Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

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

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБЛЕННЯ ЗАСТОСУНКУ ДЛЯ ІНТЕГРАЦІЇ СЕРВІСІВ ВІДТВОРЕННЯ ТА ОРГАНІЗАЦІЇ АУДІОФАЙЛІВ
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-19-1
Босенко А.М.
(прізвище, ініціали)

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

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Тітова О.В.
(посада, прізвище, ініціали)

Допускається до захисту

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

2023 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

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

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Босенку Артему Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення застосунку для інтеграції сервісів відтворення та організації аудіофайлів

затверджена наказом університету від 15 травня 2023 року № 474 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 22 травня 2023 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій та форумів, дані інтернет-мережі, документація використаних інструментів.

4. Перелік питань, що потрібно опрацювати в роботі

1. Огляд підходів для організації архітектури сучасних вебзастосунків.

2. Вивчення моделей «Плейлист» та «Пісня» які надані застосунками для організації та відтворення аудіофайлів та інтеграція використовуючи API застосунків.

3. Організація компонентів які будуть реалізовані в інтерфейсі користувача за допомогою React.

4. Організація взаємодії між інтерфейсом користувача та серверною частиною застосунку.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми обробки зображень, постановка задачі, тестові зображення.

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

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Консультант з дотримання діючих стандартів та норм	Доцент Творошенко І.С.		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	10.04.2023	
2	Аналіз завдання, підбір літератури	11.04.23-17.04.23	
3	Аналіз літератури з досліджуваної проблеми	18.04.23-20.04.23	
4	Аналіз засобів для розробки вебзастосунків	21.04.23-30.04.23	
5	Аналіз документації застосунків	01.05.23-14.05.23	
6	Програмна реалізація	15.05.23-23.05.23	
7	Оформлення пояснювальної записки	24.05.23-26.05.23	
8	Перевірка на плагіат	27.05.23	
9	Рецензування	28.05.23	
10	Підготовка презентації та доповіді	29.05.23-30.05.23	
11	Занесення роботи в електронний архів	31.05.23	
12	Попередній захист кваліфікаційної роботи	31.05.23	

Дата видачі завдання 10 квітня 2023 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Тітова О.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 65 с., 2 табл., 36 рис., 32 джерела.

МІГРАЦІЯ ПЛЕЙЛИСТІВ, REACT, TYPESCRIPT, JAVA, ІНТЕГРАЦІЯ З СЕРВІСАМИ ДЛЯ ВІДТВОРЕННЯ АУДІОФАЙЛІВ.

Об’єктом роботи є міграція плейлистів користувача між застосунками для відтворення аудіофайлів.

Метою роботи є розробка застосунку для зручної міграції плейлистів за допомогою інтеграції з сервісами для відтворення аудіофайлів.

У ході роботи проводиться аналітичний огляд існуючих застосунків для міграції плейлистів між сервісами для відтворення аудіофайлів, а також аналіз засобів для створення застосунків з використанням фреймворку React та мов програмування TypeScript та Java.

У результаті роботи здійснена програмна реалізація вебзастосунку для зручної та безкоштовної міграції плейлистів між сервісами для відтворення аудіофайлів.

PLAYLIST MIGRATIOM, REACT, TYPESCRIPT, JAVA, INTEGRATION WITH AUDIO STREAMING AND MEDIA SERVICES.

The object of the work is the migration of user playlists between audio streaming and media services.

The purpose of the work is development of application for comfort migration playlists using integration with audio streaming and media services.

The study provides an analytical review of existing applications for migrating playlists between audio streaming and media services, as well as analysis of tools for creating applications using the React framework and the TypeScript and Java programming languages.

As a result of the work, a software implementation of a web application for comfort and free migration of playlists between audio streaming and media services.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Плейлист – згрупований список збережених аудіофайлів з унікальним іменем

Міграція – процес переміщення/оновлення даних

Ендпоінт – публічна вебадреса із прихованою логікою, яку надає сервіс для взаємодії з іншими сервісами та/або користувачами

Стрімінговий сервіс – застосунок для відтворення медіа файлів

UI – User Interface (інтерфейс користувача)

Back end – частина проєкту з внутрішньою логікою інтеграції з API та базами даних

Front end – візуальна частина проєкту

ВСТУП

Загалом, міграція плейлистів з одного сервісу до іншого може бути трохи складним процесом, але в сучасному світі існують кілька ефективних і простих методів для його здійснення. Незалежно від того, чи необхідно перейти від Spotify до YouTube Music, або з іншого сервісу на Spotify, не доведеться втрачати всі свої улюблені плейлисти та треки.

Найпростіший спосіб перенесення плейлистів з одного сервісу на інший полягає в тому, щоб створити нові плейлисти на новому сервісі та додати до них всі необхідні треки. Хоча цей метод може зайняти багато часу, він дозволяє зберегти плейлисти та треки без необхідності користуватися додатковими інструментами.

Іншим способом є використання сторонніх сервісів для міграції плейлистів, таких як SongShift, TuneMyMusic, Soundiiz та інші. Ці сервіси дозволяють переносити плейлисти між різними сервісами за допомогою API інтеграції. Для цього необхідно зареєструватися на цих сервісах, підключити облікові записи Spotify та YouTube Music, вибрати плейлисти, які необхідно перенести, та запустити процес міграції. Єдиною перепоною на шляху використання цих сервісів є плата за окремий обсяг транспортування музичного контенту. Передбачається, що застосунок, який розроблено у роботі, буде дозволяти переносити аудіофайли та плейлисти безкоштовно.

Крім того, деякі платформи, такі як Spotify, мають вбудовані інструменти для міграції плейлистів. Наприклад, в Spotify є можливість експортувати плейлисти у форматі CSV та імпортувати їх в YouTube Music за допомогою Google Play Музики. Щоб скористатися цією функцією, є можливість перейти до вкладки «Файли» у своєму профілі Spotify, натиснути на «Експортувати», вибрати плейлисти, які необхідно перенести, та зберегти файл CSV [1]. Потім можна імпортувати цей файл в Google Play Музики та додати плейлисти до YouTube Music. Але це задовольняє потреби зручності для користувача, який не має навичок роботи з файловою системою.

Незалежно від того, який метод міграції обрано, важливо знати, що деякі метадані, такі як опис плейлиста та картинка обкладинки, можуть не бути перенесені разом з плейлистом. Тому, якщо необхідно зберегти всю інформацію про плейлисти, треба переносити їх за допомогою сторонніх сервісів, які дозволяють зберігати всю метадані.

Окрім збереження музичного вмісту, перенесення плейлистів з одного сервісу на інший може також знайти певні переваги. Наприклад, якщо відбувається перехід від Spotify до YouTube Music, є можливість скористатися деякими функціями, які не були доступні раніше, такими як відеокліпи та живі виступи. Крім того, перенесення плейлистів з одного сервісу на інший може допомогти знайти нові треки та виконавців, які можуть бути доступні на новому сервісі.

У застосунку пропонується актуальний підхід до розробки інтерфейсу користувача, наприклад, інтуїтивно зрозумілий покроковий процес супроводу при міграції плейлистів. Такий підхід можна втілити за допомогою фреймворку React та мови програмування Typescript. Напрямком майбутнього вдосконалення застосунка є можливість відтворювати аудіофайли прямо на сайті. Для YouTube Music передбачається можливість відтворювати відеокліпи.

1 АНАЛІЗ ОСНОВНИХ МЕТОДІВ ДЛЯ МІГРАЦІЇ АУДІОФАЙЛІВ МІЖ СЕРВІСАМИ ТА ПОРІВНЯННЯ ІСНУЮЧИХ ЗАСТОСУНКІХ ДЛЯ МІГРАЦІЇ АУДІОФАЙЛІВ

1.1 Огляд застосунків для міграції аудіофайлів та плейлистів між платформами для прослуховування аудіофайлів

У сучасному світі існує велика кількість застосунків для зручного прослуховування аудіофайлів в онлайн та офлайн режимах. Серед лідерів по стримінговим застосункам є Spotify, Apple Music та YouTube Music.

Кожен з цих сервісів має недоліки та переваги один перед одним. Наприклад, Spotify є найбільш популярним серед користувачів подібних застосунків, загальна їх кількість у Spotify – 195 мільйонів. Але водночас має одну з найвищих вартостей передплати для прослуховування музики без реклами та відсутня висока якість звуку в застосунку. В свою чергу Apple Music має одну з найбільших бібліотек з аудіофайлами і, в порівнянні із Spotify, нижчу вартість передплати. З недоліків, які відштовхують користувачів, можна назвати відсутність можливості безкоштовного використання застосунку (навіть з рекламою) та те, що Apple Music доступний лише на платформі IOS.

При необхідності змінити один сервіс для прослуховування аудіофайлів на інший постає проблема переносу вже створених та збережених аудіофайлів, які об’єднані в плейлисти [2]. Для вирішення цієї проблеми вже створені застосунки для міграції аудіофайлів та плейлистів між платформами для прослуховування аудіофайлів в онлайн та офлайн режимах.

В цьому розділі буде проведено огляд та порівняльний аналіз на такі застосунки як Soundiiz, TuneMyMusic та SongShift.

1.1.1 Soundiiz

Soundiiz – це онлайн-сервіс, який дозволяє користувачам переносити музику та плейлисти між різними сервісами стрімінгу. За допомогою Soundiiz користувачі можуть імпортувати та експортувати свою музику з більш ніж 40 сервісів, серед основних є: Spotify, Apple Music, Tidal, Google Play Music, YouTube [3].

Soundiiz був створений французьким розробником та музикантом Гійомом Фрондель (Guillaume Frondel) у 2013 році. Гійом створив Soundiiz після того, як він сам стикнувся з проблемою міграції своєї музики з одного сервісу на інший. Він був розчарований тим, що йому довелося вручну переносити свою музику та плейлисти, тому він вирішив створити засіб, який допоможе користувачам цього уникнути [4].

У 2015 році Soundiiz був запущений в якості платформи з розширенням, що дозволяє розробникам інтегрувати свої сервіси з Soundiiz. Зараз Soundiiz є одним з найбільш популярних сервісів для міграції музики та плейлистів між різними сервісами стрімінгу.

Soundiiz пропонує безкоштовний версію та версію із платною передплатою, які дають користувачам доступ до різних функцій, таких як імпорт та експорт музики та плейлистів у великій кількості форматів, автоматичний перенос пісень та багато інших.

1.1.2 TuneMyMusic

TuneMyMusic – це онлайн-сервіс для міграції музики та плейлистів між різними музичними сервісами. За допомогою TuneMyMusic можна переносити музику та плейлисти між Spotify, Apple Music, YouTube, Google Play Music та іншими сервісами [5]. Крім того є можливості:

– конвертування аудіофайлів: TuneMyMusic може конвертувати аудіофайли з одного формату в інший. Підтримувані формати включають MP3, FLAC, WAV, AAC, M4A та інші;

– об’єднання плейлистів: сервіс дозволяє об’єднувати плейлисти з різних джерел, наприклад, з Spotify та YouTube, в один плейлист;

– клонування плейлистів: TuneMyMusic може створити копію плейлисту в тому ж або іншому сервісі. Це корисна функція для тих, хто хоче зберегти оригінальний плейлист, але також хоче мати його копію в іншому сервісі;

– видалення дублікатів: TuneMyMusic допомагає знайти та видалити дублікати пісень у плейлистах;

– редагування плейлистів: сервіс дозволяє змінювати порядок пісень, додавати нові або видаляти старі пісні, змінювати назву та опис плейлисту;

– переміщення пісень між плейлистами: TuneMyMusic дозволяє переміщувати пісні між різними плейлистами в межах одного сервісу або між різними сервісами.

Засновниками застосунку TuneMyMusic є Кевін Мартін та Олександр Руденко. Кевін Мартін мав проблему із перенесенням своїх плейлистів між різними музичними сервісами, тому він вирішив створити сервіс, який допоможе іншим користувачам робити це швидко та легко. Олександр Руденко приєднався до проекту як технічний спеціаліст, і разом вони розробили перший прототип TuneMyMusic.

1.1.3 SongShift

SongShift – це застосунок для мобільних пристроїв, що дозволяє користувачам легко мігрувати свої музичні плейлисти між різними стрімінговими сервісами. За допомогою цього застосунку можна переносити

плейлисти з одного сервісу до іншого, без необхідності створення нових плейлистів або вручну додавати кожен трек окремо [6].

SongShift був створений в 2018 році американським розробником Джастіном Дуейтом. Він почав розробляти застосунок, коли зіткнувся з проблемою міграції своїх плейлистів з Apple Music на Spotify. Дуейт вирішив створити простий і зручний інструмент, який дозволить користувачам швидко переносити свої музичні плейлисти з одного сервісу на інший.

До основних функцій та можливостей SongShift належать:

– міграція плейлистів між застосунками для відтворення аудіофайлами (Apple Music, Spotify, Tidal, YouTube Music, Deezer та інші);

– перенесення треків з плейлистів із одного сервісу на інший без необхідності створення нових плейлистів;

– перетворення плейлистів з одного формату на інший (наприклад, з Apple Music на Spotify або з Spotify на YouTube Music);

– підтримка автоматичної міграції, коли користувачі створюють нові плейлисти в підтримуваних сервісах.

1.2 Огляд основних методів для міграції аудіофайлів та плейлистів

Для того аби успішно перемістити аудіофайли, які згруповані у плейлисти існує декілька поширених способів [7]. В цьому розділі розглянуто приклади такі як інструменти імпорту та експорту в застосунках, переміщення аудіофайлів та плейлистів вручну та використання наданого застосунками API у особистих застосунках [8]. Використання всіх перелічених можливостей підтримується у популярних застосунках для відтворення музики.

1.2.1 Використання засобів імпорту/експорту в застосунках

Більшість популярних застосунків для прослуховування музики мають функції імпорту та експорту, які дозволяють переміщувати музику та плейлисти між різними сервісами. Деякі застосунки, такі як Apple Music та Google Play Music, мають вбудовані функції імпорту/експорту, які дозволяють легко переміщувати музичні файли та плейлисти між різними сервісами. Інші застосунки можуть вимагати використання сторонніх застосунків або інструментів для переміщення музики та плейлистів [9].

Однак, незалежно від того, який застосунок використовується, важливо мати на увазі, що засоби імпорту/експорту можуть не завжди бути 100% надійними. Наприклад, імпорт плейлистів може не завжди працювати належним чином або може бути обмежений кількістю пісень, яку можна імпортувати одночасно. Також можуть бути проблеми зі збереженням метаданих, таких як назви пісень та виконавці, під час імпорту.

Крім того, варто пам’ятати, що деякі застосунки можуть змінювати формат музичних файлів під час імпорту, що може вплинути на якість звуку. Тому, перед використанням засобів імпорту/експорту, важливо зробити резервну копію музики та плейлистів, щоб уникнути втрати даних [10].

1.2.2 Переміщення аудіофайлів та плейлистів вручну

Найпростішим способом переносу збережених плейлистів з одного застосунку для відтворення музики, з точки зору користувача, є створення плейлистів у новому застосунку та пошук необхідних аудіофайлів з подальшим збереженням їх у створені плейлисти. Головним недоліком є велика кількість часу, що потребує такий процес з безпосереднім залученням користувача. До того ж, може виникнути ситуація, при якій у новому

застосунку необхідний аудіофайл буде недоступний через відсутність прав на відтворення цієї пісні на окремому сервісі.

Цей підхід не можна назвати ефективним з точки зору витраченого часу та отриманої користі. Наявність у користувача великої кількості збережених аудіофайлів у одному застосунку для відтворення музики може стати причиною відмови від переходу до нового та в деяких аспектах кращого застосунку [11]. Бо деякі платформи для відтворення музики з часом можуть змінювати тарифні плани підписок, або збільшувати модерацію контенту на цій платформі, шо може призвести до скорочення загальної кількості доступних аудіофайлів у сервісі. В той же час, нові сервіси для відтворення музики можуть запропонувати вигідні умови передплат, більший список доступного контенту та вищу якість аудіофайлів. І через високий рівень трудомісткість та низьку ефективність процесу переміщення аудіофайлів та плейлистів вручну користувач може від цього відмовитись.

1.2.3 Програмна інтеграція з API наданий застосунками

Більшість сучасних застосунків для відтворення музики мають спеціальну платформу з виділеними API ендпоінтами (англ. endpoint) для управління налаштуваннями, плейлистами та аудіофайлами у застосунку. Для цього, застосунки мають окремо виділений сайт з докладною та обширною документацією, яка включає в себе набір доступних ендпоінтів та правил взаємодії з ними (рис. 1.1).

Однак, слід зазначити, що для успішної інтеграції з необхідними API ендпоінтами (англ. endpoint) користувач повинен мати базові, в деяких випадках просунуті, навички у програмуванні [12].

З одного боку, використання API для інтеграції з музичними застосунками, такими як Spotify та YouTube Music, може бути корисним під час міграції збережених плейлистів з одного застосунку на інший. Інтеграція

з застосунками, які надають API, дозволяє програмістам створювати власні інструменти та розширювати функціональність музичних сервісів.

Однак, разом з перевагами існують і недоліки використання API. Перш за все, програмна інтеграції з API може займати значно більше часу та зусиль, ніж просте використання засобів, наданих відповідним сервісом [13]. Крім того, інтеграція з API може бути пов'язана з рядом проблем, таких як проблеми з безпекою, швидкістю відповіді API та несправності підключення.

