Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Факультет інформатики

Кафедра мультимедійних систем

РОЗРОБКА ВЕБ-ЗАСТОСУНКУ ІЗ РЕКОМЕНДАЦІЯМИ НА БАЗІ МУЗИЧНИХ ІНТЕРЕСІВ/НАВИЧОК

Текстова частина до курсової роботи

За спеціальністю 121 «Інженерія програмного забезпечення»

Керівник курсової роботи

ст.в. Вовк Н.Є.

(прізвище та ініціали)

_____

(підпис)

“__”_____ 2022 р.

Виконав студент Копійка В.Г.

(прізвище та ініціали)

_____

(підпис)

“__”_____ 2022 р.

Київ-2022

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри мультимедійних систем,

доцент, к.ф-м.н.

_____ О.П. Жежерун

(підпис)

“__”_____ 2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Копійці Вадиму Геннадійовичу

Факультету інформатики 4 р.н. бакалаврської програми

ТЕМА: Розробка веб-застосунку із рекомендаціями на базі музичних інтересів/навичок

Зміст ТЧ до курсової роботи:

Анотація
Вступ
Аналіз предметної області, теоретичні відомості й опис реалізації веб-застосування
Висновки
Список використаних джерел
Додатки

Дата видачі “___” _____ 2021 р. Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Тема: Розробка веб-застосунку із рекомендаціями на базі музичних інтересів/навичок

Календарний план виконання роботи:

№	Назва етапу курсової роботи	Термін виконання	Примітка
1.	Отримання завдання курсової роботи	11.10.2021	
2.	Пошук джерел за тематикою	20.12.2021	
3.	Ознайомлення з відповідною літературою	14.01.2022	
4.	Огляд існуючих аналогів/пропозицій	18.01.2022	
5.	Планування структури розділів текстової частини роботи	28.01.2022	
6.	Ініціалізація віддалених сховищ збереження проекту та налаштування середовища для розробки	30.01.2022	
7.	Узгодження функціональних можливостей веб-застосування та змісту роботи з керівником	14.02.2022	
8.	Проектування практичної частини	21.02.2022	
9.	Реалізація бекенд-частини застосування	05.04.2022	
10.	Написання окремих розділів роботи, їхнє узгодження з керівником	29.04.2022	
11.	Реалізація клієнтської частини веб-застосування	10.05.2022	
12.	Завершення написання розділів текстової частини курсової роботи	18.05.2022	
13.	Завершальне дооформлення курсової роботи	19.05.2022	
14.	Здача готової курсової роботи на перевірку	20.05.2022	

Студент Копійка В.Г.

Керівник Вовк Н.Є.

“__”_____ 2022 р.

Зміст

Анотація

У даній курсовій роботі описано перебіг розробки веб-застосування, яке вміщує в собі рекомендаційні можливості для користувачів на базі спільних музичних інтересів та навичок. Успішне створення такого застосунку й було метою курсової роботи. Під час її виконання було розглянуто необхідні для розуміння аспекти предметної області та потреби цільової аудиторії, аби зрозуміти очікування та надати якомога корисніший для них продукт.

Також у роботі розглянуто сучасні інструменти та підходи до розробки веб-застосунків. Було проведено аналіз вимог до продукту та зроблено обґрунтований вибір потрібних технологій. Описана реалізація кожної із частин застосування, що створювалося за допомогою відповідних інструментів: React і NextJS для розробки клієнтської частини та NodeJS-фреймворку Koa для серверної відповідно.

Як наслідок, у результаті роботи над курсовою було отримано веб-застосування, яке здатне зацікавити меломанів своїм рекомендаційним аспектом та стати в нагоді як під час знаходження людей за спільними музичними вподобаннями, так і під час пошуку нової музики.

Ключові слова: веб-застосування, рекомендації, музичні сервіси, клієнт, сервер, веб-розробка, фреймворки.

Перелік залучених скорочень

ARPANET – (від англ. Advanced Research Projects Agency Network) - мережа агентств передових дослідницьких проектів.

WWW – (від англ. World Wide Web) – світова веб-мережа.

UI – (від англ. User Interface) – інтерфейс для користувача

FCP - (від англ. First Contentfull Paint) – перше відмалювання контенту.

TTI – (від англ. Time To Interactive) – час до першого реагування на взаємодію.

CSR – (від англ. Client Side Rendering) – відмальовування зі сторони клієнту.

SSR – (від англ. Server Side Rendering) - відмальовування зі сторони серверу.

БД – база даних.

DBMS – (від англ. Database Management System) – система керування базою даних.

API – (від англ. Application Programming Interface) – програмований інтерфейс застосунку.

REST – (від англ. Representational State Transfer) – передавач репрезентативного стану.

SOAP – (від англ. Simple Object Access Protocol) – простий протокол доступу до об’єкта.

SQL – (від англ. Structured Query Language) – мова структурованих запитів.

Вступ

Актуальність теми та її практична значущість

Запит на розробку веб-застосувань стає дедалі більшим, адже за офіційною статистикою працевлаштування США попит на веб-розробників може збільшитися на 13% у період від 2020-го до 2030-го року. Це в середньому найвищий показник серед усіх наявних вакансій, за якими здійснюється спостереження [1]. Така тенденція не здається дивною, адже майже будь-яка компанія, що надає послуги або реалізує певну продукцію, потребує гарної веб-репрезентації у мережі Інтернет. Більше того, значна кількість успішних компаній взагалі не може уявити своє існування та прибутки без мережі та веб-застосувань, зокрема ті самі Netflix, Amazon або Facebook. Зародження теперішніх особливостей мережі, без яких людство не може уявити життя та роботу, почалося ще в далеких 60-х роках з обмеженої для певного кола користувачів мережі ARPANET. Вона була розроблена для об'єднання науково-дослідницьких центрів США, які мали укладені контракти з Міністерством оборони країни, в одну мережу, яка дозволяла здійснювати обмін даними між під'єднаними до неї системами [2]. У доступному ж для всіх вигляді, який нам сьогодні відомий, мережа Інтернет з'явилася в 1991 році разом із появою першого веб-сайту, що був створений Тімом Бернерсом-Лі для опису власного проекту під назвою WWW. Вже тоді компанія CERN, на потужностях якої розташовувався сервер із зазначеним вище сайтом, намагалася запатентувати наробки Тімоті, але той відхилив таку ініціативу, адже бажав зберегти цей проект загальнодоступним для всіх охочих. Він був впевнений, що лише таким чином ця розробка матиме майбутнє, адже буде підтримана набагато ширшим колом ентузіастів [3].

