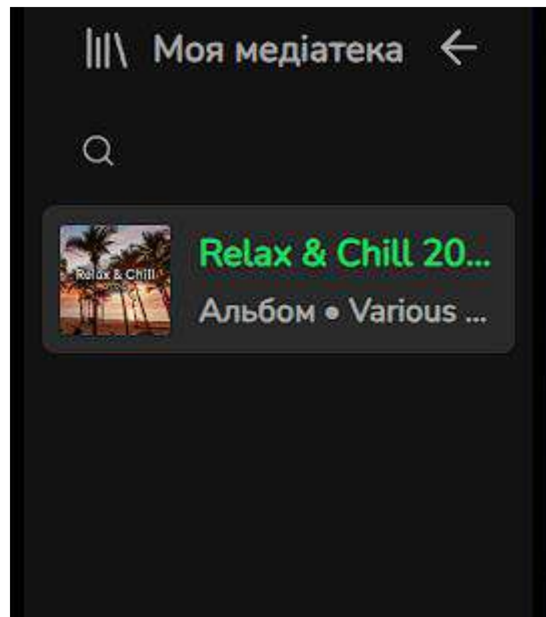

Рис. 4.11. Плейлист додався в медіатеку

Користувач також має змогу здійснювати пошук у музичному каталозі за допомогою спеціального поля пошуку, розміщеного у верхній частині інтерфейсу (рис. 4.12.). Поле підтримує динамічний пошук: результати оновлюються в реальному часі при введенні запиту. Система здійснює пошук за назвами треків, виконавців, альбомів та плейлистів.

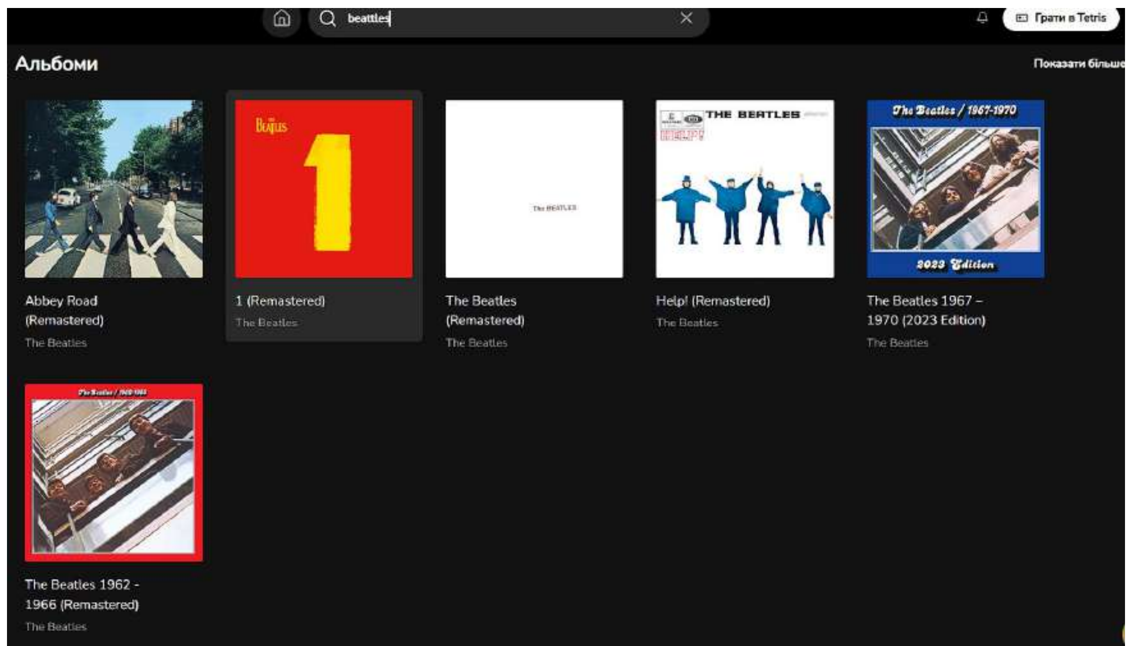

Рис. 4.12. Інтерфейс пошуку по каталогу

Для зручності користувача у верхньому правому куті інтерфейсу, поряд з аватаром або ініціалами профілю, з’являється після кліку кнопка "Вихід" (Рис. 4.13.), яка дозволяє завершити поточну сесію авторизованого користувача. Після натискання виконується вихід із системи та перенаправлення на сторінку авторизації.