Також важливо враховувати, що доступ до API надається під певні обмеження та обмежені квоти на запити. Це означає, що програміст повинен обмежувати кількість запитів та звертатися до API з розумінням щодо його обмежень, щоб уникнути проблем з застосунком [14].

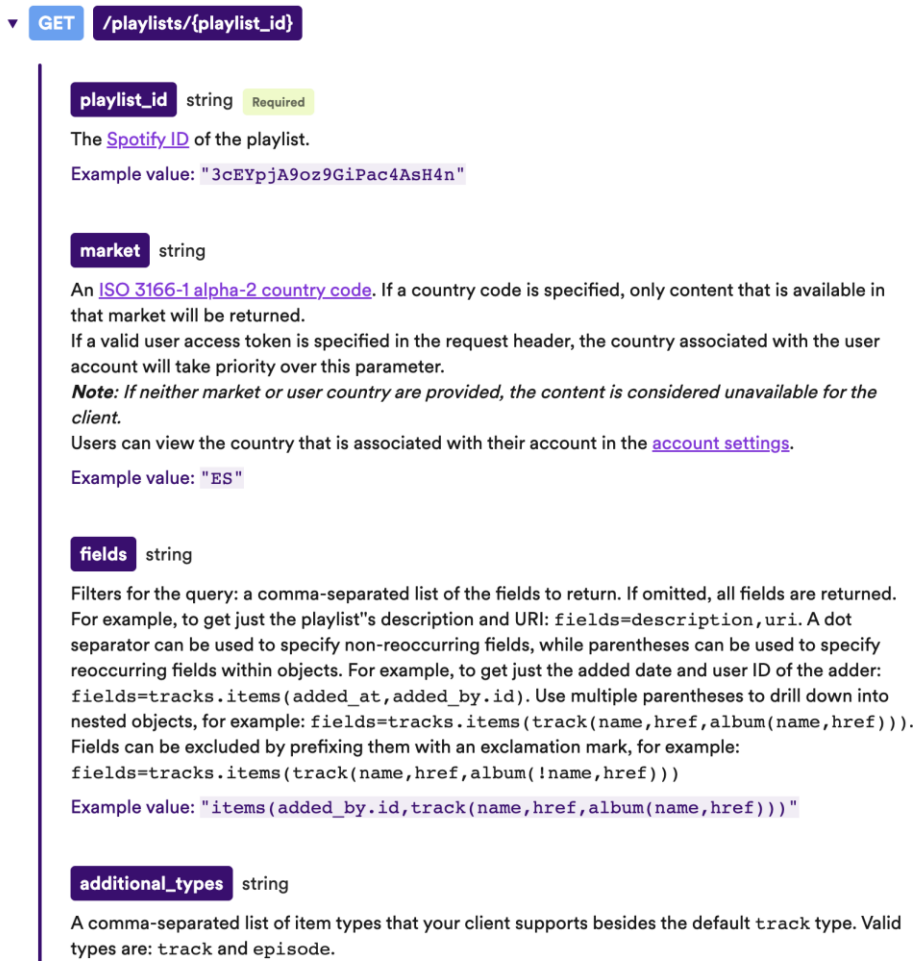


Рисунок 1.1 – Приклад документації до ендпоінта для отримання створеного користувачем плейлиста у сервісі Spotify за унікальним ідентифікатором плейлиста [15]

1.3 Порівняльний аналіз зручності та доступності застосунків для міграції аудіофайлів та плейлистів

В даному розділі складено порівняльну характеристика у вигляді таблиці трьох найбільш популярних застосунків для міграції плейлистів: Soundiiz, TuneMyMusic та SongShift.

Soundiiz є досить потужним та зручним інструментом для міграції аудіофайлів та плейлистів між різними застосунками для відтворення музики. Він підтримує багато різних музичних сервісів, включаючи Spotify, YouTube Music, Apple Music, Amazon Music та багато інших. Інтерфейс Soundiiz досить інтуїтивно зрозумілий, тому користувачі можуть легко зрозуміти, як його використовувати, головну сторінку застосунку наведено на рисунку 1.2 [16].

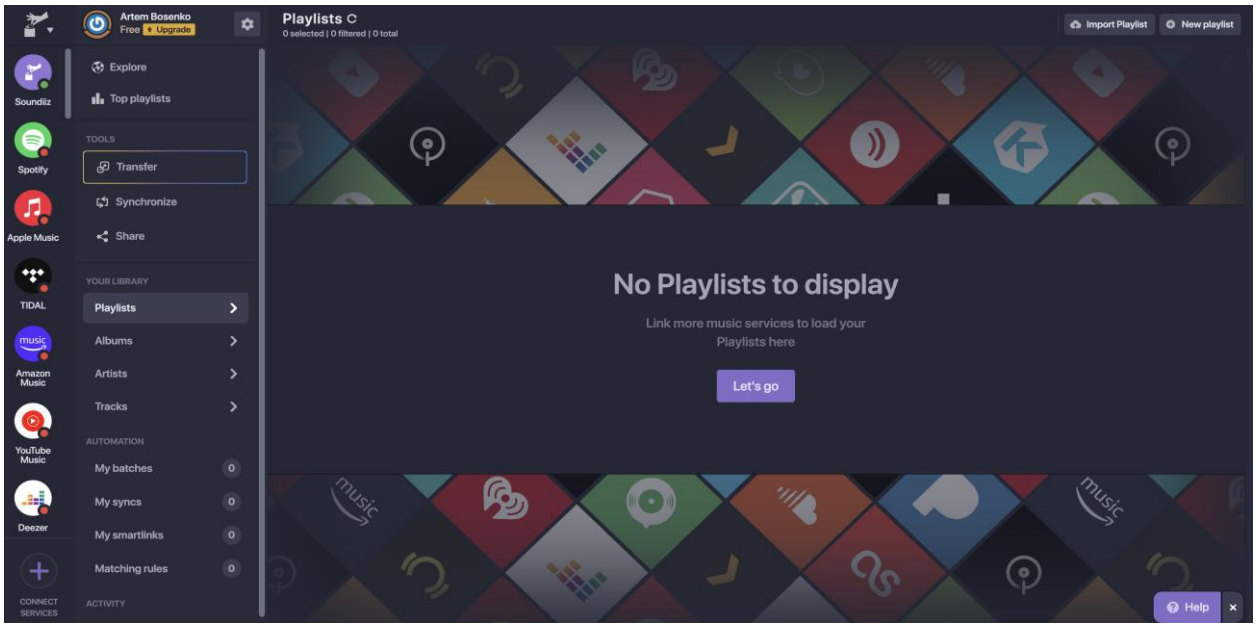


Рисунок 1.2 – Скріншот вигляду UI застосунку Soundiiz після реєстрації користувача

TuneMyMusic також є простим та зручним інструментом для міграції аудіофайлів та плейлистів між різними застосунками для відтворення музики. Цей інструмент також підтримує багато різних музичних сервісів,

включаючи Spotify, YouTube Music, Apple Music та інші. TuneMyMusic має досить простий інтерфейс, що дозволяє користувачам швидко та легко імпортувати та експортувати свою музику. В свою чергу TuneMyMusic має більш застарілий та мало функціональний UI, в порівнянні із застосунком Soundiiz, вигляд головної сторінки показано на рисунку 1.3. В свою чергу TuneMyMusic має більш застарілий та мало функціональний UI, в порівнянні із застосунком Soundiiz.

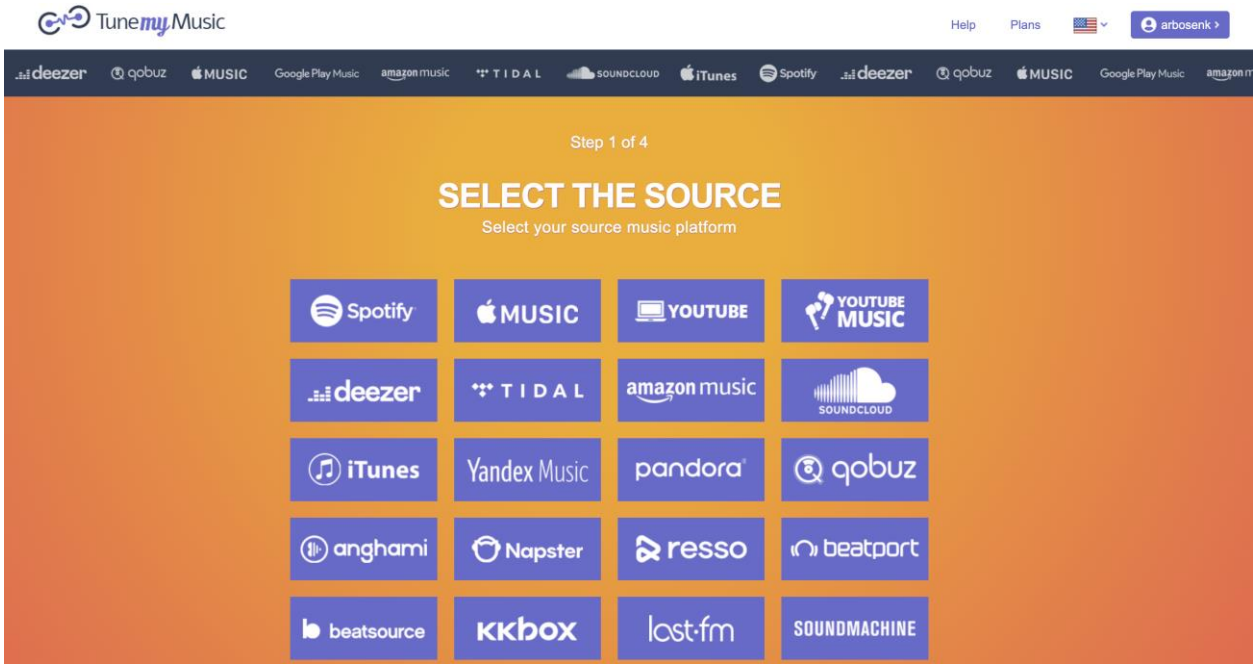


Рисунок 1.3 – Скріншот вигляду UI застосунку Soundiiz після реєстрації користувача

SongShift є більш обмеженим інструментом для міграції аудіофайлів та плейлистів між різними застосунками для відтворення музики, оскільки він працює тільки з Android-пристроями. Однак, для користувачів Android, SongShift може бути дуже зручним інструментом, оскільки він має досить простий інтерфейс та дозволяє легко переносити свою музику між різними пристроями. Для початку роботи з SongShift необхідно встановити мобільний застосунок, що є може відштовхнути користувачів, які прагнуть зробити

процес міграції використовуючи вебсайт. Порівняльна характеристика застосунків показана на таблиці 1.1.

Таблиця 1.1 – Порівняння обраних застосунків по критеріям

Критерій	**Soundiiz**	**TuneMyMusic**	**SongShift**
1	**2**	**3**	**4**
Імпорт аудіофайлів зі Spotify	Так	Так	Так
Імпорт аудіофайлів з YouTube Music	Так	Так	Так
Імпорт плейлистів зі Spotify	Так	Так	Так
Імпорт плейлистів з YouTube Music	Так	Так	Так
Експорт аудіофайлів на Spotify	Так	Так	Ні
Експорт аудіофайлів на YouTube Music	Так	Так	Ні
Експорт плейлистів на Spotify	Так	Так	Так

Продовження таблиці 1.1

1	2	3	4
Експорт плейлистів на YouTube Music	Так	Так	Ні
Інтерфейс	Простий та зрозумілий	Простий та зрозумілий	Простий та зрозумілий
Вартість	Безкоштовний на обмежений час, $4.50/місяць або $36/рік для плану Premium	Безкоштовний на обмежений час, $4.99/місяць для плану Premium	Безкоштовний для експорту до 100 пісень, $9.99/місяць для повної версії
Кількість підтримуваних музичних платформ	Понад 50	Понад 40	Декілька основних
Швидкість міграції (1 плейлист з 12 ауйдіофайлами)	8 секунд	6 секунд	16 секунд

1.4 Постановка задачі

Таким чином, створення безкоштовного сервісу для міграції аудіофайлів сформованих у плейлисти є актуальним. Тому ставиться завдання розробки зручного інтерфейсу для використання цього застосунку та стабільного і водночас надійної архітектури для інтеграції з API можливих сервісів для прослуховування музики.

Об’єктом роботи є міграція плейлистів користувача між застосунками для відтворення аудіофайлів.

Метою роботи є розробка застосунку для зручної міграції плейлистів за допомогою інтеграції з сервісами для відтворення аудіофайлів.

Для досягнення мети необхідно вирішити такі завдання:

– провести аналіз існуючих застосунків для міграції плейлистів між популярними застосунками для прослуховування музики;

– розробити та реалізувати зручний інтерфейс користувача для інтуїтивно зрозумілого використання застосунку;

– розробити архітектуру програми для інтеграції з API, який наданий застосунками для прослуховування музики, та використати можливість створити та редагувати плейлисти використовуючи наданий API;

– створити та інтегрувати базу даних для збереження інформації про вже зареєстрованих користувачів застосунку.

2 МОДЕЛІ ДЛЯ ПЕРЕДАЧІ ДАНИХ МІЖ ЗАСТОСУНКАМИ ДЛЯ ВІДТВОРЕННЯ АУДІОФАЙЛІВ

2.1 Аналіз використаних інструментів для реалізації застосунку

Для реалізації застосунку для міграції плейлистів між сервісами для відтворення музики було використані сучасні технології та підходи організації передачі даних між інтерфейсом користувача, логікою оброки даних та інтеграції з API наданим сервісами для відтворення аудіофайлів, а також для зберігання оброблених даних про користувача у базі даних.

Одним із головних компонентів будь-якого сучасного застосунку є інтерфейс користувача, адже це безпосередньо інструмент з яким взаємодіє користувач [17].

Сучасний інтерфейс користувача має буті інтуїтивно зрозумілий, тобто головний функціонал застосунку повинен бути представлений відразу після авторизації користувача. Важливою вимогою є швидкість навігації між сторінками застосунку [18]. Для реалізації цих задач було використано мову програмування Typescript та бібліотека React.

React – це відкрита JavaScript-бібліотека для розробки інтерфейсів користувача. Вона була створена Facebook і вперше була випущена у 2013 році. Історія створення React бере початок у 2010 році, коли його творець Джордан Вальк (Jordan Walke), розробляв функціональну сторінку для Facebook. Ця сторінка містила багато елементів і була дуже повільною. Тому Джордан спробував розбити сторінку на окремі компоненти і зрозумів, що це покращило її швидкість. У 2012 році Facebook розпочав роботу над новою версією своєї мобільної платформи. Вона потребувала високої швидкості, а також повинна була бути зручною для розробників. Для цього Facebook вирішив створити нову JavaScript-бібліотеку – React.

Бібліотека React досі є однією із найпоширенішою у використанні для розробки інтерфейсу користувача. За даними на 2021 рік бібліотека React

посідає третє місце серед своїх конкурентів за використанням. У 2022 році, за даними GitHub, React посідає другу сходинку із загальним рейтингом 192 тисячі уподобайок на сайті GitHub (рис. 2.1). Цю бібліотеку для своїх застосунків використовують такі компанії як: Instagram, Airbnb, Discord, Walmart [19].

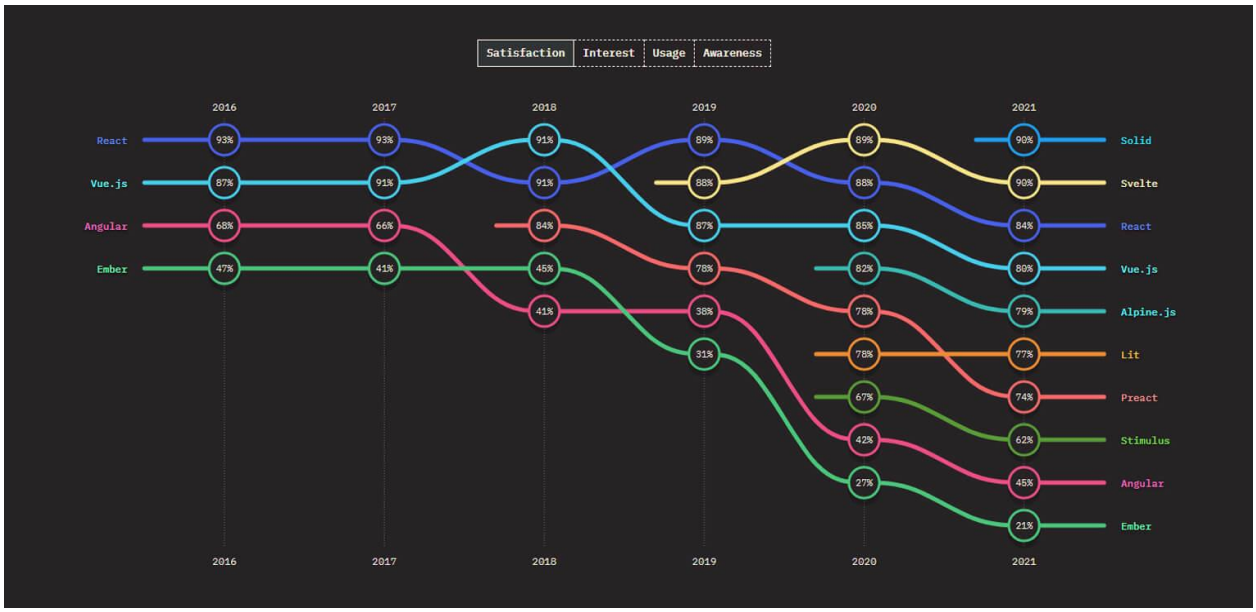


Рисунок 2.1 – Рейтинг бібліотек та фреймворків для розробки інтерфейсу користувача станом на 2021 рік

Використовуючи бібліотеку React з’являється можливість розділити вебсторінку на незалежні компоненти, що спрощує розробку та забезпечує повторне використання коду. Також, бібліотека React використовує віртуальний DOM, який дозволяє підтримувати відображення сторінки на високому рівні продуктивності. Він оновлює тільки ті елементи сторінки, які дійсно змінилися, що дозволяє зменшити кількість зайвих операцій та збільшити швидкість роботи застосунку.