Відтоді світ почав невпинно змінюватися й веб-застосунки вже тривалий час відіграють провідну роль у наданні людям різноманітних сервісів та майданчиків для взаємодії між собою. Більше того, вони продовжують ставати все складнішими, адже очікування користувачів із плином років аж ніяк не згасають. На початках

сайти складалися щонайбільше з тексту та супутніх картинок, сьогодні ж застосунки пропонують набагато ширші можливості, включаючи миттєве листування, обмін файлами та отримання потокового медіа, але на цьому збагачення функціоналу зовсім не закінчується.

Останнім часом все більше застосунків соціальних мереж та стрімінгових (від англ. streaming - трансляція) платформ починають додавати ще одну функцію. Вона безпосередньо пов’язана зі створенням рекомендацій для власних клієнтів на базі дій, що ті здійснють через акаунт, використовуючи той чи інший сервіс. До таких рекомендацій передусім належать пропозиції статей на новинних ресурсах, фільмів, серіалів або музики на медійних платформах та людей у соціальних мережах. Наявність такого функціоналу неабияк впливає на користувачів, а саме на їхню зацікавленість у сервісі та частоту відвідувань.

Наприклад, розглядаючи успіх шведської компанії Spotify, що має однойменне застосування для прослуховування музики, можна переконатися в тому, що гарна рекомендаційна система може дозволити стати одним з найкращих сервісів, незважаючи на домінування інших на ринку. Компанія й надалі продовжує інвестувати в цю систему, таким чином приваблюючи ще більше клієнтів із кожним роком [4]. Більше того, Spotify спостерігає за неабияким зростанням власної аудиторії, адже лише за один квартал 2021 року вона збільшила свою базу клієнтів на 22%, нараховуючи вже щонайменше 365 мільйонів слухачів по всьому світові [5].

Незважаючи на те, що Spotify набирає все більшу популярність, він досі піддається критиці через відсутність деяких функцій. Найбільше користувачі очікують, що сервіс стане щонайменше більш соціальним. Вони хочуть мати можливість напряму ділитися своїми музичними вподобаннями з друзями, не покидаючи застосунку. Також вони хочуть бачити й інших користувачів, що мають схожі смаки, для здійснення нових знайомств [6]. Таким чином, їхня мрія - це

застосунок, що мав би в собі основні можливості соціальної мережі та особливості Spotify.

Оскільки сьогодні важко переоцінити розповсюдженість та невід'ємність веб-застосувань, а популярність музичної теми серед сервісів з рекомендаціями продовжує невпинно зростати, було вирішено поставити за мету створення веб-застосування з рекомендаціями на базі спільних музичних інтересів між користувачами.

Структура роботи

Курсова робота має в собі такі складові: зміст, анотація, вступ, основна частина та висновок до всіх наявних розділів, загальний висновок, список джерел інформації та додатків.

Основна частина складається з трьох розділів.

У першому розділі буде проаналізовано потреби, що мають користувачі до музичних майданчиків, а також розглянуто гарний приклад веб-застосунку, що поєднує деякі особливості соціальної мережі та музичного сервісу. До того ж, будуть окреслені основні види сучасних веб-застосувань, зокрема SSR та CSR, та те, як вони влаштовані.

У другому розділі будуть згадані популярні інструменти для розробки сучасних веб-застосувань, зокрема фреймворки, та рекомендаційних двигунів. Буде обґрунтовано вибір технологій та підходів відповідно до вимог застосунку, розглянуто їхні особливості.

У третьому розділі буде здійснено аналіз технічного завдання, визначено функціонал та головні особливості застосування. Буде зроблений опис поступової розробки частин всього застосування із супутніми рисунками та додатками.

Постановка задачі

1. Здійснити аналіз потреб клієнтів музичних сервісів, а також особливостей сучасних веб-застосунків та того, як вони можуть бути влаштовані.

2. Розглянути найпоширеніші технології серед веб-розробки, обґрунтувати вибір інструментів для реалізації веб-застосування.

3. Зробити поступовий опис реалізації клієнт-серверного веб-застосування з рекомендаціями на базі музичних інтересів, спираючись на визначені раніше особливості та обрані раніше технології.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз очікувань користувачів від сучасних музичних сервісів. Розгляд вдалої пропозиції

Усе більшої значущості для суспільства музика стала набувати минулого століття. Розвиток технологій призвів до того, що пісні ставали доступнішими й згодом взагалі почали супроводжувати наше повсякдення, зокрема серед молоді. Поява доступної для всіх мережі Інтернет взагалі змінила галузь музичного мистецтва, а особливо його розповсюдження серед людей. Тепер будь-які композиції можна послухати, попередньо завантаживши їх з мережі на власний компактний пристрій, наприклад, смартфон.