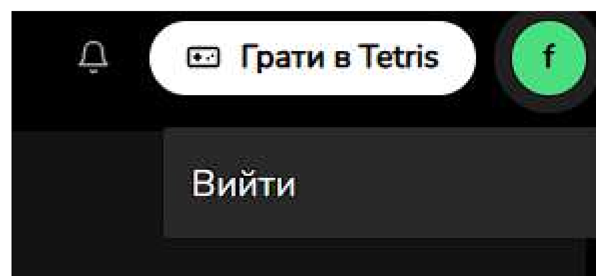

Рис. 4.13. Кнопка виходу

Уся навігація між розділами (каталог, огляд, пошук, плейлист) реалізована через бічне або верхнє меню. Інтерфейс зберігає загальну стилістику всього застосунку: темне тло, контрастні елементи керування.

У межах функціоналу рекомендацій у застосунку реалізовано адаптацію на основі дій користувача. Система аналізує активність, взаємодію з плейлистами, пошуки користувача, додавання музики до медіатеки, прослуховування та вподобання. Наприклад, щойно було додано до медіатеки плейлист із розслабляючою музикою. Тепер у блоці рекомендацій почали

з’являтися добірки з подібним настроєм — інструментальна музика, амбієнт, повільні треки з позначкою "relax" (Рис. 4.14.).

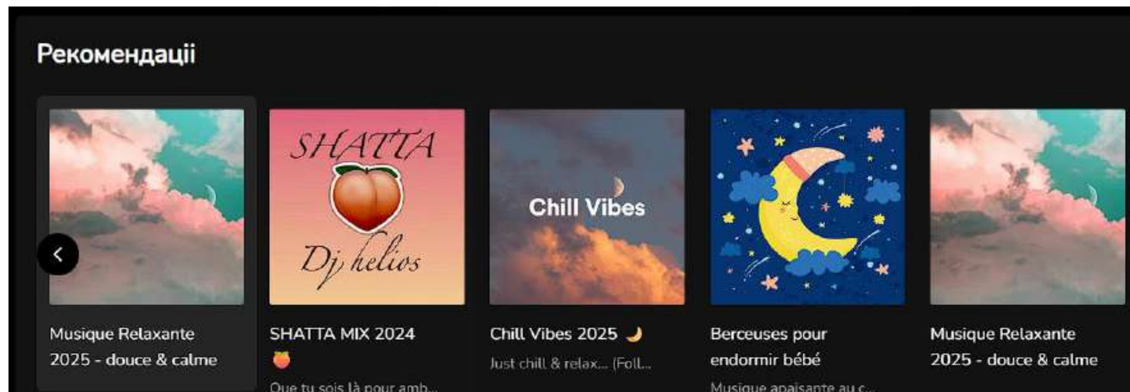

Рис. 4.14. Рекомендації сформувалися відносно дій користувача

У межах розробки застосунку також було реалізовано адаптивну верстку для коректної роботи інтерфейсу на мобільних пристроях. Усі основні функціональні блоки — каталог музики, рекомендації, перегляд плейлистів, пошук — автоматично змінюють свій вигляд залежно від ширини екрана користувача. Це забезпечує комфортне користування застосунком як на великих екранах настільних комп’ютерів, так і на смартфонах.

У мобільній версії головний екран каталогу (Рис. 4.15) перетворюється на вертикально прокручувану стрічку з плитками плейлистів, які змінюються розділ відповідно до ширини пристрою. З’являються 2 кнопки, завдяки яким можна горизонтально гортати рекомендовані плейлисти.

Сторінка перегляду плейлиста (Рис. 4.16.) адаптована для компактного розміщення обкладинки, кнопок керування та списку треків. Кнопка вподобання розташовується одразу під обкладинкою, а список треків подається у вигляді лінійного списку з інтерактивними елементами .