React дозволяє розширювати функціональність застосунку шляхом додавання нових компонентів та модулів. Крім того, React підтримує використання багатьох інших бібліотек та фреймворків, таких як Redux чи MaterialUI. Саме бібліотека готових компонентів MaterialUI використана в

реалізації інтерфейсу користувача застосунку для міграції плейлистів між сервісами для відтворення аудіофайлів.

Для того щоб досягти максимальної швидкості взаємодії з інтерфейсом користувача необхідно відокремити логіку обробки та збереження даних від інтерфейсу користувача [20]. Тому грамотна організація логіки обробки та збереження даних є наступною важливою частиною будь-якого сучасного застосунку. Для цієї мети була обрана мова програмування Java та інструмент для збирання залежностей Gradle.

Java є однією з найпопулярніших мов програмування, яку використовують для розробки сучасних застосунків. Вона відома своєю стабільністю та надійністю, що є важливим для розробки логіки обробки та збереження даних. Java має велику кількість стандартних бібліотек та фреймворків, що спрощує процес розробки та забезпечує високу якість програмного забезпечення.

Одним з найважливіших компонентів розробки сучасних застосунків є система зберігання даних. Використання баз даних дозволяє зберігати, організовувати та отримувати доступ до даних у застосунку [21].

Для збереження даних про користувача у застосунку для міграції плейлистів між сервісами для відтворення аудіофайлів було використано реляційну систему управління базами даних PostgreSQL. У свою чергу, Java має багато бібліотек для роботи з реляційними та нереляційними базами даних, включаючи Hibernate та Spring Data. Ці бібліотеки дозволяють легко підключатися до баз даних та працювати з ними в зручному форматі, що забезпечує ефективну роботу з даними у застосунку.

Gradle є інструментом автоматизації збирання та управління залежностями в проектах Java. Таким чином це дозволяє ефективно налаштувати процес збірки та розгортування застосунку, що забезпечує швидку та ефективну розробку. За допомогою Gradle можна визначити залежності проекту та автоматично встановлювати їх, що дозволяє зберігати

час та зусилля розробників [22]. Окрім того, Gradle дозволяє використовувати різні плагіни та інші інструменти для спрощення розробки.

Тому використання Java та Gradle дозволяє ефективно реалізувати логіку обробки та збереження даних у своїх застосунках. Java має багато стандартних бібліотек та фреймворків для роботи з базами даних, що спрощує розробку та забезпечує високу якість програмного забезпечення. Gradle дозволяє автоматизувати процес збирання та управління залежностями, що забезпечує ефективну та швидку розробку.

Для організації передачі даних між однією частиною застосунку з інтерфейсом користувача та іншою частиною з логікою обробки та зберігання даних необхідно використовувати архітектуру REST API.

REST – це набір архітектурних обмежень, а не протокол чи стандарт. Розробники API можуть реалізовувати REST різними способами. Коли клієнтський запит виконується через RESTful API, він передає представлення стану ресурсу тому, хто зробив запит [23]. Цю інформацію або представлення можна надіслати у різних форматах через HTTP: JSON (Javascript Object Notation), HTML, XLT, Python, PHP або простий текст. JSON є найбільш популярним форматом файлів завдяки тому, що, незважаючи на свою назву, він є безпосередньо не залежним від мови програмування, і зрозумілим як для людини, так і для машини.

Також важливою деталлю є заголовки та параметри також мають значення в методах HTTP запитів у RESTful API, оскільки вони містять важливу інформацію про ідентифікатор запиту, метадані, авторизацію, URI, кешування, кукі та багато іншого. Існують запитові та відповідні заголовки, кожен з них має свою власну інформацію про з’єднання та коди статусу HTTP.

Тому всі інструменти, які були використані у розробці застосунку для міграції плейлистів між сервісами для відтворення аудіофайлами об’єднуються у трьохрівневу архітектуру застосунку. Трьохрівнева архітектура програмування (англ. three-tier architecture) є однією з найбільш

популярних архітектур для розробки застосунків та вебсайтів. Ця архітектура включає три рівні або компоненти: інтерфейс користувача (Presentation layer), логічний рівень (Application layer) та рівень даних (Data layer). Кожен з цих рівнів має свою функцію та відповідальність.

Перший рівень, інтерфейс користувача, забезпечує зв'язок між користувачем та програмою. Цей рівень включає в себе вебсторінки, графічний інтерфейс користувача, мобільні застосунки та інші інтерфейси, що використовуються для взаємодії з користувачами. Інтерфейс користувача не повинен містити логіки бізнес-процесів, його завдання – це передача даних користувачам та отримання від них даних для подальшої обробки на наступних рівнях.

Другий рівень, логічний рівень, виконує обробку даних, отриманих від інтерфейсу користувача. Цей рівень включає в себе логіку бізнес-процесів, обчислення та перетворення даних, обробку помилок та інші функції, що необхідні для коректної обробки даних. Логічний рівень може складатися з кількох компонентів, таких як сервіси, контролери та репозиторії.

Третій рівень, рівень даних, відповідає за зберігання та доступ до даних. Цей рівень включає в себе бази даних, файлові системи та інші механізми збереження даних. Рівень даних повинен бути абстрагований від деталей логічного рівня та інтерфейсу користувача, щоб забезпечити максимальну гнучкість та легкість змін в системі.

Цей рівень також може містити інструменти для забезпечення безпеки даних, кешування, реплікації даних та інших аспектів, пов'язаних з зберіганням та доступом до даних.

Така архітектура дозволяє підтримувати високий рівень безпеки та стабільності системи шляхом забезпечення логічного розділення функцій та застосування стандартизованих методів обробки даних. Схему побудови трьохрівневої архітектури показано на рисунку 2.2.

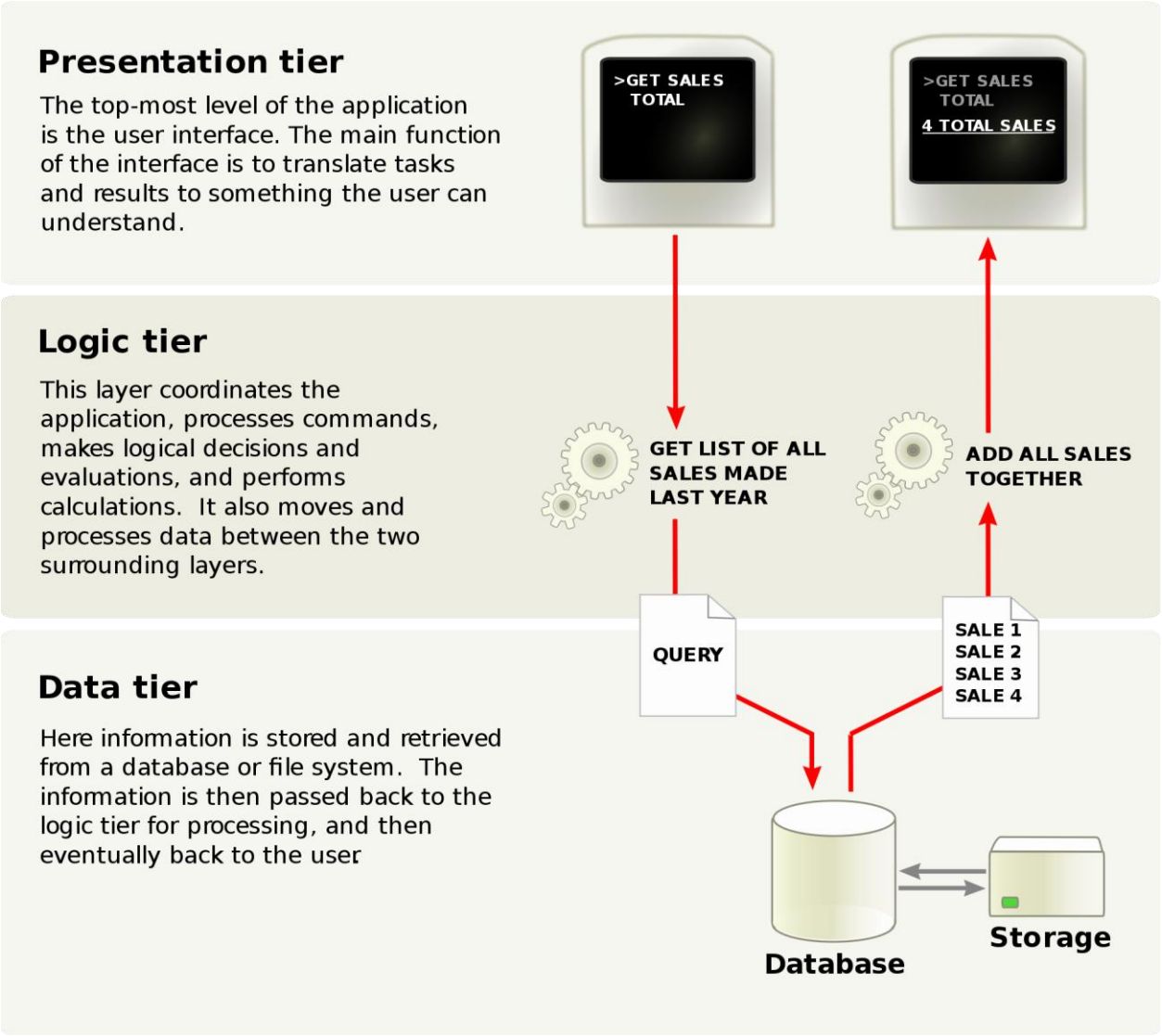


Рисунок 2.2 – Схема трьохрівневої архітектуру сучасного застосунку [24]

2.2 Моделі для інтеграції з API застосунків для відтворення та організації аудіофайлів

З моменту появи API в інформаційному просторі, стали доступні нові можливості для розробників програмного забезпечення. Зокрема, за допомогою API можна легко інтегрувати різні функції та сервіси в свої застосунки. Одним з таких сервісів є API для відтворення та організації аудіофайлів.

Для інтеграції з API застосунків для відтворення та організації аудіофайлів, необхідно вибрати необхідну модель. Моделі можуть бути побудовані на різних технологіях, таких як REST, SOAP, GraphQL тощо. Вони мають свої переваги та недоліки, тому необхідно ретельно досліджувати кожну технологію перед вибором моделі для свого застосунку. Наведемо порівняльну таблицю для технологій, наведених вище (табл. 2.1). Критеріями будуть: архітектура, формат передачі даних, тип передачі даних, можливість кешування, можливість масштабування, можливість версіювання, надійність.

Таблиця 2.1 – Порівняння технологій для реалізації API

Критерії порівняння	**REST**	**SOAP**	**GraphQL**
1	**2**	**3**	**4**
Архітектура	Концепція заснована на створенні легкого та швидкого інтерфейсу для взаємодії з даними	Концепція заснована на створенні більш складного інтерфейсу, що надає більш широкі можливості	Концепція заснована на гнучкому підході до запитів та отримання лише необхідних даних
Формат передачі даних	JSON, XML, HTML	XML	JSON
Тип передачі даних	Синхронна	Синхронна та асинхронна	Синхронна та асинхронна
Кешування	Підтримується кешування запитів	Підтримується кешування запитів	Не підтримується

Продовження таблиці 2.1

1	2	3	4
Масштабування	Легке та ефективне масштабування	Складне масштабування	Легке масштабування
Версіювання	Підтримується	Підтримується	Підтримується
Надійність	Надійніший та стійкий до збоїв	Надійний, але менш стійкий до збоїв	Надійний, але менш стійкий до збоїв

Слід зазначити, що надійність технологій для реалізації API в таблиці вище може бути виміряна за допомогою різних методів. Один з них – це використання статистичних даних про відсоток відмов технологій, які були зібрані з різних джерел, таких як документація, блоги, форуми, тестування продуктів тощо. Тому, REST є більш стійким до збоїв порівняно з SOAP та GraphQL з кількох причин:

– REST використовує HTTP протокол, який є стандартним і підтримується більшістю вебсерверів і клієнтів. Це забезпечує більш широку сумісність і забезпечує, що більшість проблем, пов’язаних з мережею, вже були вирішені із запровадженням HTTP;

– REST використовує кешування, що може знизити кількість запитів до сервера та зменшити ймовірність виникнення збоїв через перевантаження мережі. Кешування дозволяє клієнтам зберігати копії ресурсів, що дозволяє зменшити кількість запитів до сервера;

– REST використовує простішу структуру запитів та відповідей порівняно з SOAP та GraphQL. Це дозволяє зменшити можливість помилок та простіше розуміти, як працює система;

– REST працює з ресурсами, що дає можливість виконувати часткові запити та обмінюватися тільки частинами ресурсу. Це забезпечує більшу гнучкість та зменшує обсяг даних, які передаються. Як результат

зменшується ймовірність виникнення збоїв через обмеження мережевих ресурсів [25].

Завдяки цьому найпоширенішою технологією для реалізації API є REST (Representational State Transfer). Вона є простою та ефективною, тому саме ця технологія використана для реалізації API застосунків для відтворення та організації аудіофайлів.

Для застосунку для міграції плейлистів між сервісами для відтворення музикиі необхідно використовувати такі можливості як: отримання, створення, оновлення контенту плейлистів. Також необхідно використовувати отримання та пошук аудіофайлів для подальшого збереження у необхідний плейлист.

Важливо зазначити, що перед відправленням запиту за допомогою REST API будь-якого сервісу для відтворення музики, користувач повинен мати спеціальний токен доступу, що дозволяє проводити взаємодію з інструментами застосунку для окремого користувача.

2.2.1 Моделі використані для API застосунку Spotify

Spotify API надає всі необхідні можливості для успішної взаємодії з однойменним сервісом для відтворення музики. Розробка Spotify API розпочалася у 2009 році, коли Spotify був тільки випущений в Швеції і мав обмежену кількість користувачів. Розробники Spotify вирішили створити API, щоб дозволити іншим розробникам використовувати Spotify та створювати застосунки на основі музичних даних Spotify.

Перші версії Spotify API були обмежені функціональністю та досить складними для використання. Однак з часом розробники Spotify почали додавати нові функції та поліпшувати API, щоб зробити його більш доступним та зручним для використання.

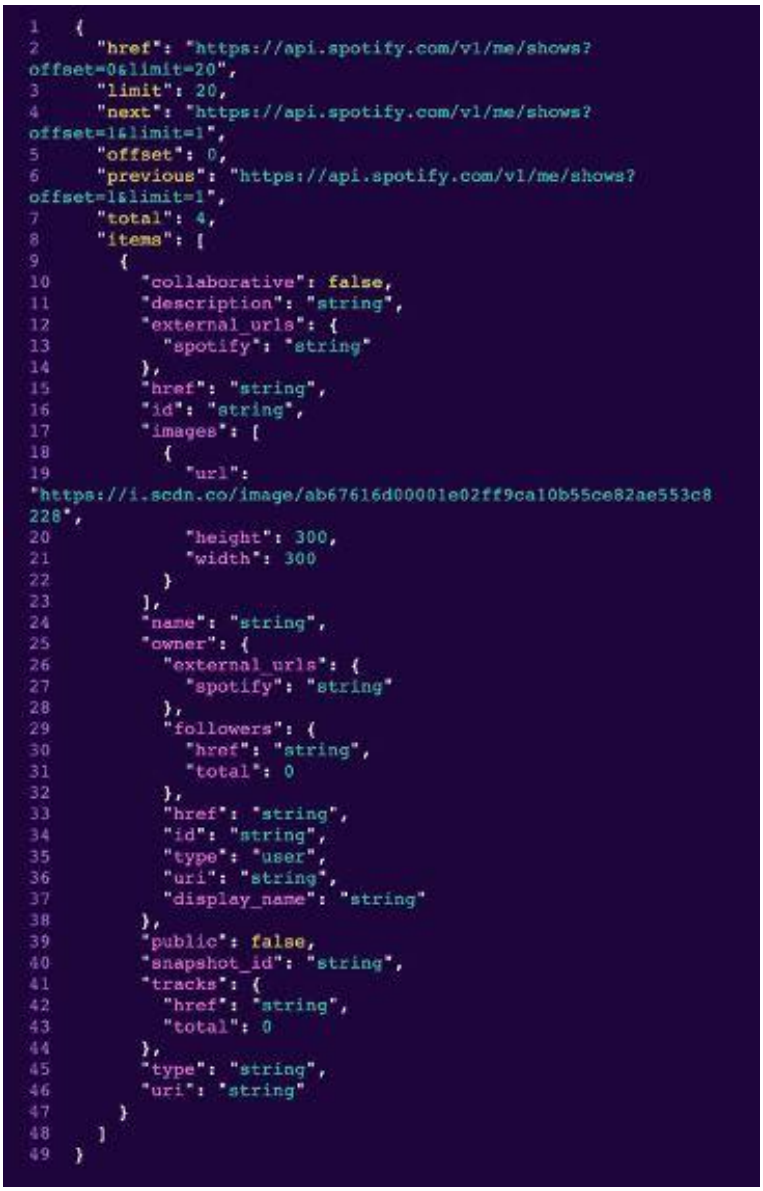
Сьогодні Spotify API є одним з найпопулярніших API серед розробників музичних застосунків та сервісів. Він містить широкий спектр функцій, включаючи доступ до музики, плейлистів, аудіофайлів. Особливо варто зазначити, що Spotify API надає можливість взаємодії з Spotify Connect, що дозволяє розробникам контролювати відтворення музики на різних пристроях, таких як комп'ютери, мобільні пристрої та інші підключені пристрої.