Ще не так давно більшість людей здебільшого слідкувала за світом музики лише під час прослуховування радіо або перегляду музикальних телепрограм. Згодом розпочалася ера цифрових магазинів медіа, де здійснювалася купівля як окремих пісень, так і цілих альбомів. Їхня популярність свідчила про те, що час компакт-дисків, а особливо касет, із записаними піснями почав спливати. Успішним початківцем у цьому бізнесі став насамперед майданчик iTunes від Apple. На цей магазин лише у період з моменту релізу перших пристроїв iPod припали мільярди продажів різного цифрового контенту. На жаль, останнім часом згадки про легендарний iTunes все рідше з’являються в новинах, а в соціальних мережах про нього взагалі майже забули. Причина такої зміни вподобань користувачів та згасання популярності полягає в появі стрімінгових (від англ. streaming - потоковий) сервісів [7]. На нових майданчиках з’явилася можливість слухати композиції від улюблених артистів без жодних обмежень. Для того, аби відкрити доступ до всіх пісень, достатньо лише створити акаунт й купити пакет послуг на певний проміжок часу. Тож людям припало до душі здійснювати плату, наприклад, раз на місяць, отримуючи всю бібліотеку. Це назавжди змінило світ музики як для слухачів, так і артистів, адже тепер люди стали набагато частіше слухати й

знаходити нові для себе композиції. Як не дивно, але навіть цього стає недостатньо для сучасних користувачів.

Сьогодні соціальні мережі стали невід'ємною частиною взаємозв'язків між людьми. Обмін думками, розповсюдження інформації та звичайне спілкування через них набуло небачених масштабів. Це стало чимось обов'язковим у житті більшості. Саме тому користувачі очікують, що будь-які сервіси повинні мати принаймні мінімальний перелік можливостей, які ті звикли бачити в соціальних мережах. Як наслідок, люди мають надію, що платформи для прослуховування музики на кшталт Spotify та Deezer також почнуть надавати більше можливостей, які б дозволили їм покращити взаємодію з іншими користувачами. До досить бажаних входить як функція обміну музикою, так і можливість знаходити нових людей за спільними жанровими вподобаннями, розширивши акаунти користувачів [6]. На жаль, зараз досить мало гарних платформ, які б могли запропонувати такий функціонал. Команда Spotify хоч і намагається наздоганяти тренди серед власних клієнтів, надалі розвиваючи рекомендаційну систему, випускає оновлення з новими можливостями набагато рідше за постійну й не таку потрібну зміну інтерфейсу.

Новими настроями серед людей і затримкою нововведень на існуючих музичних сервісах вдало скористалися розробники Vampr Inc. Вони розробили майданчик, який дозволив музикантам і слухачам перебувати в тісному взаємозв'язку між собою. Таким чином, у людей зі спільними смаками в музиці з'явилася змога знаходити один одного в цій мережі й продовжувати спілкування як у соціальних, так і кар'єрних цілях. Також сервіс Vampr надає змогу користувачам "прив'язувати" акаунти з платформ YouTube та SoundCloud, аби така інтеграція дозволила їм легше ділитися не тільки піснями відомих артистів, а й власними доробками. Головною метою цієї платформи є полегшення пошуку музикантам нових людей для співпраці, аби творити спільними зусиллями та, як наслідок, потім об'єднуватися в групи. Це також чудова мережа для звичайних

любителів знаходити щось нове у світі музики, адже тут є можливість слідкувати за новинами від артистів, котрі знаходяться найближче за геолокацією [8].

Vampr – це достатньо гарний приклад платформи, яка задовольнила потреби користувачів музичних майданчиків. Вона вдало поєднала популярні функції окремих соціальних мереж та сервісів для прослуховування музики. Про успіх цього сервісу можна було бути впевненними від початку стартап-кампанії в 2019 році. Лише за годину розробники отримали весь мінімальний внесок, який вказали на платформі Wefunder. Трошки згодом вони отримали кошти від двох тисяч інвесторів, яких зацікавила ідея Vampr, а це вже перевищувало максимально дозволений внесок, який платформа може прийняти за проміжок часу у дванадцять місяців [8]. Про акутальність такого сервісу свідчить і його популярність серед користувачів різних мобільних платформ. Лише в цифровому магазині застосунків Google Play кількість завантажень перевищує позначку в п’ятсот тисяч [9]. Більше того, компанія Apple під час щорічного визначення найкращих сервісів зазначила Vampr в офіційному списку переможців. Це все спровокувало додаткові надходження інвестицій в проект від найбільших компаній світу [8], що остаточно й закріпило успіх такого сервісу. Тож Vampr став гарним прикладом для всіх майбутніх майданчиків, які матимуть дотичну до музичної галузь та аудиторію.

1.2 Розгляд сучасних стратегій до розробки веб-застосувань

Нещодавно насиченість веб-ресурсів у мережі Інтернет обмежувалася статичною репрезентацією текстової інформації з мінімальним вкрапленням іншого медіа. Про інтерактивність не йшло й мови, а адаптивність обмежувалася здебільшого зміною розміру шрифтів на сторінці. Зараз веб-сайти не тільки стали більш наповненими різними елементами та стилізованістю, а й почали перетворюватися у повноцінні застосування, за допомогою яких можна здійснювати найрізноманітнішу роботу. Тепер вони можуть бути повноцінними платформами для надання різних послуг, починаючи від сервісу для бронювання готелів та квитків і закінчуючи платформою для одночасної роботи над документами в реальному часі на кшталт Booking та Office 365 відповідно.

Від усіх веб-застосувань очікується відповідна працездатність і можливості, адже всі вони мають різну мету й повинні задовольняти потреби власних клієнтів на належному рівні. Через це в світі веб-розробки з’явилося декілька стратегій до рендерингу (від англ. rendering - відмальовування) контенту на сайтах. Слід одразу пояснити значення слова «ренедринг» у контексті таких стратегій, адже його можна переплутати з рендерингом графічних елементів сторінки рушіями браузерів Gecko та WebKit у Firefox та Google Chrome відповідно. У згаданому ж рендерингу передусім йдеться саме про генерування (від англ. generate - створення) HTML-розмітки сторінок, всіх їхніх тегів та вузлів. Тож серед основних типів створення веб-контенту виділяють два: CSR (Client Side Rendering), що стосується клієнту, та SSR (Server Side Rendering), за який відповідає сервер. Кожна така стратегія має сильні й слабкі сторони в задоволенні тих чи інших потреб як розробників, так і користувачів.