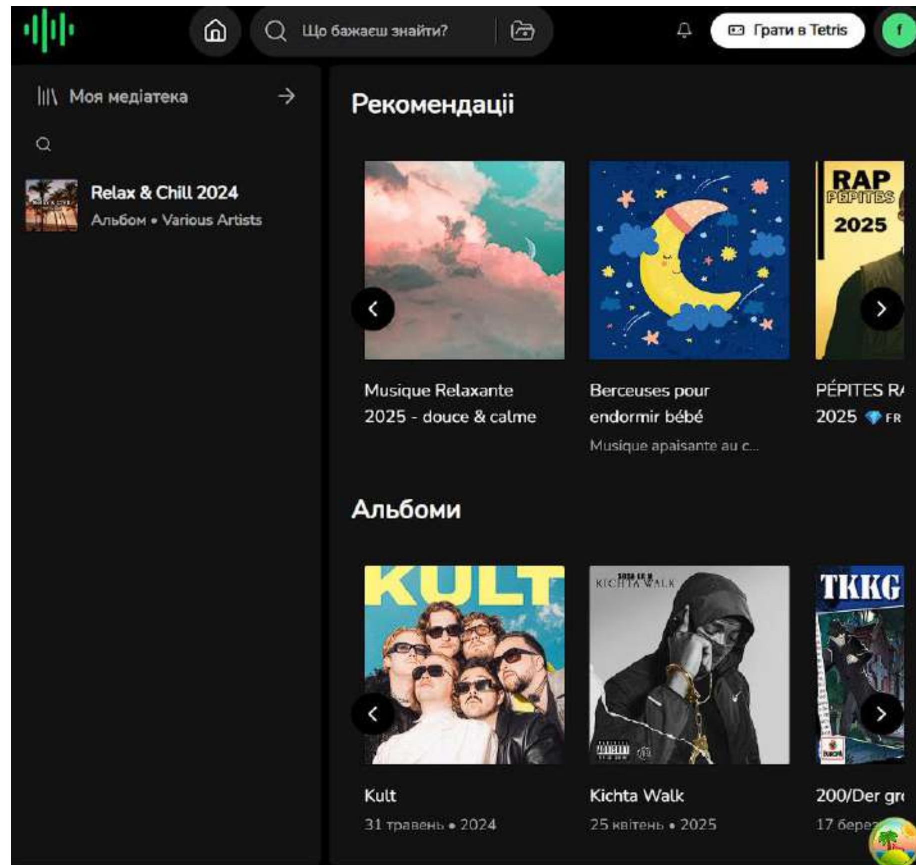

Рис. 4.15. Адаптивність каталогу

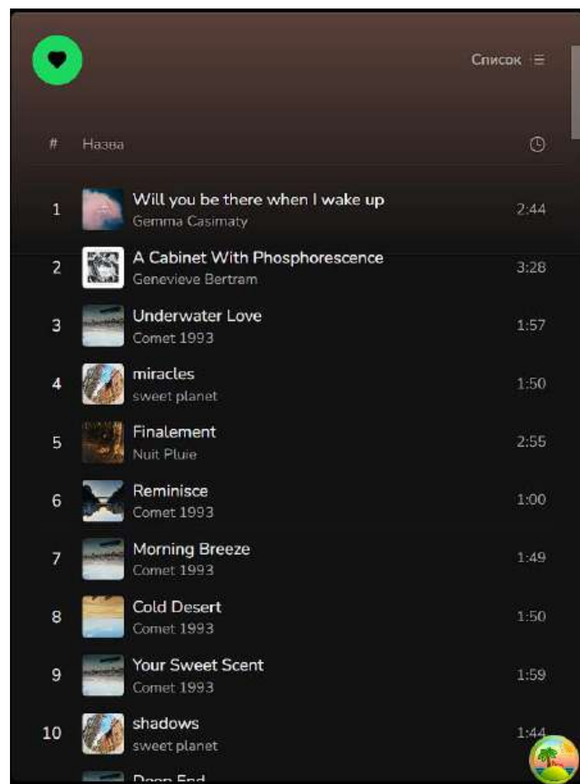

Рис. 4.16. Адаптивність плейлиста

Пошукова система в мобільній версії (Рис. 4.17.) активується через поле вводу верхній частині екрана. Результати запитів відображаються у форматі

списку з можливістю миттєвого переходу до треку, альбому чи виконавця. Кожен плейлист займає половину ширини контейнера, і таким чином утворюється сітка з двох елементів в рядку.

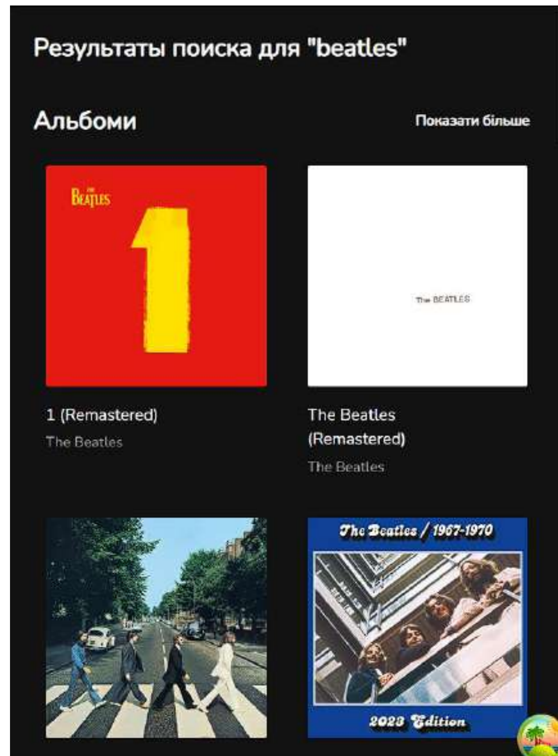

Рис. 4.17. Адаптивність результатів пошуку

Усі інтерактивні компоненти адаптуються відповідно до розміру екрана, а зайві декоративні елементи приховуються або згруповуються в меню, щоб не перевантажувати інтерфейс.