У загальному, Spotify API є потужним інструментом для розробників, який дозволяє створювати різноманітні музичні застосунки та сервіси, що взаємодіють з музичним сервісом Spotify.

Розглянемо можливість отримання загального списку плейлистів користувача. Для цього потрібно за допомогою REST можна здійснити, використовуючи HTTP GET запит за наступним URL: *https://api.spotify.com/v1/me/playlists* [26]. У відповідь буде отримано модель з інформацією про загальну кількість плейлистів, та список із інформацією про кожний окремий плейлист. Для кожного плейлиста необхідно дізнатися наступні характеристики: назву та загальну кількість пісень (рис. 2.3).

Для створення плейлиста користувача через Spotify API за допомогою REST потрібно відправити POST запит на адресу *https://api.spotify.com/v1/users/{user_id}/playlists*, де user_id – це ідентифікатор користувача. Також потрібно передати у відповідному форматі JSON об'єкт з наступною інформацією про плейлист: назва, опис плейлиста та інформацію про публічність плейлиста.

Для оновлення аудіофайлів у плейлисті можна скористатися HTTP запитом PUT на адресу *https://api.spotify.com/v1/playlists/{playlist_id}/tracks* з оновленими даними про пісні в тілі запиту. Аналогічно для видалення пісень з плейлисту можна використати HTTP запит DELETE.



```
{
    "href": "https://api.spotify.com/v1/me/shows?
offset=0&limit=20",
    "limit": 20,
    "next": "https://api.spotify.com/v1/me/shows?
offset=1&limit=1",
    "offset": 0,
    "previous": "https://api.spotify.com/v1/me/shows?
offset=1&limit=1",
    "total": 4,
    "items": [
      {
        "collaborative": false,
        "description": "string",
        "external_urls": {
          "spotify": "string"
        },
        "href": "string",
        "id": "string",
        "images": [
          {
            "url":
"https://i.scdn.co/image/ab67616d00001e02ff9ca10b55ce82ae553c8
228",
            "height": 300,
            "width": 300
          }
        ],
        "name": "string",
        "owner": {
          "external_urls": {
            "spotify": "string"
          },
          "followers": {
            "href": "string",
            "total": 0
          },
          "href": "string",
          "id": "string",
          "type": "user",
          "uri": "string",
          "display_name": "string"
        },
        "public": false,
        "snapshot_id": "string",
        "tracks": {
          "href": "string",
          "total": 0
        },
        "type": "string",
        "uri": "string"
      }
    ]
}
```


Рисунок 2.3 – Приклад моделі загальної моделі «Плейлисти», яка надається Spotify API

2.2.2 Моделі використані для API застосунку YouTube Music

В один ряд зі Spotify по можливостям та охопленій аудиторії серед сервісів для відтворення аудіофайлів стає також застосунок YouTube Music. Для успішної інтеграції YouTube Music необхідно використовувати YouTube Data API [27]. Цей інтерфейс дозволяє взаємодіяти із усією платформою YouTube, адже застосунок YouTube Music – це застосунок, який базується на

аудіо та відеофайлах які знаходяться в YouTube. З єдиним виключенням, що в YouTube Music можна відтворювати тільки музику та музичні кліпи, інші відеофайли не доступні для відтворення у YouTube Music. Загалом YouTube Data API надає великий список можливостей:

– пошук відео: за допомогою YouTube Data API розробники можуть здійснювати пошук відео за різними параметрами, такими як ключові слова, категорії, теги, час публікації;

– отримання даних про канали: розробники можуть отримувати інформацію про канали на YouTube, включаючи їхню статистику, опис та додаткову інформацію;

– редагування списку відтворення: API дозволяє розробникам додавати та видаляти відео зі списку відтворення користувача на YouTube;

– додавання відео: розробники можуть додавати нові відео на YouTube, включаючи відео та метадані.

Розглянемо можливість отримання всіх плейлистів через YouTube Data API за допомогою REST, необхідно зробити запит GET за адресою: *https://www.googleapis.com/youtube/v3/playlists?part=snippet&mine=true&key=[API_KEY],* де: part - список розділів запиту, що повертається. У прикладі необхідно отримати лише snippet, тобто назву, опис і т.д. плейлистів. Mine – фільтр, який повертає тільки плейлисти користувача, який виконує запит. Key – API ключ автентифікації користувача для відправки запиту. У відповідь на запит, якщо вимоги аутентифікації виконані, буде отримано загальну кількість та список плейлистів користувача, приклад показано на рисунку 2.4.

Для створення нового плейлиста користувача через YouTube Data API за допомогою REST необхідно відправити POST запит за адресою *https://www.googleapis.com/youtube/v3/playlists?part=snippet.* Також, при відправленні запиту необхідно вказати назву плейлиста, опис та обрану мову. Без цієї інформації неможливо створити плейлист за допомогою взаємодії з YouTube Data API.

Для оновлення списку файлів у вже створеному плейлисті користувача через YouTube Data API за допомогою REST необхідно виконати PUT запит на наступний URL: https://www.googleapis.com /youtube/v3/playlistItems?part=snippet&key={API_KEY}. Де part – список розділів запиту, в якому передається інформація про плейлист та файли. В даній роботі необхідно передати лише snippet. Key – API ключ, сгенерований для доступу до YouTube Data API.

```
{
  "kind": "youtube#playlistListResponse",
  "etag": etag,
  "nextPageToken": string,
  "prevPageToken": string,
  "pageInfo": {
    "totalResults": integer,
    "resultsPerPage": integer
  },
  "items": [
    {
      "kind": "youtube#playlist",
      "etag": etag,
      "id": string,
      "snippet": {
        "publishedAt": datetime,
        "channelId": string,
        "title": string,
        "description": string,
        "thumbnails": {
          (key): {
            "url": string,
            "width": unsigned integer,
            "height": unsigned integer
          }
        },
        "channelTitle": string,
        "tags": [
          string
        ],
        "defaultLanguage": string,
        "localized": {
          (key): string
        },
        "privacyStatus": string,
        "status": {
          "privacyStatus": string
        }
      },
      "contentDetails": {
        "itemCount": integer
      }
    }
  ]
}
```

Рисунок 2.4 – Приклад моделі «Плейлист», яка надається YouTube Data API

2.3 Моделі для організації поширення даних всередині застосунку для міграції плейлистів між застосунками для відтворення аудіофайлів

Однією з ключових необхідностей використання моделей для організації поширення даних всередині застосунку для міграції плейлистів між застосунками для відтворення аудіофайлів є забезпечення сумісності форматів даних між різними застосунками [28]. Різні сервіси для відтворення аудіофайлів можуть використовувати різні формати даних для представлення плейлистів, метаданих про треки і звукові файли. Використання моделей дозволяє уніфікувати ці формати даних і забезпечити безперешкодну міграцію плейлистів між різними сервісами.

2.3.1 Моделі для поширення даних між front end та back end

Зі сторони інтерфейсу користувача (front end) надходять REST запити на сторону організації та обробки даних (back end). Після успішного виконання логіки у відповідь на запит інтерфейс користувача отримує спрощену модель даних (рис. 2.5). Це використовується для спрощення обробки даних та економії витрат пам’яті у браузері на стороні інтерфейсу.

Рисунок 2.5 – Спрощення модель «Плейлист», яка буде використана на стороні інтерфейсу користувача

2.3.2 Моделі для поширення даних між back end та базою даних

У базі даних застосунку для міграції плейлистів між сервісами для відтворення музики зберігається лише інформація про користувача, який зареєстрований у самому застосунку. Також, важливими даними є згенеровані користувачем авторизаційні токени для кожного окремого сервісу для відтворення музики. Тому буде використано лише одну модель для поширення даних між back end та базою даних, структуру якої показано на рисунку 2.6.

Рисунок 2.6 – Приклад моделі «Користувач»

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПОСТАВЛЕНОЇ ЗАДАЧІ ТА ТЕСТУВАННЯ ЗАСТОСУНКУ

3.1 Обґрунтування вибору середовища програмної реалізації застосунку

Для розробки прототипів та макетів дизайну в кваліфікаційній роботі було використано вебзастосунок Figma, яка надає великий функціонал для створення макетів будь-якої складності (рис. 3.1).

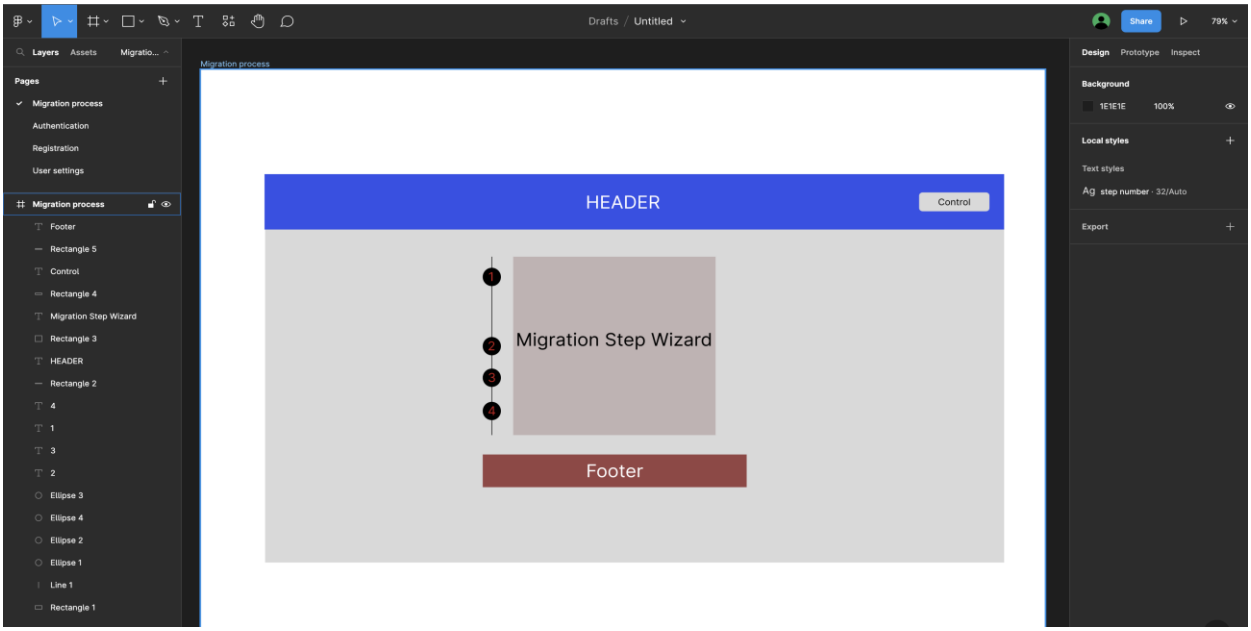


Рисунок 3.1 – Приклад інтерфейсу редактора Figma

Головними перевагами Figma серед схожих застосунків є можливість створювати дизайн макетів, який можна експортувати в різні формати, що дозволяє розробникам без проблем використовувати ці файли для створення застосунків. Figma має автоматизацію рутинної роботи: застосунок містить в собі інструменти для автоматизації рутинних задач, таких як розміщення елементів на макеті, що зменшує час, необхідний для розробки макетів.

Для реалізації клієнтської частини було використано зручний та багатофункціональний редактор Webstorm, який містить підтримки більшості

мов програмування і також сучасних фреймворків для розробки інтерфейсу користувача. Також містить безкоштовні плагіни для зручного програмування вебзастосунку за допомогою React [29]. Вигляд інтерфейсу редактора Webstorm наведено на рисунку 3.2.

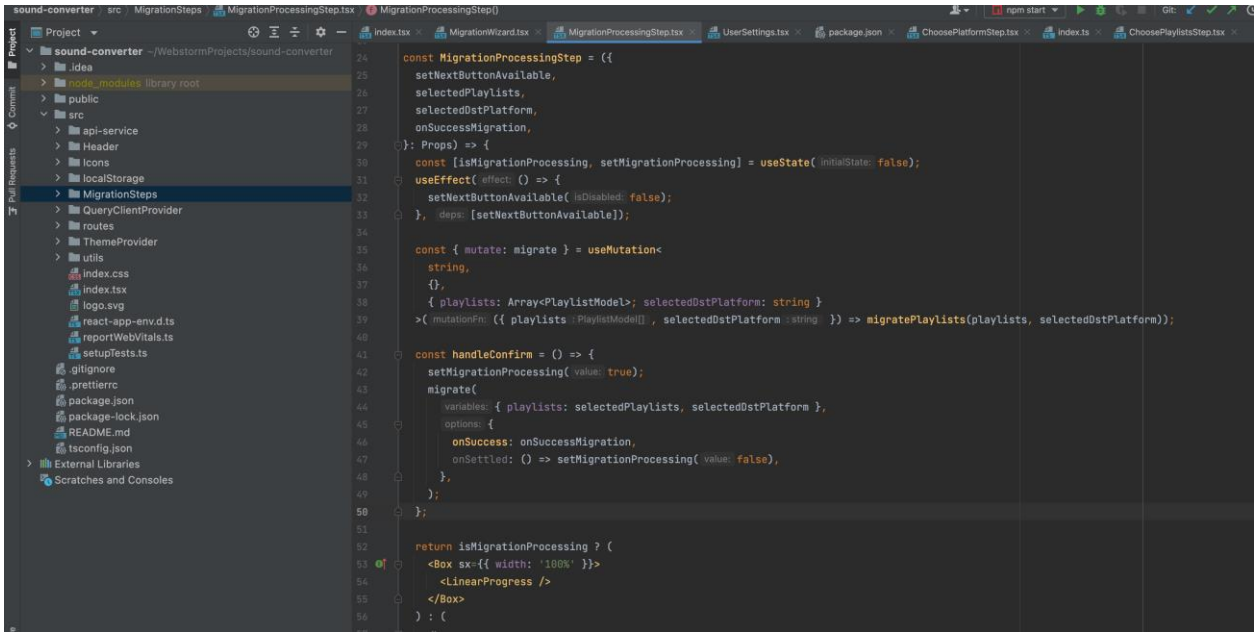


Рисунок 3.2 – Інтерфейс редактора Webstorm

Для розробки серверної частини з повною підтримкою мови програмування Java використовується інтегроване середовище розробки коду IntelliJ Idea.

Це середовище є потужним та зручним інструментом для розробки на мові програмування Java з великою кількістю функціональних можливостей та інструментів для підвищення продуктивності розробки. Найбільшими перевагами є вбудована підтримка сучасних фреймворків та бібліотек, таких як Spring, Hibernate, Struts, Maven і Gradle.

Також є підтримка широкого спектру інструментів для автоматичного рефакторингу коду, таких як перейменування, вилучення методу, переміщення класу та багато іншого. І найбільшою перевагою IntelliJ IDEA є інтуїтивно зрозумілий та зручний інтерфейс, що дозволяє швидко знайти

необхідний інструмент для покращення процесу розробки застосунків (рис. 3.3).

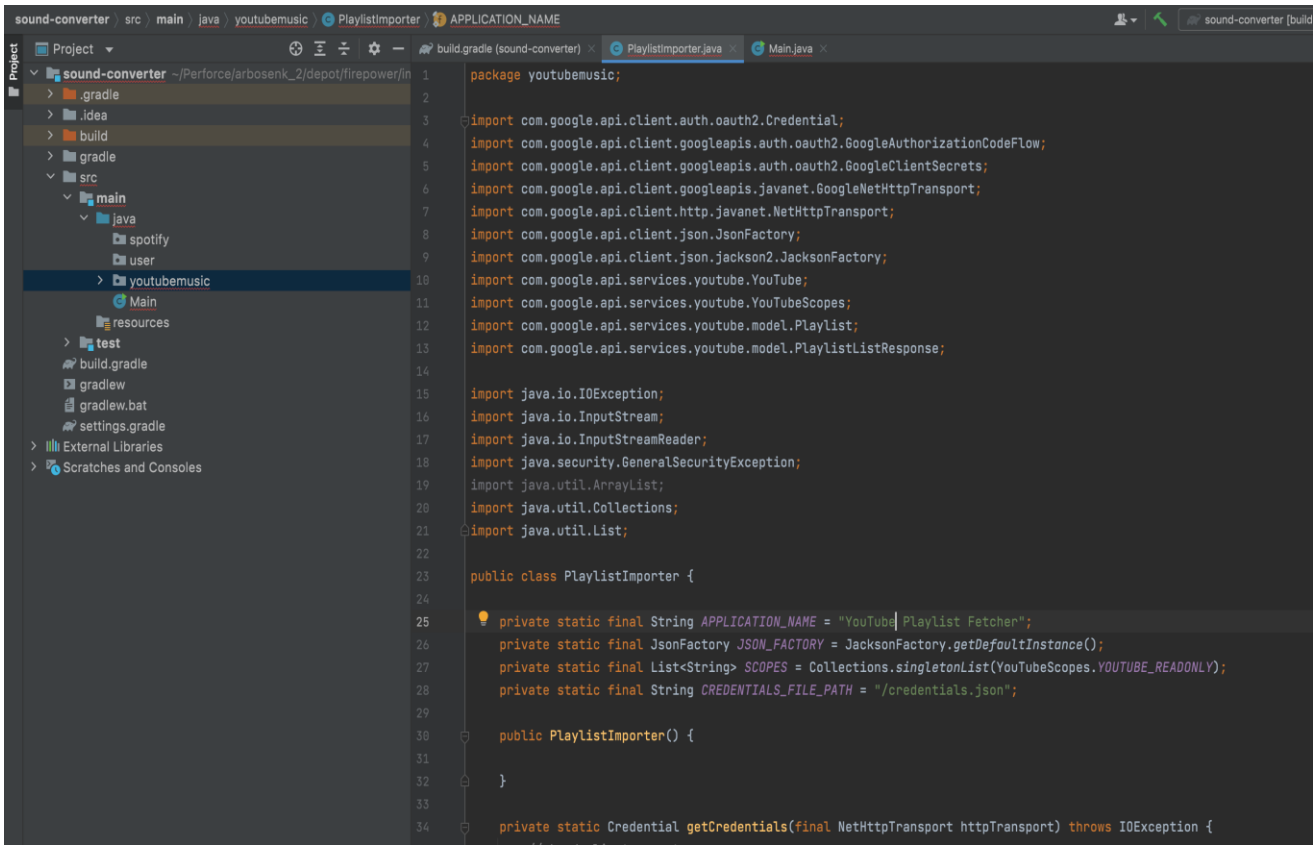


Рисунок 3.3 – Інтерфейс інтегрованого середовища розробки IntelliJ Idea

Для реалізації бази даних була обрана об’єктно-реляційна система управління базами даних PostgreSQL. PostgreSQL використовує та розширює функціонал мови SQL у поєднанні з багатьма функціями, які безпечно зберігають та масштабують найскладніші робочі навантаження даних.