Відмальовування на стороні клієнта полягає в тому, що HTML створюється безпосередньо у веб-переглядачі на пристрої користувача. Усі компоненти сторінки генеруються внаслідок виконання коду мови JavaScript в браузері. Задля того, аби

користувач побачив повноцінну сторінку потрібно завантажити необхідний для відображення елементів бандл (від англ. bundle – набір) коду. Покликання на цей JavaScript-код знаходиться в секції “scripts” файлу HTML (HyperText Markup Language), котрий одразу отримують користувачі, звертаючись до веб-сайту в браузері, зокрема за прямим URL (Uniform Resource Locator). Отриманий документ вміщує в собі мінімально необхідну інформацію для того, аби в браузера була можливість відмальовувати весь сайт із супутніми сторінками. Таким чином, здебільшого все, що написано в такому файлі, - це покликання на index.js певного фронтенд-фреймворку. Такий файл часто називають контейнером застосування, адже в ньому потім відбувається відтворення всього вмісту веб-застосунку [10].

```
<html>

 <head>
   <title> CSR App </title>
 </head>

 <body>
     <div id="app"> </div>
     <script src="../src/index.js"> </script>
 </body>

</html>
```

Рисунок 1.1 - Приклад CSR-файлу, який відіграє роль контейнера [10]

Отже, браузер ні від кого не отримує жодну додаткову HTML-розмітку, котру можна було б обробити в браузері та відобразити. Клієнт на пристрої користувача, зокрема коли здійснюється якась навігація по сайту, кожного разу самостійно генерує нову сторінку внаслідок перероблення DOM (Document Object Model). Всі ці необхідні «вказівки» для зміни сторінки наявні в завантаженому JavaScript-коді каркасів на кшталт React або Vue, а переміщення визначено у відповідних навігаційний секціях цих фреймворків. Тож всі необхідні для належного функціонування сайту дії відбуваються в браузері користувача.

Такий підхід, як і будь-який інший, має певні недоліки та переваги. Найголовнішою особливістю CSR часто виокремлюють швидку роботу

застосування, котра можлива після завантаження всього необхідного коду JavaScript. Багато розробників вважають, що це дозволяє створювати веб-застосунки, досвід користування якими збігається зі звичайними мобільними додатками. Так би мовити, необхідно почекати лише раз під час першого заванатження, аби потім користуватися застосунком без будь-яких візуальних затримок, зокрема під час навігації. На жаль, це одночасно є й недоліком, адже для роботи принаймні одної сторінки, необхідно завантажити весь застосунок. Це може неабияк вплинути на досвід користування веб-сайтом як під час повільного Інтернет-з'єднання, так і в разі його відвідування на дуже слабких пристроях. Тож такий підхід можна назвати повною протилежністю принципу «тонкого клієнта», коли від клієнтської частини не вимагається майже нічого, крім наявності мережі, адже швидкість першого відмалювання контенту FCP (First Contentful Paint) у такому разі залежить більшою частиною від його потужності. До того ж, CSR викликає неабиякі труднощі для пошукових систем та SEO (Search Engine Optimization), адже «веб-павуки» пошуковика не зможуть проаналізувати вміст HTML-файлу, який не дає жодної корисної інформації без супутнього JavaScript.

Відмальовування контенту на стороні серверу – це повна протилежність підходу CSR. Під час використання SSR клієнтська частина зазвичай майже не виконує жодних дій, що пов'язані з побудовою переважної частини вмісту HTML-файлів. Таким чином, цей підхід передбачає, що компоненти сторінки генеруються на стороні серверу. У свою чергу, клієнтська частина отримує все для відображення вмісту веб-застосування, коли робить запит до серверу, на якому й відбувається рендеринг контенту. Тож все, що необхідно браузеру користувача, - це обробити отриманий документ із розміткою для його відображення користувачеві. Серверне відмальовування може відбуватися на різних етапах, котрі, якщо дозволяє фреймворк, може визначити розробник. Серверна частина застосування може здійснювати надсилання як попередньо підготовлених за певними параметрами сторінок, так і тільки-но згенерованих за запитом клієнта [10].

Оскільки SSR виступає повною протилежністю CSR, він має переваги там, де інший підхід - ні, і навпаки. Тож тут серед гарних якостей слід виокремити швидкий FP (First Paint) та FCP. Користувачеві не потрібно чекати повного провантаження веб-застосування, що допоможе пришвидшити початок споглядання та взаємодії з вмістом сайту, зокрема не очікуючи на великі обсяги допоміжного JavaScript [11]. Таким чином, на клієнт часто покладені значно менші вимоги стосовно роботи з отриманими від сервера даними, що дозволяє йому бути менш потужним, ніж під час використання CSR. Пошукові системи легко працюють із серверним рендерингом, оскільки ті здатні проаналізувати вміст HTML-сторінки, зокрема тегів, без необхідності виконувати JavaScipt-код. На противагу є й відповідні недоліки. SSR-застосунки не можуть запропонувати такої «безшовності» під час навігації, як ті, що відмальовуються на стороні клієнту, адже під час таких дій необхідно кожного разу робити запит до серверу й чекати на відповідь. Це інколи неабияк сповільнює досвід користування, якщо мова йде про «чистий» SSR.

Усі згадані особливості кожної зі стратегій обов’язково слід враховувати, перш ніж починати розробку нового проекту. Несвідомий вибір CSR або SSR може призвести до того, що під час надання користувачам уже готового веб-застосування стане зрозуміло, що було обрано помилковий підхід, котрий погано впливає на визначену цільову аудиторію.