Висновки до розділу 4

У четвертому розділі було розглянуто процес розробки програмного забезпечення, реалізацію основних функціональних модулів інтерфейсу користувача. Розглянуті були компоненти, які забезпечують взаємодію з системою: авторизації, реєстрації, каталогу музики, рекомендаційного механізму та адаптації інтерфейсу до мобільних пристроїв.

ВИСНОВКИ

В результаті виконання цієї дипломної роботи було створено веб-сайт, який дає змогу зберігати, впорядковувати та прослуховувати персональну музичну колекцію. Сайт має алгоритм, який формує рекомендації на основі індивідуальних уподобань користувача.

На початку роботи, у першому розділі, було проведено детальний аналіз існуючих музичних платформ, що дозволило виявити недоліки у вже доступних сервісах, відсутність гнучких інструментів для персоналізації каталогу музики. Саме це стимулювало необхідність розробити власний застосунок, який міг би задовольнити потреби користувачів у зручному і простому доступі до їхніх музичних колекцій.

Другий розділ присвячений побудові алгоритму, що став основою для подальшої технічної реалізації. У ньому детально описано, як відбувається обробка запитів користувача, фільтрація і сортування музичних записів, а також розробка алгоритму рекомендацій, що враховує поведінку та вподобання.

Третій розділ містить опис структури бази даних, включаючи вибір системи управління та методи міграції даних. База даних була спроектована таким чином, щоб забезпечити збереження інформації про треки, плейлисти, користувачів, їхні вподобання та історію прослуховувань.

У четвертому розділі описано процес розробки програмного забезпечення, починаючи з авторизації і реєстрації, що включає інтеграцію з популярними платформами для входу через Google, Facebook і Apple. Інтерфейс розроблено з урахуванням зручності користувача: наявні поля для введення електронної пошти і пароля, повідомлення про помилки під час реєстрації, а також покрокова процедура введення даних. Каталог музики організований за зразком відомих стрімінгових сервісів, з функціями перегляду плейлистів, пошуку та додавання музичних добірок до медіатеки. Особливістю стало впровадження адаптивного дизайну, який коректно

відображається на різних пристроях — від десктопів до мобільних телефонів. Це значно розширює можливості користування застосунком у різних ситуаціях.

Ще однією важливою складовою стала система рекомендацій, яка динамічно реагує на активність користувача. Ддодавання до бібліотеки плейлисту з музикою конкретного жанру автоматично сформувало пропозиції аналогічних композицій.

Результатом розробки став комплексний веб-застосунок, який задовольняє запити різних категорій користувачів. Розроблена система забезпечує можливістьзберігати і слухати музику, отримувати якісні рекомендації, організовувати колекції і шукати композиції за запитами різних категорій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дьяків В.І. Основи розробки веб-застосунків / В.І. Дьяків. — Київ: КНУ, 2020. — 256 с.
2. Стівенсон В. Програмування на JavaScript для веб-розробників / В. Стівенсон. — Львів: Видавництво «Растр», 2019. — 432 с.
3. Microsoft Docs. ASP.NET Core Documentation [Електронний ресурс]. – Режим доступу : https://learn.microsoft.com/en-us/aspnet/core/
4. React Official Documentation [Електронний ресурс]. – Режим доступу : https://reactjs.org/docs/getting-started.html
5. PostgreSQL Documentation [Електронний ресурс]. – Режим доступу : https://www.postgresql.org/docs/
6. Джонсон М. Архітектура REST API / М. Джонсон. — Харків: Видавництво «СофтСервіс», 2021. — 178 с.
7. Смирнов А.Р. Бази даних: проектування та реалізація / А.Р. Смирнов. — Одеса: ОНУ, 2018. — 304 с.
8. Котляр І.В. Веб-технології та розробка інтерфейсів / І.В. Котляр. — Київ: Наукова думка, 2022. — 212 с.
9. YouTube Developer Guide [Електронний ресурс]. – Режим доступу : https://developers.google.com/youtube
10. Офіційний сайт Spotify [Електронний ресурс]. – Режим доступу : https://www.spotify.com
11. Гринько О.І. Системи управління базами даних / О.І. Гринько. — Львів: Видавництво ЛНУ, 2017. — 289 с.
12. Мельник П.О. Розробка веб-додатків на React / П.О. Мельник. — Харків: Фоліо, 2021. — 345 с.
13. W3Schools. HTML, CSS, JavaScript Reference [Електронний ресурс]. – Режим доступу : https://www.w3schools.com/
14. Щербина В.В. Моделювання даних у СУБД / В.В. Щербина. — Київ: КНТ, 2019. — 276 с.

15. Ткаченко С.П. Інтернет-технології та веб-розробка / С.П. Ткаченко. — Одеса: ОНУ, 2020. — 314 с.