На сьогоднішній день PostgreSQL знаходиться у списку популярніших СУБД, завдяки своєї надійності, архітектурі, цілісності даних та розширюваності [30].

PostgreSQL є не просто реляційним, а об’єктно-реляційним і підтримує складні структури та визначені користувачем типи даних. PostgreSQL має безліч функцій, спрямованих на обезпечення цілісності даних.

Також СУБД PostgreSQL надає можливість керувати даними, незалежно від розміру бази даних. PostgreSQL має безліч розширених функцій, такі як:

- визначені користувачем типи;
- наслідування таблиці;
- складний запірний механізм;
- довідкова цілісність зовнішнього ключа;
- перегляди, правила, підзапит;
- вкладені транзакції;
- багатоверсійний контроль паралельності (MVCC);
- асинхронна реплікація.

Для збільшення зручності налаштування та взаємодії з СУБД PostgreSQL було використано сучасний інструмент pgAdmin 4.

PgAdmin 4 дозволяє легко керувати об’єктами бази даних, такими як таблиці, індекси, перегляди, функції та інші. Він також надає можливість перегляду структури бази даних та її об’єктів.

Також, у застосунку pgAdmin 4 є підтримка вбудованого редактору запитів, що дозволяє легко створювати та виконувати SQL-запити. Це допомагає швидко отримувати необхідні дані з бази даних та виконувати SQL-запити [31]. Для збереження даних про налаштування СУДБ існує підтримка створення резервних копій.

PgAdmin 4 дозволяє легко створювати резервні копії бази даних та відновлювати їх в разі потреби. Це допомагає зберігати дані та забезпечує швидке відновлення у випадку втрати.

Потужною перевагою є графічний інтерфейс користувача у цьому застосунку: pgAdmin 4 має інтуїтивно зрозумілий та легкий у використанні графічний інтерфейс користувача, що дозволяє легко керувати базою даних та її об’єктами (рис. 3.4).

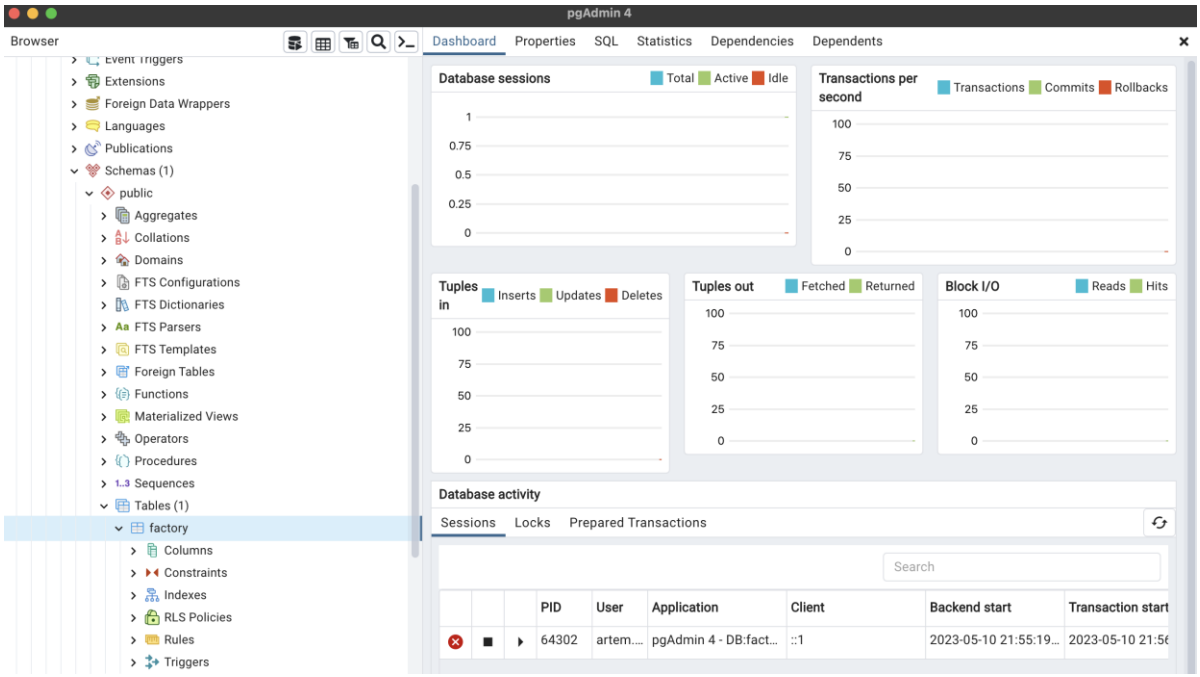


Рисунок 3.4 – Інтерфейс застосунку pgAdmin 4

3.2 Програмна реалізація застосунку для міграції плейлистів між застосунками для відтворення аудіофайлів

У даній роботі для реалізації інтерфейсу користувача використана бібліотека компонентів Material UI. Використання цієї бібліотеки дозволить побудувати клієнтську частину у одному стилі з урахуванням сучасних вимог до вигляду окремих компонентів та сторінок, які складаються з цих компонентів [32].

На сторінці авторизації знаходиться поле для вводу поштової адреси та пароля, перемикач для збереження згенерованого токену авторизації у локальному сховищі браузеру при включенні перемикача, кнопку переходу до форми реєстрації та кнопку відправлення запиту (рис. 3.5).

Для реалізації даної форми було використано бібліотеку Formik. Використання цієї бібліотеки забезпечує егко і швидко створювати форми з меншою кількістю коду, що полегшує їх розробку та підтримку. Також використання Formik відкриває можливість керувати станом форми та

валідацією введених даних, навіть з використанням сторонніх бібліотек. Це дозволяє розробникам зосередитися на важливих завданнях та мінімізувати кількість багів, пов’язаних з формами.

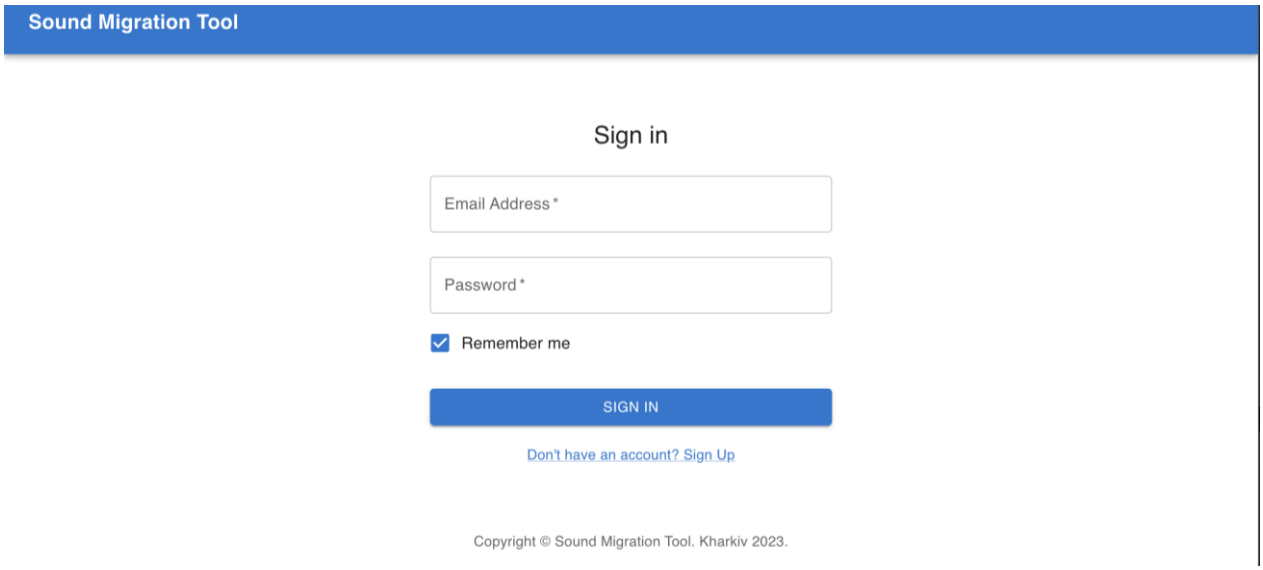


Рисунок 3.5 – Форма авторизації користувача

Після введення даних у необхідні поля на формі запускається процес валідації введених даних. У випадку, якщо користувач надав дані, що не відповідають вимогам, які прописані у схемі для валідації форми (рис. 3.6), буде відображено помилки на біля відповідного поля (рис. 3.7, 3.8).

```
export const loginSchema = yup.object().shape( additions: {
  email: yup
    .string() StringSchema<string | undefined, AnyObject, undefined, "">
    .required( msg: 'Email is required') StringSchema<NonNullable<string | undefined>, AnyObject, undefined, "">
    .trim() StringSchema<NonNullable<string | undefined>, AnyObject, undefined, "">
    .matches(
      regex: /^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$/,
      options: 'Email is not valid',
    ),
  password: yup
    .string() StringSchema<string | undefined, AnyObject, undefined, "">
    .required( msg: 'Password is required') StringSchema<NonNullable<string | undefined>, AnyObject, undefined, "">
    .matches(
      regex: /^.*(?=.{8,})((?=.*[!@#$%^&*()\-_=+{};:,<.>]){1})(?=.*\d)((?=.*[a-z]){1})((?=.*[A-Z]){1}).*$/,
      options: 'Password must contain at least 8 characters, one uppercase, one number and one special case character',
    ),
});
```

Рисунок 3.6 – Схема валідації даних для форми авторизації користувача

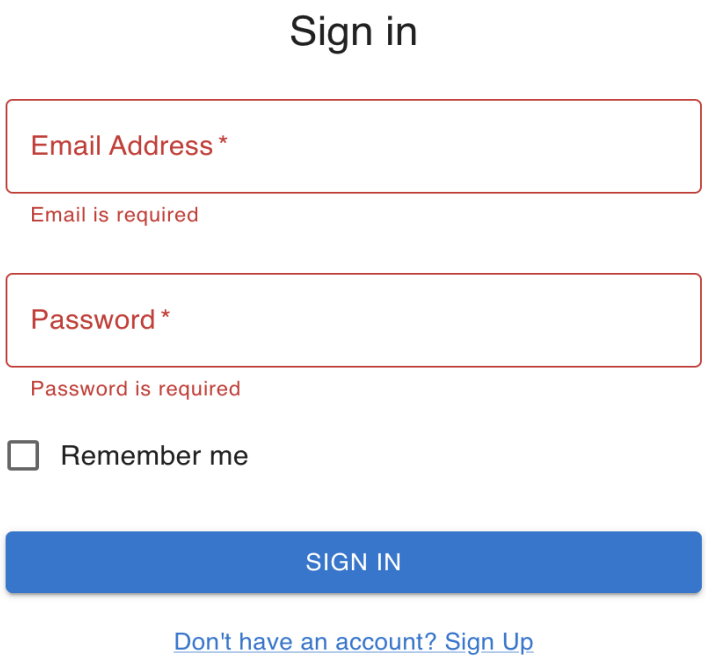


Рисунок 3.7 – Вигляд форми, якщо користувач не ввів дані

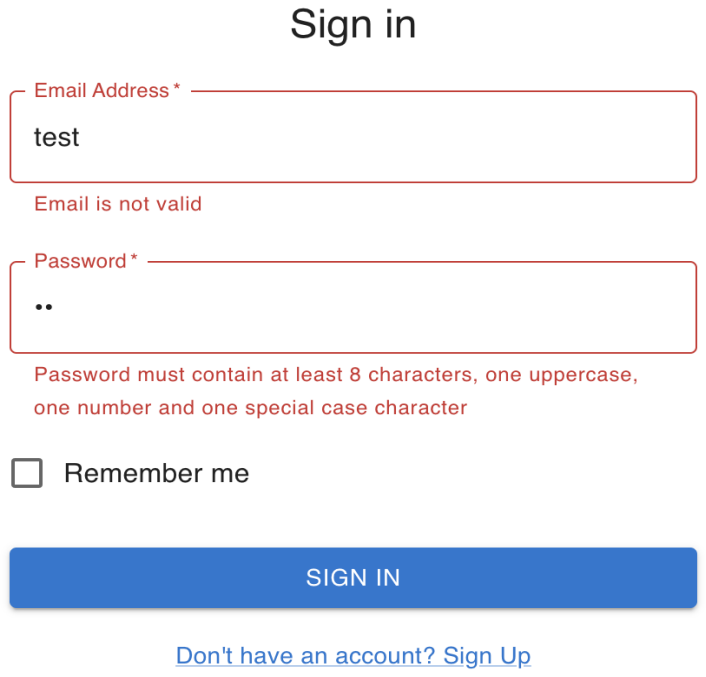


Рисунок 3.8 – Вигляд форми, якщо користувач ввів некоректний формат даних

Якщо валідація даних на клієнтській частині пройшла успішно, буде надіслано запит для авторизації на серверну частину застосунку. За умови, що дані про користувача вже є у базі даних, буде виконано перехід до

головної сторінки застосунку та збережено токен авторизації у локальне сховище браузеру, якщо користувач зазначив це на формі.

Якщо користувач ще не був зареєстрований у застосунку, необхідно перейти до відповідної форми (рис. 3.9) натиснувши кнопку «Don’t have an account? Sign Up».

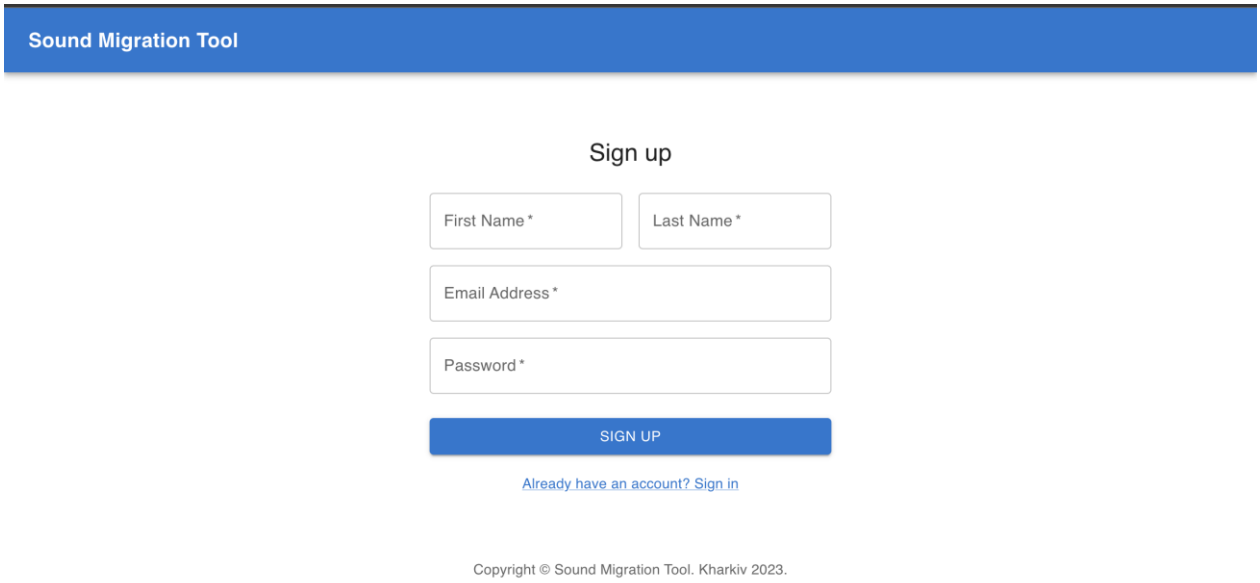


Рисунок 3.9 – Форма реєстрації користувача

На формі для реєстрації користувача знаходяться поле для вводу імені та фамілії користувача, поштової адреси, пароля та поле для підтвердження пароля, кнопку переходу до форми авторизації та кнопку відправлення запиту.