1.3 Висновки до розділу 1

У цьому розділі розглянуті шляхи, за допомогою яких люди отримували доступ до прослуховування пісень, було згадано про зміни вподобань меломанів до цифрової музики та платформ її розповсюдження з плином часу. Також було проведено аналіз веб-застосунку Vampr, який припав до душі користувачам, адже той гарно поєднав у собі тренди сучасних соціальних мереж та сервісів для прослуховування музики.

Було здійснено аналіз головних сучасних стратегій для відмальовування контенту веб-застосувань, а саме CSR та SSR. Розглянуті їхні особливості, котрі визначають їхнє призначення для тих чи інших обставин і потреб зі сторони користувачів та розробників.

РОЗДІЛ 2. ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО ТЕХНОЛОГІЇ

2.1 Вибір технологій для розробки клієнтської частини

2.1.1 Огляд каркасів для клієнтської розробки та обрання оптимального

Важливий вибір під час розробки веб-застосувань завжди припадає на інструменти для створення їхньої клієнтської частини. Саме вона в будь-якого застосування виступає візитівкою, адже впливає на перше і, можливо, найголовніше враження в користувачів. Навряд чи зараз є ті, хто буде із задоволенням користуватися ПЗ, яке не має належного інтерфейсу, навіть якщо застосування дозволяє гарно виконувати потрібну роботу. Тож фреймворк (від англ. framework - каркас) для фронтенду (від англ. front-end – зовнішній, кінцевий інтерфейс), який дозволить якомога зручніше створити гарний користувацький інтерфейс, - це один із найперших кроків перед початком розробки, який слід ґрунтовно обміркувати.

Будь-які фреймворки в галузі програмування – це набір ПЗ, який вміщує в собі підготовлені окремим колом розробників рішення, що дозволяють прискорити розробку іншим програмістам. Цей набір вміщує в собі вже готовий код, котрий надає розробникам можливість не вигадувати власні стратегії, логіку або графічні елементи, використовуючи вже стандартизовані окремими фахівцями наробки.

Коли мова йде про створення інтерфейсу для веб-застосунків, то він завжди складається з таких трьох складових: HTML-розмітки, CSS-стилів та JavaScript-коду. Кожна з них відіграє окрему роль у візуалізації клієнтської частини застосування. HTML-розмітка – це те, завдяки чому вибудовується структура веб-сторінки, що вміщує різного роду семантичні елементи, які допомагають браузеру та пошуковим системам краще розуміти її вміст. CSS-стилі відповідають за надання згаданим вище компонентам HTML певної стилізації, що стосується найрізноманітніших параметрів, починаючи від кольорів і закінчуючи адаптивністю сторінки до різних розмірів веб-переглядача. Код JavaScript

відповідає за забезпечення функціональності елементів сторінки, їхніх змін та реагування на певні дії зі сторони користувача [12]. Для того, аби робити будь-які веб-застосування, необхідно працювати з кожною складовою й писати багато чого з нуля, що часто призводить до повторюваного коду. Саме тому використання фреймворків – це важлива частина процесу розробки програмістами, робота яких поставлена в чіткі часові межі.

Внаслідок швидкого розвитку насиченості й можливостей веб-сайтів, постійно з'являються нові фреймворки для розробки їхньої клієнтської частини. Серед найпопулярніших каркасів на кінець 2021 року, за статистикою StackOverflow, виділять передусім такі: Angular, React та Vue [13]. Усі вони мають «відкритий» код та, що цікаво, використовують мову JavaScript. Причина її використання полягає в тому, що браузерні рушії досі здатні сприймати лише її код, а тому будь-якому фреймворку, що використовує іншу мову, слід спершу проводити конвертування [14]. Така особливість може неабияк уповільнити роботу застосування, адже виконується зайва дія з «перекладу на браузерну». Тож здебільшого саме через цю особливість ці популярні фреймворки побудовані на JavaScript.

Почнемо короткий огляд фреймворків з Angular. Він був створений компанією Google і вперше презентований світові ще в 2010 році [15] Хоч сьогодні його популярність позаду згаданих вище конкурентів [16], він все одно залишається одним із найпоширеніших інструментів для розробки фронтенду. Він широко використовується в створенні великих веб-застосувань, адже опирається на MVC (Model-View-Controller) в структурі проекту, що неабияк сприяє логічному розподіленню його коду. Більше того, підтримується підхід «two-way binding» даних, що дозволяє одразу відображати зміни на репрезентації моделі, якщо та зазнала будь-яких змін. До того ж, цей фреймворк підтримує звичне бекенд-розробникам (від англ. back-end – серверна, «прихована» частина) вживлення залежностей на кшталт того, що є в каркасах .Net для C# та Spring для Java. Також

до пакетів Angular входять готові рішення для забезпечення належної безпеки, зокрема автентифікації та авторизації, що неабияк заощаджує час реалізації. Тож усі ці особливості неабияк вплинули на поступовий ріст спільноти навколо Angular. Популярність його репозиторію на GitHub хоч і досі достатньо велика, вже значно менша за згаданих конкурентів [16]. Причиною в цьому часто виокремлюють недолік каркасу, що стосується відсутності допоміжної віртуальної DOM для сторінки. Вона виступає спрощеною копією справжньої моделі, за допомогою якої здійснюється швидка зміна окремих елементів в інших фреймворках. Це дозволяє запобігти зайвому перемалюванню всієї DOM сторінки, звертаючи увагу лише на ті компоненти, що зазнали змін [17]. Через відсутність такої особливості з'являються й інші супутні проблеми, що можуть стосуватися як сповільнення рендерингу сторінки, так і загального погіршення користувацького досвіду. Попри це, Angular досі посідає поважну сходинку серед найкращих фронтенд-фреймворків.