16. Node.js Documentation [Електронний ресурс]. – Режим доступу : https://nodejs.org/en/docs/

17. Winston L. Веб-безпека для розробників / Л. Вінстон. — Київ: Видавництво IT, 2022. — 198 с.

18. Офіційний сайт Google Developers [Електронний ресурс]. – Режим доступу : https://developers.google.com/

19. GitHub Docs [Електронний ресурс]. – Режим доступу : https://docs.github.com/en

20. Wikipedia [Електронний ресурс]. – Режим доступу : https://uk.wikipedia.org/

ДОДАТКИ

Додаток А

Auth

```
import Divider from "@/components/ui/divider/divider";
import { Form } from "@/components/ui/form/form";
import FormButton from "@/components/ui/form/from-button";
import Input from "@/components/ui/form/input";
import PasswordInput from "@/components/ui/form/password-input";
import { labels } from "@/constants/labels.constant";
import { loginFormSchema } from "@/lib/auth";
import { FirebaseError } from "firebase/app";
import { CircleAlert } from "lucide-react";
import { useState } from "react";
import { useAuth } from "../context/useAuth";
import AuthWithLabels from "./ui/auth-with-lables";
import ChangeForm from "./ui/change-form";
import Title from "./ui/title";

function Notification() {
  return (
    <div className="mb-6 flex h-12 items-center gap-2 rounded-md bg-red-600 px-4 text-
white mdmobile:w-[90%]">
      <CircleAlert height={16} width={16} />
      <span>Неправильное имя пользователя или пароль.</span>
    </div>
  );
}

function LoginForm() {
  const [isNotFound, setIsError] = useState(false);
  const [isLoading, setIsLoading] = useState(false);

  const user = useAuth();
  async function handleSubmit({
    email,
    password,
  }: {
    email: string;
    password: string;
  }) {
    setIsLoading(true);
    try {
      //await doSignUserWithEmailAndPassword({ email, password })

      await fetch("/login?useCookies=true", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
```

```
          email: email,
          password: password,
        }),
      })
        .then((data) => {
          // handle success or error from the server
          if (data.ok) {
            user.setCurrentUser({ email, displayName: email });
            return;
          }
          setIsError(true);
        })
        .catch((error) => {
          // handle network error
          console.error(error);
        });
    } catch (error) {
      console.error(error);

      if (error instanceof FirebaseError) {
        if (
          error.code === "auth/user-not-found" ||
          error.code === "auth/wrong-password"
        ) {
          setIsError(true);
        }
      }
    } finally {
      setIsLoading(false);
    }
  }

  return (
    <>
      <Title>Войти в MusicLib</Title>
      {isNotFound && <Notification />}
      <AuthWithLabels
        className="w-full mdmobile:max-w-80"
        labels={labels}
        type="login"
      />
      <Divider className="my-10 w-[75%] opacity-50" />
      <Form
        onSubmit={handleSubmit}
        schema={loginFormSchema}
        options={{ mode: "onChange" }}
        className="flex w-full items-center justify-center mdmobile:px-6"
      >
        {({ formState, register }) => {
          return (
            <div className="grid w-full gap-4 mdmobile:max-w-80">
```

```
          <Input
            label="Электронная почта или имя пользователя"
            placeholder="Электронная почта или имя пользователя"
            registration={register("email")}
            errors={formState.errors["email"]}
          />
          <PasswordInput
            placeholder="Пароль"
            errors={formState.errors["password"]}
            registration={register("password")}
          />
          <FormButton isLoading={isLoading}>Войти</FormButton>
        </div>
      );
    }}
  </Form>
  <Divider className="my-10 w-[75%] opacity-50" />
  <ChangeForm to={"/auth/register"} type="login">
    Регистрация в MusicLib
  </ChangeForm>
 </>
 );
}

export default LoginForm;

import { Form } from "@/components/ui/form/form"
import FormButton from "@/components/ui/form/from-button"
import PasswordInput from "@/components/ui/form/password-input"
import PasswordRequirements from "@/components/ui/form/password-requirements"
import { useFormNextStep } from "@/components/ui/form/useFormNextStep.hook"
import getFormData from "@/components/ui/stepper/services/getFormData"
import setFormData from "@/components/ui/stepper/services/setFormData"
import { registerInputSchema } from "@/lib/auth"

const passwordSchema = registerInputSchema.pick({ password: true })

const PasswordStep = () => {
const { handleNextStep } = useFormNextStep()
const formData = getFormData()

return (
	<Form