Як і у випадку з розробкою форми авторизації, форма реєстрації також реалізована з використанням бібліотеки Formik.

Також по аналогії з використанням схеми валідації для форми авторизації було розроблено схему валідації даних для форми реєстрації користувача (рис. 3.10).

```
export const registrationSchema = yup.object().shape( additions: {
  firstName: yup.string().required( msg: 'First name required'),
  lastName: yup.string().required( msg: 'First name required'),
  email: yup
    .string() StringSchema<string | undefined, AnyObject, undefined, "">
    .required( msg: 'Email is required') StringSchema<NonNullable<string | undefined>, AnyObject, undefined, "">
    .trim() StringSchema<NonNullable<string | undefined>, AnyObject, undefined, "">
    .matches(
      regex: /^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$/,
      options: 'Email is not valid',
    ),
  password: yup
    .string() StringSchema<string | undefined, AnyObject, undefined, "">
    .required( msg: 'Password is required') StringSchema<NonNullable<string | undefined>, AnyObject, undefined, "">
    .matches(
      regex: /^.*(?=.{8,})((?=.*[!@#$%^&*()\-_=+{};:,<.>]){1})(?=.*\d)((?=.*[a-z]){1})((?=.*[A-Z]){1}).*$/,
      options: 'Password must contain at least 8 characters, one uppercase, one number and one special case character',
    ),
  confirmPassword: yup
    .string() StringSchema<string | undefined, AnyObject, undefined, "">
    .required( msg: 'Confirm your password') StringSchema<NonNullable<string | undefined>, AnyObject, undefined, "">
    .oneOf( arrayOfValues: [yup.ref( key: 'password')], message: "Passwords don't match."),
});
```

Рисунко 3.10 – Схема валідації даних для форми реєстрації користувача

На формі реєстрації користувачу необхідно заповнити всі наявні поля, у при спробі надіслати дані з порожніми полями буде показано помилки зі спеціальним текстом біля кожного відповідного поля (рис. 3.11).

Також, якщо введені користувачем дані не проходять вимоги, описані у схемі валідації, буде також показані помилки біля відповідних полів (рис. 3.12).

Для імені та прізвища користувача немає обмежень по кількості літер. Для полів електронної пошти та пароля обмеження співпадають з формою авторизації.

Введені дані у поле підтвердження паролю мають збігатися з даними, які користувач увів до поля вводу паролю.

За умови успішної валідації даних на клієнтській частині буде відправлено запит на реєстрацію користувача до серверної частини з подальшим додавання цих даних у базу даних.

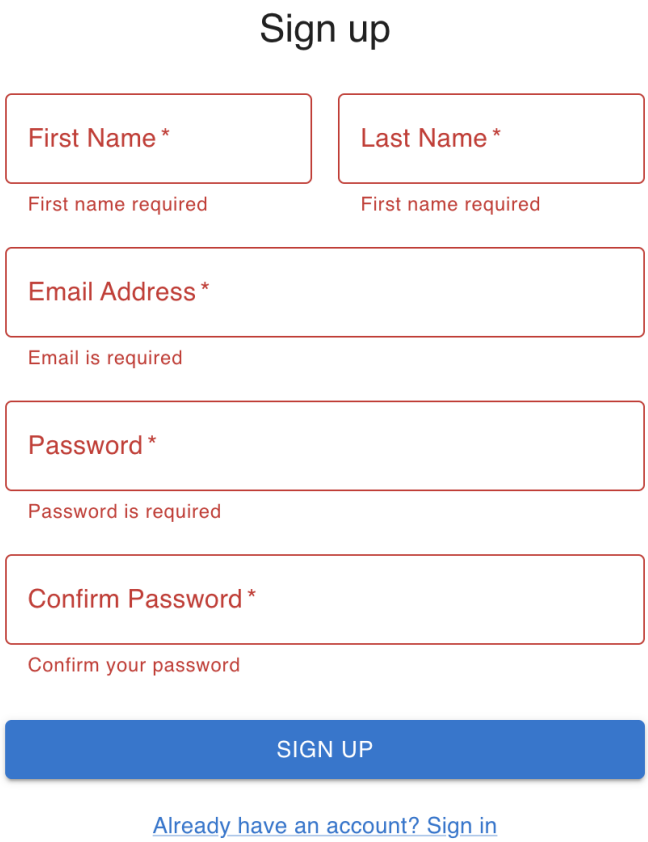


Рисунок 3.11 – Вигляд форми, якщо користувач не ввів дані

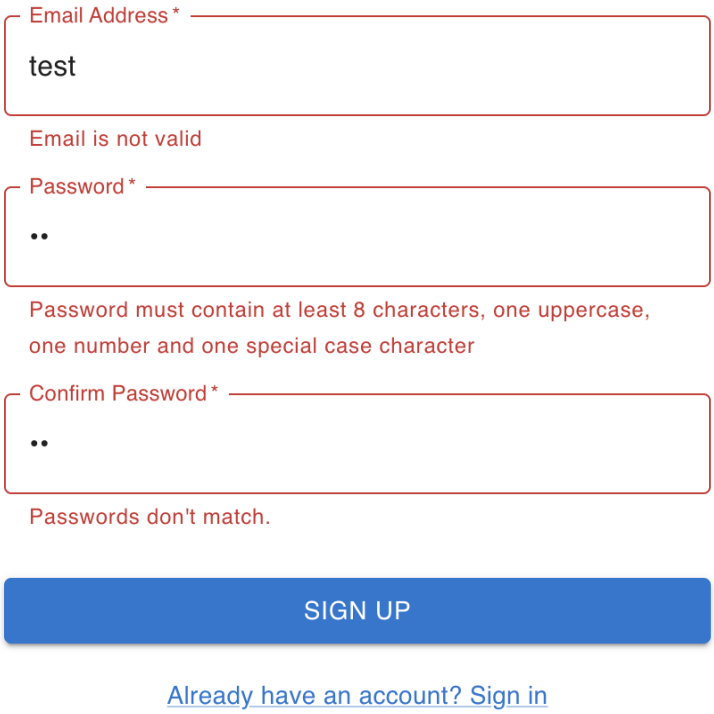


Рисунок 3.12 – Вигляд форми, якщо користувач ввів некоректний формат даних

Після успішної авторизації користувача, буде відкрито головну сторінку застосунку (рис. 3.13). На ній відразу зображено форму для покрокового проходження процесу міграції плейлистів між сервісами для відтворення аудіофайлів. Це зроблено для того, аби користувачу було зручно відразу розпочати роботу з функціоналом, для використання якого користувач відкрив застосунок.

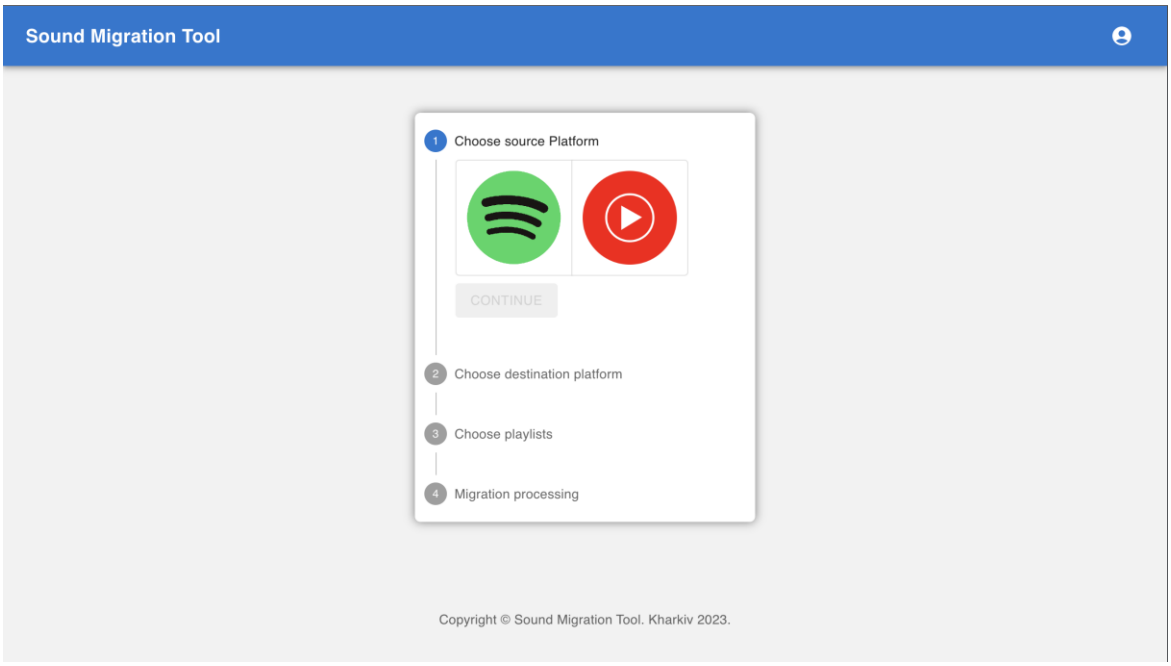


Рисунок 3.13 – Вигляд головної сторінки застосунку

На першому кроці користувач має вибрати з якого сервісу необхідно виконати міграцію плейлистів. В даний момент у застосунку підтримуються два головних сервіса: Spotify та YouTube Music.

Слід зазначити, що у майбутньому планується розширення списку підтримуваних сервісів длля міграції плейлистів.

Для вибору необхідного сервісу потрібно натиснути на відповідну іконку сервісу, після чого зміниться вигляд кнопки для обраного елементу (рис. 3.14).

Другим кроком у процесі міграції є обрання сервісу в якому будуть створені нові плейлисти (рис. 3.15).

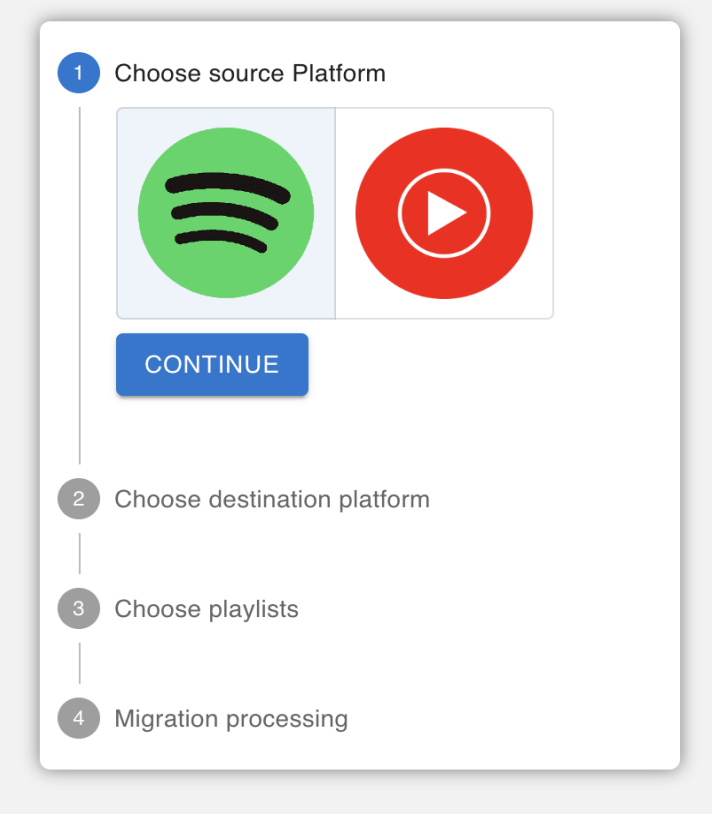


Рисунок 3.14 – Вигляд обраного сервісу на першому кроці міграції

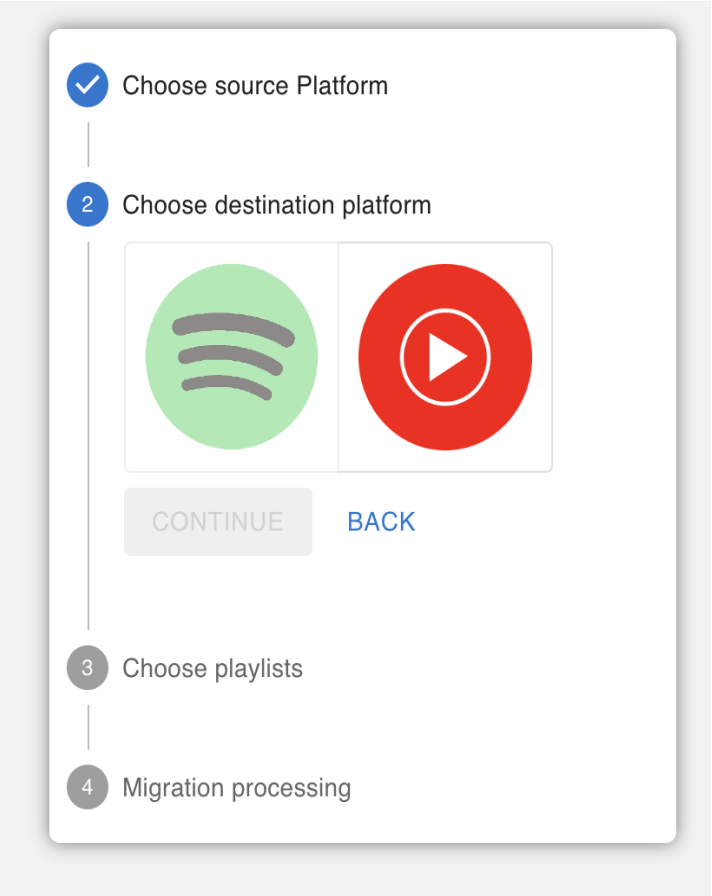


Рисунок 3.15 – Загальний вигляд другого кроку процесу міграції

На цьому етапі одна з опцій для вибору буде недоступної, в залежності від обраної опції на першому кроці. Бо міграція плейлиста в рамках одного сервісу неможлива та не відповідає головній меті застосунку. При наведені курсору на опцію, що була обрана на першому кроці, буде показане повідомлення, що цей сервіс вже було обрано користувачем (рис. 3.16).

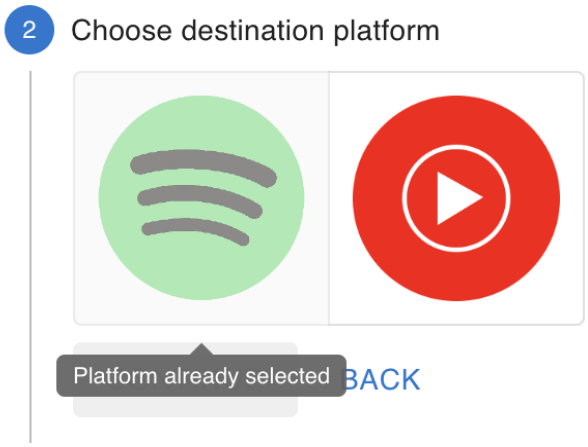


Рисунок 3.16 – Повідомлення що показане при наведенні курсору на опцію, що була обрана на першому кроці

При переході на третій крок процесу міграції відразу здійснюється відправка запиту на серверну частину застосунку для отримання списку всіх доступних плейлистів користувача у сервісі, обраному на першому кроці.

Поки йде обробка запиту, на клієнтській частині відображається іконка завантаження, аби користувач розумів що відбувається певний процес роботи застосунку (рис. 3.17).

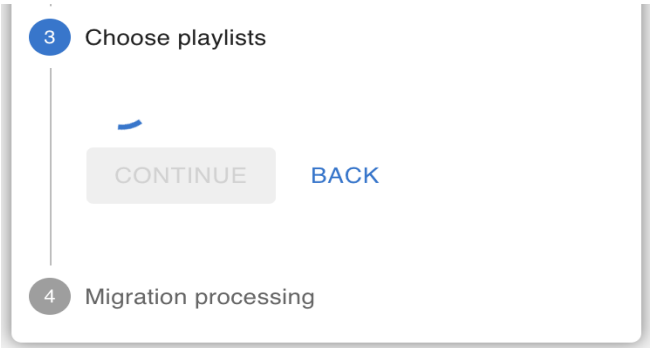


Рисунок 3.17 – Вигляд частини динамічної іконки завантаження

Після того як запит на отримання наявних плейлистів користувача потрапляє на сервер, відбувається відправка запиту на сторону відповідного обраного сервісу з токеном авторизації користувача використовуючи ендпоінт наданий API сервісом. Якщо валідація даних та запиту на сервісі пройшла успішно, серверна частина застосунку отримає список із моделей «Плейлист», описаних у другому розділі. Після чого буде відправлено на клієнтську частину оброблені та стиснуті дані про плейлисти користувача. І в результаті замість іконки завантаження буде відображено список наявних плейлистів користувача у обраному на першому кроці сервісі (рис. 3.18).