React – це дещо молодший за Angular каркас, який був розроблений і випущений компанією Meta (у минулому Facebook Company) у 2013 році [18]. Він вбачався в ролі основної технології для створення інтерфейсів власних продуктів, зокрема до них входять такі відомі мережі, як Facebook та Instagram. За опитуванням Stack Overflow, яке було проведено в серпні 2021 року, частка популярності фреймворку React визначається сорока відсотками, де Angular займає близько тридцяти, Vue – двадцяти, а залишок припадає на всіх інших відповідно [17]. Більше того, за статистикою постів, що також була зібрана цим форумом, кількість згадок фреймворку за тегом "reactjs" перевищує будь-що серед схожих технологій на платформі Stack Overflow [19].

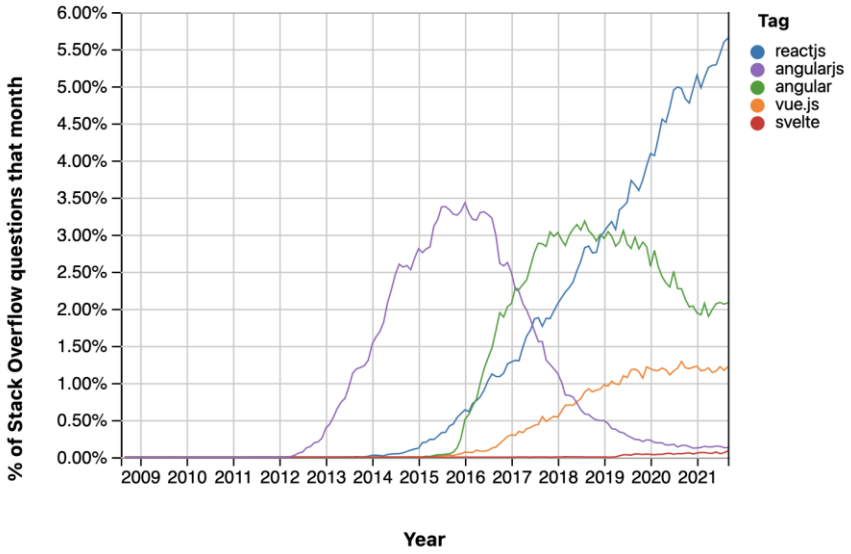

Рисунок 2.1 - Інфографіка Stack Overflow щодо популярності згадувань фреймворків на форумі [19]

Причина такого успіху полягає в його початкових перевагах, зокрема він був першим каркасом, який запропонував компонентну розробку веб-застосувань, що дозволяла зручно розбивати код компонентів сторінок застосування. До того ж, робота React базується на взаємодії із віртуальною DOM, чого часто не вистачає в Angular. Як уже було згадано, ця допоміжна модель є набагато «легшою» копією реальної. Зміни, що мають бути застосовані до елементів сторінки, спочатку відображаються лише на ній, а отриманий результат потім напряму порівнюється із структурою справжньої моделі. Під час цього порівняння всі відмінності будуть визначені алгоритмом фреймворку і застосовані лише до тих місць реальної DOM, де відповідно відбулися зміни у віртуальній. Серед недоліків найчастіше виокремлюють те, що написання складних проектів на React часто є причиною досить заплутаної архітектури застосувань, зокрема через відсутність підтримки шаблонів проектування MVC або MVP (Model-View-Presenter). До того ж, його часто сварять за відсутність уже готових рішень, що стосуються безпеки або побудови компонентів. Через це його часто називають бібліотекою, а не повноцінними фреймворком, адже більшість зручних можливостей надають саме

допоміжні пакети від спільноти, які потрібно обирати й завантажувати власноруч [14].

Vue – це наймолодший представник серед згаданих фронтенд-фреймворків, що був випущений у 2014 [20] ролі альтернативи для React та Angular. Цікаво, що Vue в багатьох аспектах нагадує поєднання деяких особливостей цих двох фреймворків, адже він користується віртуальною DOM для здійснення змін компонентів, підтримує Two-way Binding даних та базується на реалізації MVC. Хоч популярність цього каркасу продовжує невпинно зростати, налічуючи близько вісімсот тисяч щотижневих завантажень на npm [21], він має один досить суттєвий для розробників недолік. Vue має ту ж саму проблему, що й React, у сенсі відсутності готових рішень. Фреймворку від Meta тут допомагає велике коло розробників, набагато більша популярність та дещо довша присутність на ринку. Тут в аналогу гірші позиції, адже слід бути готовим, що більшість з того, що існує в React та Angular, доведеться писати з нуля. Більше того, Vue не може похизуватися рівнем якості документації та регулярними оновленнями, що мають конкуренти від великих IT-компаній, а це ще більше сповільнює розробку на ньому [14].

Тепер, коли нам відомі головні особливості найпопулярніших фреймворків, слід проаналізувати їхні показники швидкості в роботи з елементами структури DOM-сторінки. Це дозволить визначити, наскільки респонсивним (від англ. responsive – чуйний) в порівнянні є користувацькі інтерфейси, що були написані на React, Angular та Vue. Ефективність роботи зі змінами компонентів сторінки безпосередньо впливає на те, яку плавність і безвідмовність досвіду отримають кінцеві користувачі. Для того, аби порівнювати показники, слід мати однакові умови під час тестування кожного з фреймворків. За здійсненим аналізом в роботі Матіаса Левліна [22], що була проведена за дотримання рівних умов середовища, можна побачити, що кожний із фреймворків інколи «виходить уперед» в тих чи інших операціях, що пов’язані з DOM. Загальне тестування проведено в такі три

етапи: додавання, зміна та видалення елементу зі структури моделі. Цікаво, що коли мова йде про проведення маніпуляції з великою кількістю елементів, наприклад, коли треба замінити у кожному з десяти тисяч блокових елементів «div» текст його абзацу, то з-поміж трьох фреймворків завжди перемагає саме React із невеликим відривом від Vue. Такий показник легко пояснюється наявністю віртуальної DOM у цих каркасах, котрий допомагає уникнути повного перероблення реальної моделі. У результуючій нижче таблиці (рисунок 2.2), котра підсумовує загальні результати всіх проведених тестувань фронтенд-каркасів, можна з легкістю переконатися в тому, що саме React двічі продукує найкращий результат з-поміж усіх, жодного разу не демонструючи найгірший.