		onSubmit={handleNextStep}
		schema={passwordSchema}
		className="mobile:max-w-[25em] w-full mobile:px-10  mt-8 mx-auto pb-
5"
		options={{
			defaultValues: { password: formData.password },
			mode: "onChange",
		}}
```

```
>
        {({ formState, register, watch }) => {
                setFormData({ ...formData, ...watch() })

                const inputValue = watch("password")

                return (
                        <>
                                <PasswordInput
                                        errors={formState.errors["password"]}
                                        registration={register("password")}
                                        showError={false}
                                />
                                <PasswordRequirements
                                        isInputEmpty={!inputValue}
                                        error={formState.errors["password"]}
                                />
                                <FormButton> Далее </FormButton>
                        </>
                )
        }}
</Form>
)
}

export default PasswordStep
import { Navigate } from "react-router-dom";
import { useAuth } from "../context/useAuth";
import Loader from "@/components/ui/loader/loader";

export const ProtectedRoute = ({ children }: { children: React.ReactNode }) => {
  const { isLoading, currentUser } = useAuth();

  return (
    <Loader isLoading={isLoading}>
      {currentUser ? children : <Navigate to={"/auth/login"} replace />}
    </Loader>
  );
};

import { type FC, type ReactElement } from "react";
import { useAuth } from "../context/useAuth";
import { Navigate, useLocation } from "react-router-dom";
import type { TUseFormNextStepState } from "@/components/ui/form/useFormNextStep.hook";

const RedirectIfAuthenticated: FC<{ children: ReactElement }> = ({
  children,
}) => {
  const { state }: TUseFormNextStepState = useLocation();
  const { currentUser } = useAuth();
```

```
  if (state?.authWithService) return children;

  return !currentUser ? children : <Navigate to={"/app"} />;
};

export default RedirectIfAuthenticated;

import Divider from "@/components/ui/divider/divider";
import { Form } from "@/components/ui/form/form";
import FormStepper from "@/components/ui/form/form-stepper";
import FormButton from "@/components/ui/form/from-button";
import Input from "@/components/ui/form/input";

import { labels } from "@/constants/labels.constant";

import { Link, useLocation } from "react-router-dom";

import AboutStep from "./about-step";
import PasswordStep from "./password-step";
import AuthWithLabels from "./ui/auth-with-lables";

import {
  useFormNextStep,
  type TUseFormNextStepState,
} from "@/components/ui/form/useFormNextStep.hook";
import getFormData from "@/components/ui/stepper/services/getFormData";
import setFormData from "@/components/ui/stepper/services/setFormData";

import { registerInputSchema } from "@/lib/auth";
import Title from "./ui/title";
import ChangeForm from "./ui/change-form";

const tabs = [
  {
    title: "Придумай пароль",
    component: <PasswordStep />,
  },
  {
    title: "О себе",
    component: <AboutStep />,
  },
];

const InitialSchema = registerInputSchema.pick({ email: true });

const InitialRegisterFormScreen = () => {
  const { handleNextStep } = useFormNextStep();
  const { email } = getFormData();

  return (
```

```
  <div className="mobile:max-w-[25em] w-full mobile:px-8 px-4 bg-primaryBg pb-5">
    <Title>Зарегистрируйтесь и погрузитесь в музыку</Title>
    <Form
      onSubmit={handleNextStep}
      schema={InitialSchema}
      options={{ defaultValues: { email }, mode: "onChange" }}
    >
      {({ formState, register, watch }) => {
        setFormData({ ...getFormData(), ...watch() });

        return (
          <>
            <Input
              registration={register("email")}
              label="Электронная почта"
              placeholder="name@domain.com"
              errors={formState.errors["email"]}
            />
            <FormButton>Далее</FormButton>
          </>
        );
      }}
    </Form>

    <Divider className="my-10">или</Divider>
    <AuthWithLabels labels={labels} type="register" />
    <Divider className="my-10 opacity-30" />

    <ChangeForm to={"/auth/login"} type="login">
      Войдите
    </ChangeForm>
  </div>
 );
};

function RegisterForm() {
 const { state }: TUseFormNextStepState = useLocation();

 return state !== null ? (
  <div className="max-mobile:px-4  max-w-[436px] w-full bg-primaryBg  ">
    <FormStepper tabs={tabs} />
  </div>
 ) : (
  <InitialRegisterFormScreen />
 );
}

export default RegisterForm;
```