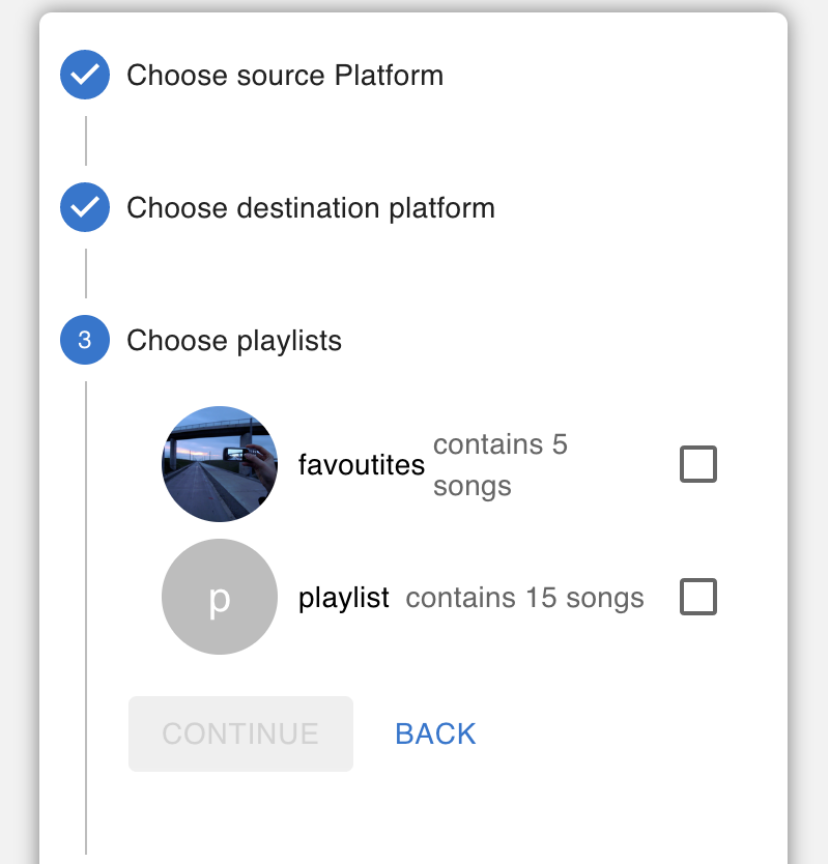


Рисунок 3.18 – Вигляд списку плейлистів користувача

На цьому кроці користувачу необхідно обрати хоча б один плейлист, але в одночас є можливість обрати всі доступні плейлисти (рис. 3.19). Після цього користувачу відкривається можливість перейти до наступного кроку.

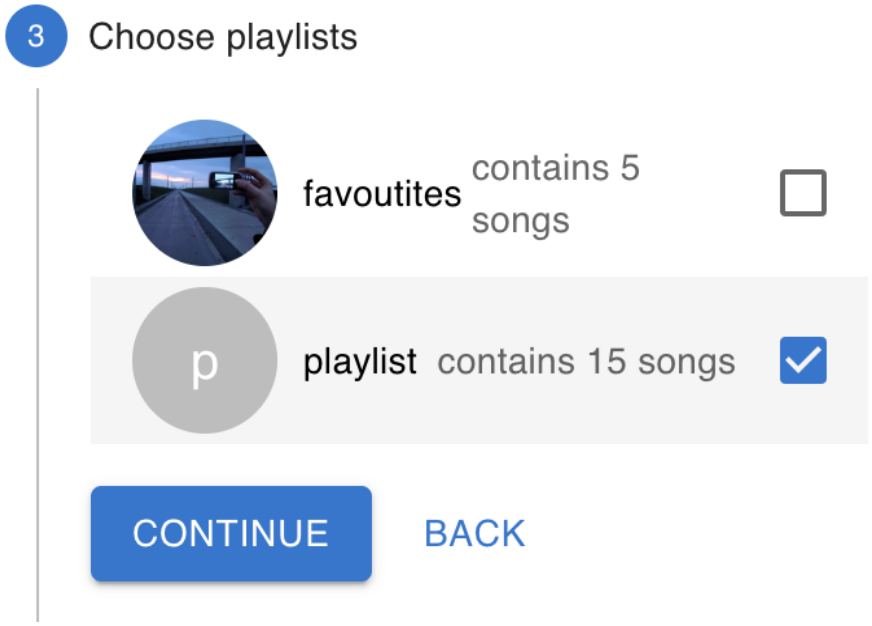


Рисунок 3.19 – Вигляд обраного плейлиста із доступного списку

На четвертому кроці користувач спочатку має можливість ознайомитись з обраними сервісом, в якому будуть створені плейлисти та зі списком обраних для міграції плейлистів (рис. 3.20). Користувач має бути впевненим у виборі, бо в ситуації якщо користувач помилився у виборі списку плейлистів для міграції та вирішить перервати процес міграції, вже створені до того плейлисти у застосунку будуть збережені.

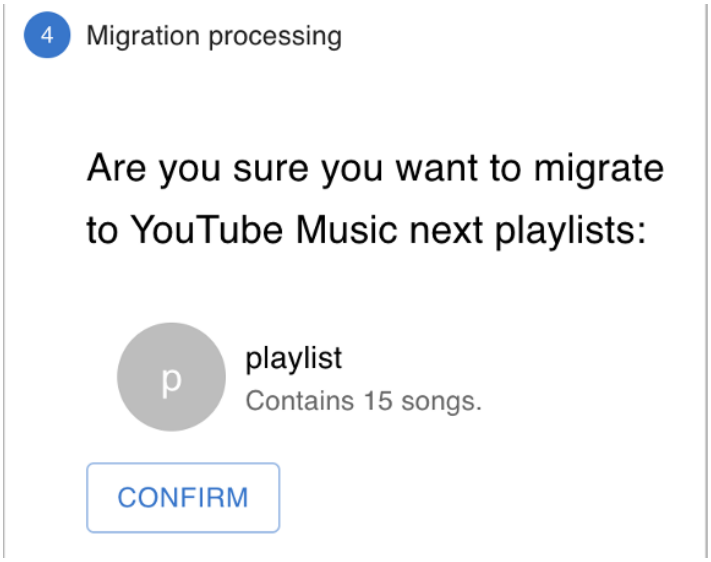


Рисунок 3.20 – Повідомлення про необхідність підтвердити обраний список плейлистів для міграції

Після підтвердження інформації на клієнтській частині застосунку буде відображено динамічний індикатор процесу міграції (рис. 3.21).

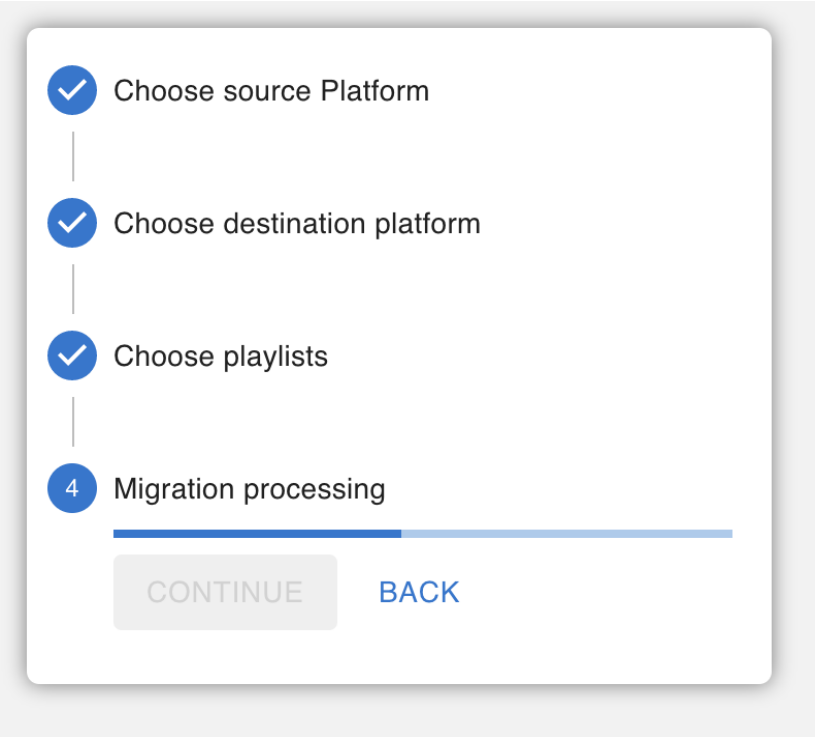


Рисунок 3.21 – Вигляд динамічного індикатора процесу міграції

Також з клієнтської частини буде відправлено запит на сервер виконання міграції плейлистів. На серверній частині після отримання списку плейлистів будуть по черзі створюватись плейлисти використовуючи відправку запитів до ендпоінтів наданих API обраного сервісу для відтворення музики. Після успішного створення плейлиста, сервер по черзі відправляє запит для знаходження аудіофайлу по назві на цьому сервісі. Якщо ауйдіофайл знайдено відбувається відправка запиту на оновлення створеного плейлиста з додаванням знайденого аудіофайлу.

Якщо в процесі виконання четвертого кроку виникає помилка будь-яка помилка на сервері, то на клієнт буде відправлено відповідь з повідомленням про помилку. Прикладом такої помилки є закінчення терміну дії токену авторизації користувача у обраному сервісі (рис. 3.22). Після обробки

відповіді з помилкою на клієнті буде відображено модальний діалог з текстом помилки, який було надіслано з серверу.

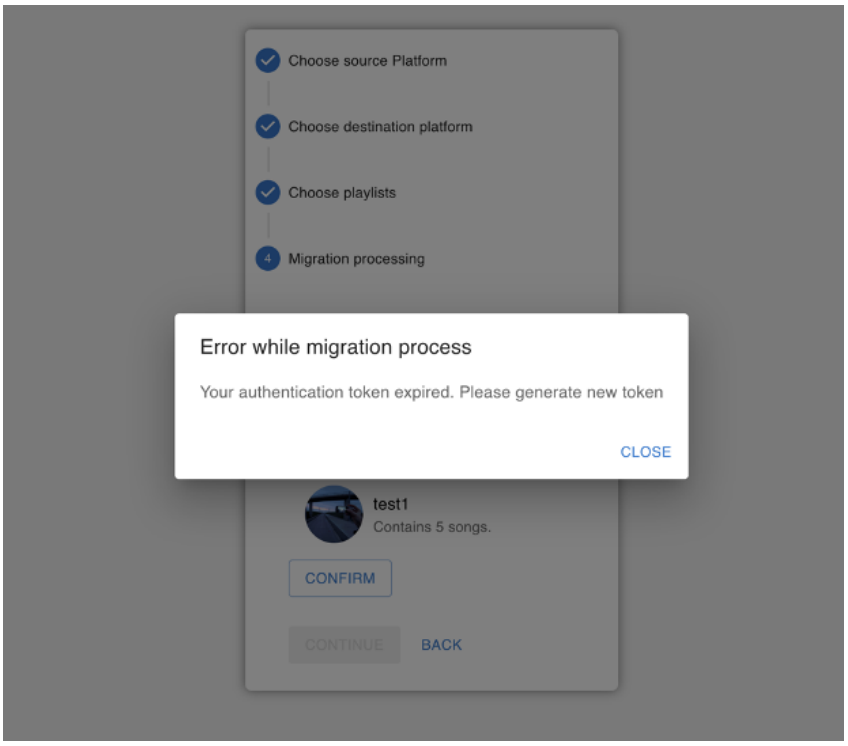


Рисунок 3.22 – Модальний діалог з помилкою, яка виникла у процесі міграції плейлиста

У випадку коли процес міграції завершився успішно і плейлисти було створено у необхідному сервісі для відтворення аудіофайлів, буде відображено відповідне повідомлення з можливістю розпочати процес наступної міграції (рис. 3.23).

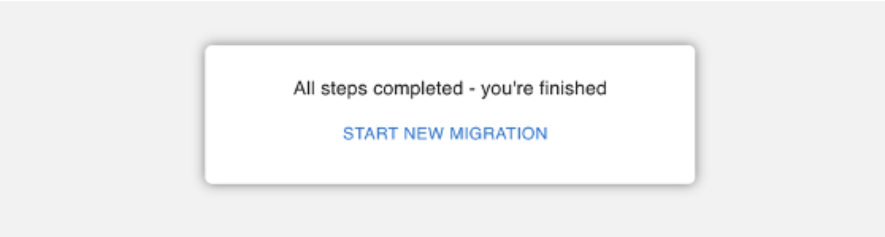


Рисунок 3.23 – Повідомлення про успішне завершення міграції плейлистів між сервісами

Також у застосунку у користувача є можливість використовувати навігацію у застосунку. При натиску на іконку користувача у хедері сторінці відкривається спливаюче вікно зі списком доступних переходів по сторінці: можливість переходу на сторінку «Profile» або можливість вийти з акаунту користувача (рис. 3.24).

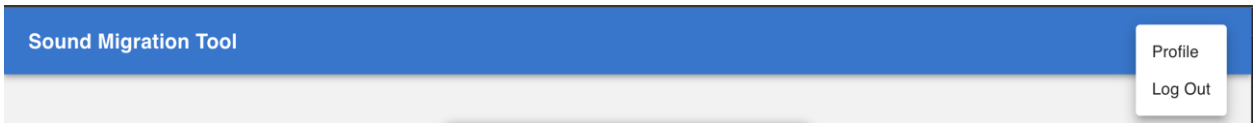


Рисунок 3.24 – Спливаюче вікно після натискання на іконку користувача

При натисканні «Log Out» буде закрито головну сторінку застосунку, видалено токен авторизації користувача у застосунку та відкрито форму авторизації користувача у застосунку.

При натисканні «Settings» буде відкрито сторінку на якій відображено дані про користувача: його ім’я та прізвище, а також інформація про згенеровані токени авторизації у доступних сервісах для відтворення аудіофайлів. У випадку відсутності токенів, будуть показані кнопки для подальшої генерації цих токенів на сервісах (рис. 3.25).

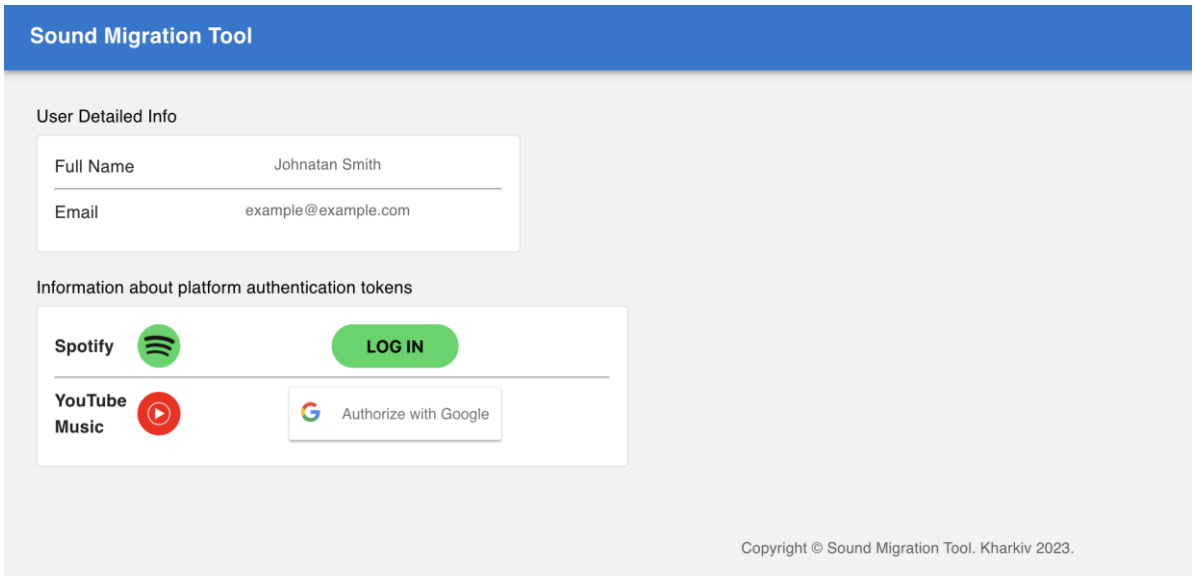


Рисунок 3.25 – Сторінка з інформацією про користувача та згенеровані токени авторизації у сервісах для відтворення аудіофайлів

3.3 Тестування розробленого застосунку

Тестування застосунку є важливою складовою процесу розробки програмного застосунку. Воно допомагає виявляти помилки та проблеми, забезпечує високу якість коду та знижує ризик непередбачених наслідків у готовому застосунку.

Unit тести допомагають перевірити, чи працюють окремі частини коду правильно, як очікується. Це допомагає виявляти помилки та проблеми ще на ранніх етапах розробки, що знижує ризик їх виникнення у фінальному продукті. Також unit тести допомагають виявити помилки при зміні коду, коли новий код може вплинути на роботу вже написаних функцій.

End-to-end тести дозволяють перевірити роботу програми в цілому, включаючи взаємодію з користувачем, роботу з базою даних, зовнішніми сервісами та іншими компонентами. Це дає можливість перевірити, чи працює застосунок в реальному середовищі, а також це допомагає виявляти проблеми зі зв’язком між компонентами та роботою застосунку в цілому.

Тестування допомагає забезпечити високу якість програмного забезпечення та знижує ризик виникнення проблем та помилок у фінальному продукті. Використання unit та end-to-end тестів допомагає забезпечити стабільну та надійну роботу програмного застосунку в умовах реального використання.

3.3.1 Написання unit тестів

У дані роботі для написання unit тестів використовується бібліотеки React Testing Library. Цей дуже важливий етап у загальному процесі розробки застосунку у якому інтерфейс користувача реалізовано за допомогою фреймворку React. Це дозволяє перевірити функціональність компонентів, забезпечуючи надійність та стабільність застосунку.

Однією з головних переваг використання React Testing Library полягає в тому, що ця бібліотека розроблена для цілей тестування компонентів на рівні їх функціональності, яку потенційний користувач бачить та взаємодіє з нею.