	React v16.12.0	Vue v2.6.11	Angular v8.2.14	Svelte v3.20.0
Add 10,000 (ms)	30.96	25.36	52.75	31.26
Edit one (ms)	16.58	22.23	6.08	0.11
Edit 10,000 (ms)	17.86	20.64	896.76	885.03
Remove one (ms)	16.54	24.51	0.09	0.53
Remove 10,000 (ms)	7.39	33.33	23.83	22.97
Compilation (s)	3.96	3.07	8.70	1.61

Рисунок 2.2 – Отримані показники швидкості фреймворків за Маттіасом Левліном [22]

Його кращі показники підтверджують те, що React, попри очевидні недоліки у вигляді відсутності підтримки дизайн-шаблонів на кшталт MVC, залишається найоптимальнішим вибором для тих проектів, що матимуть одночасно відображати велику кількістю змін на сторінці. Молодший Vue впевнено його наздоганяє. У той час коли йдеться про роботу з одним індивідуальними елементом на сторінці, то перемога завжди дістається Angular, хоча показники в компіляції проекту та масивних маніпуляціях постійно найнижчі.

Отже, кожний із розглянутих каркасів має сильні та слабкі сторони. Як наслідок, кожний окремий розробник або команда має визначити найоптимальніший для себе вибір, який гарно підпаде під певні потреби. Тут слід

враховувати як загальну працездатність написаних на них застосувань, так і час, котрий необхідний для їхньої розробки на React, Angular або Vue. Оскільки саме React використовувався під час розробки відомих мереж від компанії Meta та стрімінгового сервісу Netflix, а загальні показники працездатності UI (User Interface) на ньому досі вражають, можна впевнено обирати його в ролі інструмента для розробки фронтенду схожих застосувань. У нашому випадку веб-застосунок також володітиме особливостями соціальних мереж у поєднанні з аспектами музичного сервісу, тому React стане оптимальним вибором для розробки його користувацького інтерфейсу.

2.1.2 Вибір стратегії для рендерингу контенту, базуючись на React

React, як і багато інших аналогів, за замовченням надає змогу створювати застосування, що базуватимуться на залученні стратегії CSR. Таким чином, всі веб-застосунки, які пишуться на «чистому» фреймворку, насичуються вмістом лише шляхом виконання його бібліотеки коду на клієнтській частині, тобто веб-переглядачем на пристрої користувача.

Оскільки у React є багато допоміжних бібліотек і навіть окремих надбудованих фреймворків, його рідну особливість використання CSR можна з легкістю замінити. Для того, аби вирішити, чи є в цьому необхідність і обрати оптимальну стратегію відмальовування контенту, слід проаналізувати особливості таких протилежних підходів до рендерингу, як CSR та SSR, що були описані в першому розділі роботи. Відмальовування на стороні клієнту гарно підійде для застосувань, для яких показники FCP та TTI (Time To Interactive) не такі важливі, як і SEO-оптимізація, але дуже потрібно мати повністю завантажений застосунок, навігація по якому збігатиметься зі швидкістю офлайн-додатків. Тоді як рендеринг на серверній частині – це навпаки чудова альтернатива для тих, хто хоче надавати користувачам швидке перше підвантаження веб-сайту, бажає отримати достатньо швидкі й одночасні показники FCP та TTI, і переймається рейтинговим місцем у видачі пошукових систем. Раніше питання вибору стосувалося лише цих двох стратегій, але сьогодні існують і ті, що дозволяють отримати переваги кожної з них.

Творцям надбудованих над React фреймворків NextJS та Gatsby, вдалося поєднати кращі сторони CSR та SSR. Вони дозволили створювати застосування, котрі позбулися проблеми повільного FCP, адже під час першого звернення до них відбувається те ж саме, що й на SSR-застосунках. Тож клієнти спочатку отримують повністю підготовлену сторінку, а згодом і весь необхідний код для отримання переваг стратегії CSR. Таким чином, уникається відчуття провантаження веб-застосування, котре часто мають спостерігати користувачі, дивлячись на якусь іконку-очікування або напис «loading…» під час першого звернення до нього. Отже,

ці каркаси дозволили одночасно позбутися головних недоліків і отримати переваги раніше згаданих стратегій.

Тепер прослідкуємо за основними кроками, які відбуваються зі сторони надбудованих каркасів NextJS або Gatsby, які дозволяють досягти вище описаного результату.

По-перше, такі фреймворки відповідають клієнту відмальованою ними сторінкою, до якої відбулося безпосереднє звернення через веб-переглядач користувачем. Це також вирішує проблему, що стосується неможливості пошуковими системами аналізувати вміст CSR-застосування, адже тепер вони здатні отримати насичену контентом сторінку. У той час, коли користувач уже спостерігає повноцінну сторінку в браузері й може починати взаємодіяти з нею, одночасно підвантажується необхідний JavaScript-бандл. Він виконує ту ж саму роль, що й у звичайних застосунків на «чистих» React, Vue або Angular, котрі використовують стратегію відмалювання на клієнті за замовченням. При цьому підготовлена сторінка зовсім не стикається з проблемою блокування UI через одночасне завантаження JavaScript, адже вся вона була попередньо відмальована фреймворком.

По-друге, коли весь необхідний JavaScript був отриманий і виконаний, CSR-застосування повністю розгортається замість раніше отриманої сторінки від серверу. Процес, котрий це реалізує, називається гідрацією, саме він допомагає здійснити ефективне оновлення веб-сайту. Він передбачає перевикористання раніше отриманої SSR-шляхом сторінки замість повної генерації її копії. Фреймворк сканує існуючу структуру й додає у відповідних місцях очікуваний функціонал, тобто він займається прикріпленням усіх слухачів подій на певні елементи DOM сторінки. Для того, аби цей процес був успішно виконаний, потрібно отримувати одну й ту ж саму HTML-структуру як у готовій сторінці від сервера, так і в тій, що передбачається у завантаженому коді JavaScript. Якщо мова йде безпосередньо про фреймворк React, то в ньому цей процес викликається майже

однойменною функцією React.hydrate(), котрою зокрема і користуються NextJS та Gatsby [10].