Додаток Б

Album

```
import type {
TrackItemType,
TracksType,
} from "@/features/playlists/api/get-playlists"
import { api } from "@/lib/api-client"
import type { QueryConfig } from "@/lib/client-query"
import type { ArtistType } from "@/utils/tracks-utils"
import { queryOptions, useQuery } from "@tanstack/react-query"

type AlbumTracksType = Omit<TracksType, "items"> & {
items: TrackItemType[]
}

export type AlbumCopyType = {
text: string
type: string
}

export type GetAlbumReturn = {
album_type: "album" | "single" | "compilation"
total_tracks: number
id: string
images: { url: string }[]
name: string
release_date: string
type: "album"
artists: ArtistType[]
tracks: AlbumTracksType
popularity: number
copyrights: AlbumCopyType[]
}

export const getAlbum = (albumId: string): Promise<GetAlbumReturn> => {
return api.get(`/albums/${albumId}`).then(res => res.data)
}

export const getAlbumQueryOptions = (albumId: string) => {
return queryOptions({
        queryKey: ["album", albumId],
        queryFn: () => getAlbum(albumId),
})
}
type UsePlaylistOptions = {
albumId: string
queryConfig?: QueryConfig<typeof getAlbumQueryOptions>
}
```

Продовження Додатку Б

```
export const useAlbum = ({ albumId = "", queryConfig }: UsePlaylistOptions) => {
return useQuery({
	...getAlbumQueryOptions(albumId),
	...queryConfig,
})
}

import { api } from "@/lib/api-client"
import type { QueryConfig } from "@/lib/client-query"
import { queryOptions, useQuery } from "@tanstack/react-query"
import type { GetAlbumReturn } from "./get-album"

type GetUserAlbumsParams = {
userID: string | undefined
limit?: number
offset?: number
}

export const getUserAlbums = ({
userID,
limit,
offset,
}: GetUserAlbumsParams): Promise<GetAlbumReturn[]> => {
return api
	.get(`/artists/${userID}/albums`, {
		params: {
			limit,
			offset,
		},
	})
	.then(res => res.data)
}

export const getUserAlbumsQueryOptions = ({
userID,
limit,
offset,
}: GetUserAlbumsParams) => {
return queryOptions({
	queryKey: ["artists", userID, "albums"],
	queryFn: () => getUserAlbums({ userID, limit, offset }),
})
}
type UseUserAlbumsOptions = GetUserAlbumsParams & {
queryConfig?: QueryConfig<typeof getUserAlbumsQueryOptions>
}

export const useUserAlbums = ({
userID = "",
limit = 10,
offset = 0,
```

```
queryConfig,
}: UseUserAlbumsOptions) => {
return useQuery({
	...getUserAlbumsQueryOptions({ userID, limit, offset }),
	...queryConfig,
})
}

import { api } from "@/lib/api-client"
import type { QueryConfig } from "@/lib/client-query"
import { queryOptions, useQuery } from "@tanstack/react-query"
import type { GetAlbumReturn } from "./get-album"

export const getAlbums = async (ids: string): Promise<GetAlbumReturn[]> => {
const { data } = await api.get(`/albums?ids=${ids}`)
return data.albums
}

export const getAlbumsQueryOptions = (albumsIds: string[]) => {
const ids = albumsIds.toString()

return queryOptions({
	queryKey: ["albums"],
	queryFn: () => getAlbums(ids),
	initialData: [],
	refetchInterval: 1000 * 60 * 60 * 24,
})
}
type UseAlbumsOptions = {
albumIds: string[]
queryConfig?: QueryConfig<typeof getAlbumsQueryOptions>
}

export const useAlbums = ({ albumIds, queryConfig }: UseAlbumsOptions) => {
return useQuery({
	...getAlbumsQueryOptions(albumIds),
	...queryConfig,
})
}

import { FULL_MONTH } from "@/features/playlists/contants";
import type { AlbumCopyType } from "../api/get-album";

function AlbumCopy({
  releaseDate,
  copyrights,
}: {
  releaseDate: string | undefined;
  copyrights: AlbumCopyType[] | undefined;
}) {
  const [year, month, day] = releaseDate?.split("-") || [];
```

```
  return (
    <>
      <div />
      <p className="text-sm text-textButton">
        {day} {FULL_MONTH[+month]} {year} г.
      </p>
      <div></div>
      {copyrights &&
        copyrights.map((r) => (
          <p key={r.text + r.type} className="text-[0.68rem] text-textButton">
            {r.text}
          </p>
        ))}
    </>
  );
}

export default AlbumCopy;


import { Button } from "@/components/ui/button";
import TracksTable from "@/features/playlists/compontents/tracks-table";
import { convertTime } from "@/utils/convert-time";
import { joinArtists } from "@/utils/tracks-utils";
import { Clock, Play } from "lucide-react";
import { useAlbum } from "../api/get-album";
import { useNavigate } from "react-router-dom";

function AlbumsTracks({ id }: { id: string }) {
  const { data, isLoading } = useAlbum({
    albumId: id,
  });

  const navigation = useNavigate();

  return (
    <div className="relative z-20 p-[var(--content-spacing)]" key={id}>
      <TracksTable
        isLoading={isLoading}
        data={data?.tracks.items}
        grid={"compact"}
        ids={data?.tracks.items.map((item) => item.id)}
        columns={[
          {
            title: "#",
            field: "#",
            Cell: ({ index }) => {
              return (
                <>