Такий підхід надає можливість перевірити, чи працюють компоненти так, як очікує користувач, замість перевірки даних які поширюються всередині компонентів, що були використані в окремий проміжок часу. Це забезпечує більшу гнучкість та стійкість до змін в коді, оскільки при переробці коду не потрібно змінювати структуру та логіку unit тесту.

У застосунку для зручного тестування загального вигляду сторінки або окремого компонента було використано snapshot тестування. Розробка застосунку із використанням тестування за допомогою снепшотів(англ. snapshots) дозволяє забезпечити відслідковування змін у візуальному вигляді компонентів.

Снепшот-тестування дозволяє зберегти вигляд, тобто HTML код, компонента після процесу рендерингу з урахуванням будь-якої взаємодії зі сторони користувача. І після автоматичного збереження вигляду компоненту або цілої сторінки з’являється можливість перевіряти його при кожному тестуванні. Якщо фактичний вигляд компоненту або сторінки не збігається зі збереженим снепшотом, тест буде провалено, що дозволяє вчасно виявити помилку що призвела до зміни зовнішнього вигляду компонента. А також це дає можливість легко перевіряти, як компонент змінюється під час переробки чи вдосконалення програмного коду.

Таким чином, написання unit тестів з використанням React Testing Library та снепшот-тестування є важливою складовою у забезпеченні надійності та стабільності застосунку, в розробці якого було використано фреймворк React. Це дозволяє вчасно виявляти та виправляти помилки, а також зменшує час, який потенційно може бути витрачений на ручне тестування. Приклад звіту про успішне виконання тестів на рисунку 3.26.

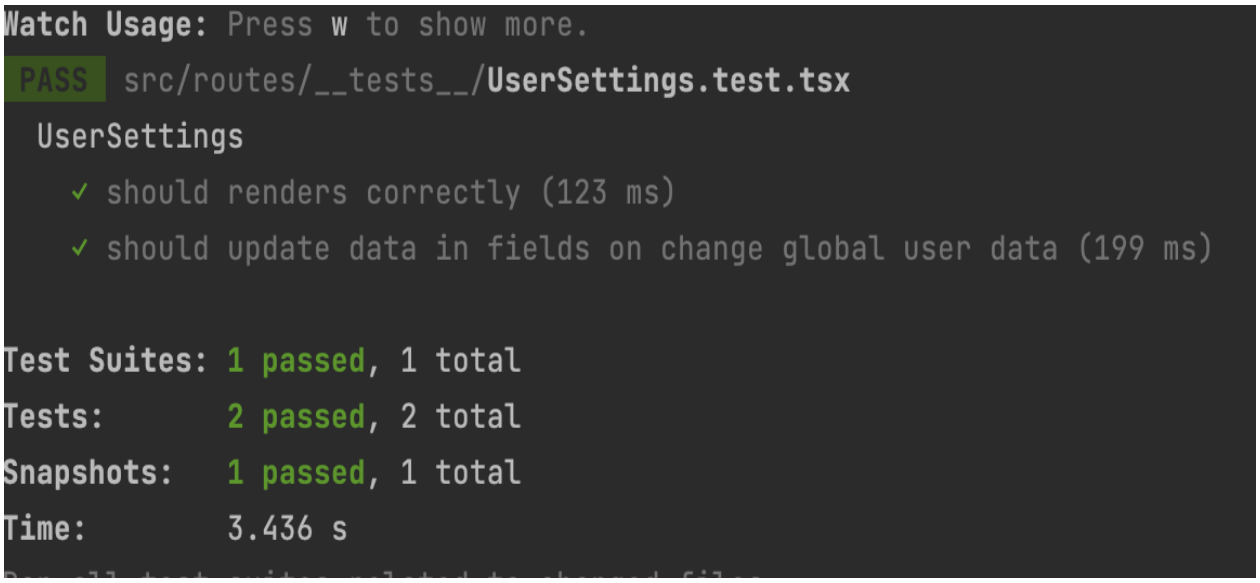


Рисунок 3.26 – Приклад результату успішного виконання unit-тесту використовуючи react testing library

3.3.2 Написання end-to-end тестів

В даній роботі end-to-end тести допомагають перевірити взаємодію між інтерфейсом користувача та серверною логікою обробки даних, переданих з клієнтської частини. Тому беручи до уваги, що у застосунку є дві ключові моделі, взаємодії з якими потрібно перевірити – це користувач та плейлист, необхідно у end-to-end перевірити процес реєстрації та авторизації користувача у застосунку програми. Також, іншою моделлю є плейлист, для цього необхідно перевірити можливість відправки необхідних запитів з клієнтської частини на сторону обробки надісланих даних про готовий для відправки плейлист.

Але, як і у випадку з unit тестами, необхідно також перевірити валідацію та можливість застосунку реагувати на негативні сценарії поведінки користувачів на стороні серверної обробки даних.

Для прикладу розглянемо декілька сценаріїв пов’язані з реєстрацією та авторизацією користувача. У випадку якщо користувач, який ще не зареєстрований у застосунку, буде намагатися увійти у застосунку, за умови

що дані з інтерфейсу користувача пройшли валідацію на допустимі символи, буде відправлено у відповідь на запит авторизації помилку з кодом 404, тобто користувача з заданою електронною поштою не знайдено у базі даних.

В іншому сценарії на етапі реєстрації користувач відправляє дані на сторону сервера, після успішної валідації на допустимі символи на стороні клієнта. Але електронна пошта, яку було вказано під час реєстрації вже існує у базі даних. У цьому випадку у відповідь на запит про реєстрацію користувача буде надіслано помилку з кодом 400.

У випадку, коли користувач вже зареєстрований та авторизований у застосунку є можливість використання головного функціоналу програми – міграції плейлистів.

Проте, якщо при відправці першого запиту на отримання загального списку плейлистів з кількістю аудіофайлів користувач не має згенерованого токена авторизації для необхідних сервісів для відтворення аудіофайлів або строк дії токену вже завершився, у відповідь на цей запит буде надіслано помилку з кодом 400.

Отже, всі ці описані сценарії було розроблено з використанням сучасного фреймворку для написання end-to-end тестів Cypress.

Cypress – це відкрите програмне забезпечення для автоматизованого тестування вебзастосунків. Використання Cypress для автоматизованого тестування має кілька переваг, таких як швидкість, простота використання для реалізації тестів та налаштування роботи, що дозволяє розробникам швидко та ефективно створювати тести для вебзастосунків.

Також застосунок Cypress має вбудовану підтримку для налагоджування тестів, що дозволяє досліджувати проблеми, які виникають під час виконання тестів, і швидко їх виправляти. Приклад звіту про успішне виконання тестів реалізованих з використанням Cypress на рисунку 3.27.

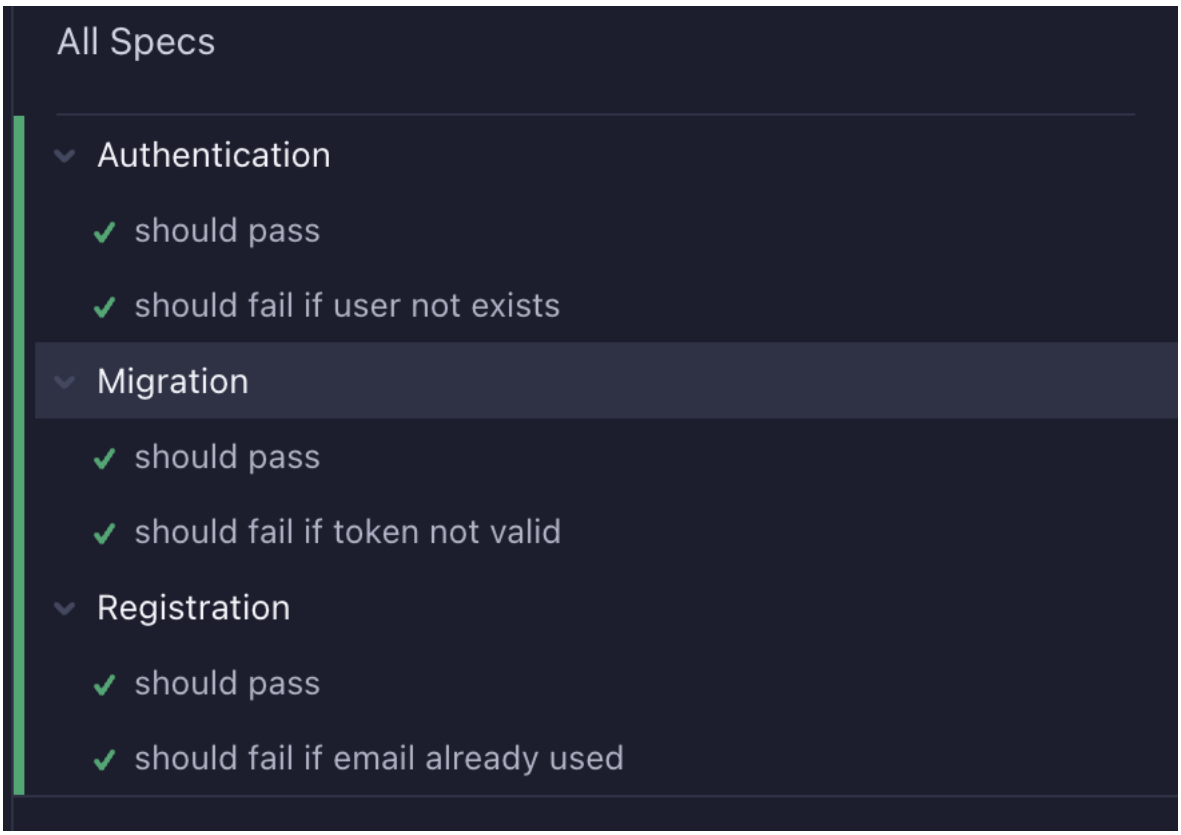


Рисунок 3.27 – Приклад результату успішного виконання end-to-end тестів у фреймворці Cypress

ВИСНОВКИ

У рамках кваліфікаційної роботи було розроблено і реалізовано застосунок для міграції плейлистів між сервісами для відтворення аудіофайлів. Головними перевагами реалізованого застосунку є сучасний та інтуїтивно зрозумілий інтерфейс користувача, а також безкоштовність використання можливостей, в той час коли в альтернативних застосунках необхідно оформлювати обов’язкову передплату.

Для цього були вирішені такі завдання:

– розроблено та реалізовано інтерфейс користувача з використанням фреймворку React та мови програмування typescript;

– розроблено серверну частину застосунку для обробки та подальшої передачі/отримання даних використовуючи наданий API сервісами для відтворення аудіофайлів;

– розроблено можливість поширення даних між стороною інтерфейсу користувача та серверної обробки даних з використанням архітектурного стилю REST;

– створено базу даних для збереження інформації про зареєстрованих у застосунку користувачів з використанням СУБД PostgreSQL.

Для залучення більш широкої аудиторії користувачів потрібно розширити функціонал, який надає користувачам застосунок. Серед можливих варіантів є створення єдиної бази для прослуховування аудіофайлів без залежності від конкретного одного сервісу. Виходячи з існуючої проблеми пов’язаної з тим, що на деяких сервісах для прослуховування музики можуть бути недоступні окремі аудіофайли, через політику прав сервісу, а в той час на іншому сервісі цей аудіофайл доступний для використання, є попит на прослуховування всіх можливих аудіофайлів без прив’язки до одного конкретного сервісу.

Також у майбутньому у застосунку можна додати можливість онлайн прослуховування аудіофайлів. Всі сучасні сервіси для відтворення музики надають API для досягнення цієї мети.

Результати роботи опубліковано у вигляді тези доповіді під час Міжнародного молодіжного форуму «РАДІОЕЛЕКТРОНІКА І МОЛОДЬ У XXI СТОЛІТТІ» [1].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Босенко А.М. (2023). Розробка застосунку для інтеграції сервісів відтворення та організації аудіофайлів. *Радіоелектроніка та молодь у XXI столітті: 27 міжнародний молодіжний форум*. С. 92-93.

2. Тітов С. В., & Тітова О. В. (2014). Інформаційно-освітнє середовище навчального закладу: розвиток засобів і способів комунікаційної й інформаційної взаємодії. Вісник Харківської державної академії культури, (43), 144-150.

3. Тітов С. В., & Тітова О. В. (2015). Оцінка юзабіліті освітніх сайтів: методи і технології. Вісник Харківської державної академії культури. Серія: Соціальні комунікації, (47), 127-134.

4. About Soundiiz. URL: https://soundiiz.com/about (дата звернення 29.04.2023).

5. TuneMyMusic application. URL: https://www.tunemymusic.com/ (дата звернення 29.04.2023).

6. SongShift application. URL: https://songshift.com/ (дата звернення 29.04.2023).

7. Путятін Є. П., Гороховатський В. О., & Матат О. О. (2006). Методи та алгоритми комп'ютерного зору: навч. посіб. Харків: ТОВ «Компанія СМІТ».

8. Lyashenko V., Kobylin O., & Ahmad M. A. (2014). General methodology for implementation of image normalization procedure using its wavelet transform. *International Journal of Science and Research (IJSR),* 3 (11), pp. 2870-2877.

9. Gorokhovatskyi V., Gorokhovatskyi O., Yevgenyi P., & Olena P. (2018, August). Quantization of the Space of Structural Image Features as a Way to Increase Recognition Performance. In *2018 IEEE Second International*

Conference on Data Stream Mining & Processing (DSMP), (pp. 464-467). IEEE.

10. Gorokhovatskyi V.O., Tvoroshenko I.S., and Peredrii O.O. (2020). Image classification method modification based on model of logic processing of bit description weights vector. *Telecommunications and Radio Engineering*, 1 (79), pp. 59-69. DOI: 10.1615/TelecomRadEng.v79.i1.60.

11. Daradkeh Y.I., Tvoroshenko I., Gorokhovatskyi V., Latiff L.A., and Ahmad N. (2021). Development of Effective Methods for Structural Image Recognition Using the Principles of Data Granulation and Apparatus of Fuzzy Logic. *IEEE Access, 9*, pp. 13417-13428.

12. Тітов С.В., Тітова О.В., Чорна О.С. (2022). Опис нескоротних наборів ознак в приблизних множинах з використанням систем числення. *Збірник наукових праць Харківського національного університету Повітряних Сил. № 1(71)*, с. 106-110.

13. Mashtalir S., Mashtalir V., & Stolbovyi M. (2018, August). Representative Based Clustering of Long Multivariate Sequences with Different Lengths. In *2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP)*, pp. 545-548.

14. Kobylin O., Vyskrebentseva S., & Petrova R. (2019). Обробка даних, що містять пропуски в задачах кластеризації. *Системи управління, навігації та зв'язку. Збірник наукових праць, 5(57).*

15. Spotify get playlist documentation. URL: https://developer.spotify.com/documentation/web-api/reference/get-playlist (дата звернення 30.04.2023).

16. Lyashenko V., Matarneh R., Kobylin O., & Putyatin Y. (2016). Contour detection and allocation for cytological images using Wavelet analysis methodology.

17. Sitnikov D., Titova O., Minukhin S., Kovalenko A., Titov S. (2018). Informativity of Association Rules from the Viewpoint of Information Theory.

Conference: *2018 IEEE International Scientific-Practical Conference Problems of Infocommunications*. Science and Technology.

18. Kinoshenko D., Kobylin O., Mashtalir S., & Stolbovyi M. (2019, March). Metric video retrieval speedup by irrelevant data elimination. In *Eleventh International Conference on Machine Vision (ICMV 2018)* (Vol. 11041, pp. 176-183). SPIE.

19. List of 10 Best Front end Frameworks to Use For Web Development. URL: https://www.monocubed.com/blog/best-front-end-frameworks/ (дата звернення 30.04.2023).

20. Кобилін О. А., & Творошенко І. С. (2021). Методи цифрової обробки зображень.

21. Lyashenko V., Mohammad A., & Kobylin O. (2015). Experiments with Fusion of Images with Use of Wavelet Transformation in Problems of the Text Information Analysis.

22. What is Gradle documentation. URL: https://docs.gradle.org/current/userguide/what_is_gradle.html (дата звернення 01.05.2023).

23. What is REST API article. URL: https://www.redhat.com/en/topics/api/what-is-a-rest-api (дата звернення 01.05.2023).

24. Multitier architecture article. URL: https://en.wikipedia.org/wiki/Multitier_architecture (дата звернення 03.05.2023).

25. REST API Security principals. URL: https://www.ibm.com/docs/en/inventory-visibility?topic=security (дата звернення 03.05.2023).

26. Get current Spotify user playlists documentation. URL: https://developer.spotify.com/documentation/web-api/reference/get-a-list-of-current-users-playlists (дата звернення 03.05.2023).

27. YouTube Data API documentation. URL: https://developers.google.com/youtube/v3/getting-started (дата звернення 03.05.2023).

28. 5 Ways to share data between applications. URL: https://medium.com/codex/sending-data-between-applications-e08fb0028a71 (дата звернення 03.05.2023).

29. Plugins for Webstorm. URL: https://plugins.jetbrains.com/webstorm (дата звернення 03.05.2023).

30. What is the most popular database in the world. URL: https://www.c-sharpcorner.com/article/what-is-the-most-popular-database-in-the-world/ (дата звернення 05.05.2023).

31. PSQL Tool. URL: https://www.pgadmin.org/docs/pgadmin4/latest/psql_tool.html (дата звернення 05.05.2023).

32. About Material UI library documentation. URL: https://mui.com/about/ (дата звернення 05.05.2023).