                  <span className="pointer-events-none absolute right-1 top-[50%] translate-y-
[-50%] group-hover:hidden">
```

```
              {index + 1}
            </span>
            <Button
              className="hidden group-hover:block"
              variant={"iconTransparent"}
              size={"icon"}
            >
              <Play
                width={"16"}
                height={"16"}
                fill="white"
                stroke="white"
              />
            </Button>
          </>
        );
      },
    },
    {
      title: "Назва",
      field: "track",
      Cell: ({ entry: { name, artists } }) => {
        return (
          <div className="flex items-center gap-2">
            <div>
              <div className="text-ellipsis-custom leading-6">{name}</div>
              <button
                className="text-ellipsis-custom text-sm text-textButton hover:underline
group-hover:text-white"
                onClick={() => {
                  navigation(`/app/artist/${artists[0].id}`);
                }}
              >
                {joinArtists(artists)}
              </button>
            </div>
          </div>
        );
      },
    },
    {
      title: <Clock className="h-4 w-4 text-textButton" />,
      field: "duration_ms",
      Cell: ({ entry: { duration_ms } }) => {
        const [_, m, s] = convertTime(duration_ms);

        return (
          <span className="text-sm text-textButton">
            {m}:{s.toString().padStart(2, "0")}
          </span>
        );
```

```
          },
        },
      ]}
    />
  </div>
 );
}

export default AlbumsTracks;

import CardItem from "@/components/ui/card-item/card-item";
import { ContentNotFound } from "@/features/playlists/compontents/content-not-found";
import PlaylistControl from "@/features/playlists/compontents/playlist-control";
import PlaylistHeader from "@/features/playlists/compontents/playlist-header";
import { useColor } from "@/hooks/use-color";
import { cn } from "@/utils/cn";
import { getTrackDurationForAlbum, joinArtists } from "@/utils/tracks-utils";
import { useNavigate, useParams } from "react-router-dom";
import { useAlbum } from "../api/get-album";
import { useUserAlbums } from "../api/get-user-albums";
import AlbumCopy from "./album-copy";
import AlbumsTracks from "./album-tracks";

import { useRecentlyListened } from "@/utils/recently-listened-context";

function AlbumView() {
  const { albumId } = useParams()!;
  const navigate = useNavigate();
  const {
    isLoading,
    data: albumData,
    isError,
  } = useAlbum({ albumId: albumId as string });

  const { addToRecentlyListened } = useRecentlyListened();
  const topFiveAlbums = useUserAlbums({
    userID: albumData?.artists[0].id,
  });

  const bgColor = useColor(albumData?.images[0]?.url);

  const tracksTotalDurationMs = getTrackDurationForAlbum(
    albumData?.tracks.items,
  );

  if (isError) {
    return <ContentNotFound />;
  }

  return (
    <section className="relative @container">
```

```
<PlaylistHeader
  isLoading={isLoading}
  data={{
    label: "Альбом",
    image: albumData?.images[0]?.url,
    name: albumData?.name,
    description: "",
    display_name: joinArtists(albumData?.artists),
    total: albumData?.tracks.total,
    totalDuration: tracksTotalDurationMs,
  }}
/>
<div className="relative">
  <div
    className="tracks-bg absolute left-0 top-0 h-[14.5rem] w-full"
    style={{ backgroundColor: bgColor }}
  ></div>
  <PlaylistControl
    itemType="album"
    onSave={() => {
      addToRecentlyListened(albumId as string, "album");
    }}
  />
  <AlbumsTracks id={albumId!} />
</div>
<div className="z-10 p-[var(--content-spacing)]">
  <AlbumCopy
    releaseDate={albumData?.release_date}
    copyrights={albumData?.copyrights}
  />
  <div className="mt-14 @container">
    <h3 className="text-2xl font-bold hover:underline">
      {albumData?.artists[0].name}: інші Альбоми
    </h3>
    <div
      className={cn(
        `mt-6 grid grid-cols-[repeat(2,_minmax(7rem,_15rem))] grid-rows-[auto] @[530px]:grid-cols-[repeat(3,_minmax(7rem,_15rem))] @[720px]:grid-cols-[repeat(4,_minmax(7rem,_15rem))] @[905px]:grid-cols-[repeat(5,_minmax(7rem,_15rem))] @[1095px]:grid-cols-[repeat(6,_minmax(7rem,_15rem))] @[1285px]:grid-cols-[repeat(8,_minmax(7rem,_15rem))] @[1420px]:grid-cols-[repeat(10,_minmax(7rem,_15rem))]`,
      )}
    >
      {// @ts-expect-error  @ts-ignore (will be fixed in the next step)
      topFiveAlbums.data?.items.map((item) => (
        <CardItem
          key={item.id}
          onCLick={() => {
            navigate(`../album/${item.id}`);
```

```
              }}
              item={{
                imageSrc: item.images.reverse()[0]?.url,
                name: item.name,
                subText: `${item.release_date.split("-")[0]} • ${item.album_type}`,
              }}
            />
          ))}
        </div>
      </div>
    </div>
  </section>
 );
}

export default AlbumView;
```