Таким чином, саме завдяки гідрації вдається створити невидимий для користувачів перехід від отриманої від сервера статичної сторінки до веб-застосування з якостями CSR і, як наслідок, зробити можливим поєднання переваг обох стратегій.

Отже, для того, аби скористатися згаданими вище перевагами саме на React, слід відповідно обрати NextJS або Gatsby. Ці в багатьох речах схожі технології також мають відмінності. Основним є те, що NextJS пропонує більше різних підходів до використання стратегій рендерингу контенту. Найголовнішою особливістю є те, що Gatsby не передбачає можливості ISR (Incremental Static Regeneration) та DSSR (Dynamic Server Side Rendering) на відміну від конкурента. Перша можливість дозволяє навіть повністю статичним сторінкам, що створюються лише під час побудови проекту, автоматично оновлювати свій вміст. Це досягається за допомогою використання параметру перевірки під назвою «revalidate». Він надає змогу спровокувати автоматичне перезбирання вмісту статичної сторінки через певний проміжок часу. Розробник сам обирає сторінки, які мають слідувати ревалідації (від англ. revalidate – підтвердження відповідності, перевірка). Для того, аби досягти того ж результату на Gatsby, необхідно постійно перебудовувати проект, аби зміни були відображені на будь-якій статичній сторінці. Другий підхід дозволяє створювати попередньо побудовані сторінки з «пустими» полями на місці очікуваної інформації, для отримання якої буде зроблено відповідні дії під час прямого запиту зі сторони користувача. Таким чином, є можливість скоротити як час очікування для клієнта, так і навантаження на сервер, адже той не матиме робити повну генерацію сторінки й запит до якогось API одночасно [10].

Як наслідок, враховуючи вище згадані аспекти, вибір на користь фреймворку NextJS буде доречнішими. Його додаткові можливості дозволять гнучкіше

оптимізувати веб-застосування, адже буде можливість використовувати сторінки із різними варіаціями рендерингу контенту відповідно до потреб та мети кожної.

2.2 Вибір технологій для розробки серверної частини застосування

2.2.1 Бекенд та його відповідальності

Бекенд – це та частина будь-якого проекту, яку користувачі явно не бачать, але якість їхнього досвіду користування застосуванням напряму залежить саме від неї. На цій частині здебільшого відбуваються всі процеси, що насичують фронтенд даними. Якщо використовувати більш формальне визначення фронтенду та бекенду, то вони відповідають за окремі функціональні «прошарки», а саме за шари представлення та доступу до даних відповідно. Отже, до сфери відповідальності бекенду передусім належить надання відповідного API (Application Programming Interface) для роботи з даними. До нього й будуть здійснюватися звернення зі сторони фронтенду, через який користувачі неявно взаємодіють із сервером [23]. Для того, аби зрозуміти, за що відповідає серверна частина, слід перерахувати те, що здебільшого входить до її відповідальності. Вона передусім відповідає за обробку запитів за певними «шляхами» від клієнтської частини, а також за відпрацювання окремого коду для зміни сторінки, доступ до них та маніпуляцію над ними, шифрування та дешифрування чутливої інформації користувача тощо.

API – це набір певних правил, котрий дозволяє різному ПЗ «спілкуватися» один з одним. Ця комунікація може полягати в провокуванні найрізноманітніших дій зі сторони чи то застосунків, чи то сервісів у відповідь на конкретні запити до когось із них через API. Саме у цій взаємодії сторона, котра робить звернення до іншої для отримання якихось даних, відіграє роль клієнта, а та, котра відповідає на звернення, - роль серверу [24].

Серед веб-сервісних API виокремлюють такі протоколи, як SOAP (Simple Object Access Protocol), варіації RPC (Remote Procedure Call) і т.п. Особливість веб-сервісів передусім полягає в тому, що вони розташовуються в мережі Інтернет, тобто мають власні унікальні URL (Uniform Resource Locator) на просторах WWW

(World Wide Web). За такими адресами й відбувається надання певних послуг шляхом звернення до їхнього інтерфейсу. Якщо мова йде про популярний серед розробників REST (Representational State Transfer) API, то його часто не відносять до окремого протоколу. Він насамперед виступає в ролі набору певних архітектурних принципів [25], котрі визначають взаємодію між сервером та клієнтом, котрі не прив'язані до жодних пропрієтарних протоколів або форматів для передавання даних. Усього налічується 5 основних REST принципів, до яких входять:

a) уніформність інтерфейсу – ця особливість визначає, що вигляд одного й того самого запиту до API не має різнитися, незважаючи на те, звідки такий робиться;

b) розподіленість сереверу та клієнта – це риса, яка зазначає, що серверна та клієнтська частини мають бути незалежними один від одної. Таким чином, все, що має знати клієнт – це покликання для отримання того чи іншого ресурсу від серверу, а він, у свою чергу, не повинен жодним чином модифікувати клієнтську частину, окрім як передавати запитаний ним ресурс;

c) безстановість – це те, що визначає сервер як такий, що не повинен запам'ятовувати жодної додаткової інформації клієнта для обробки його запитів. Таким чином, звернення від клієнтської частини має містити всю необхідну інформацію для серверу, щоб він успішно пропрацював його

d) багатошаровість системи – це той аспект, що зазначає наявність різних функціональних шарів, на яких відбуваються певні дії. Таким чином, ні сервер, ні клієнт не мають знати, що вони напряму «спілкуються» між собою без жодних прошаркових сервісів чи застосунків;

e) кешованість - це те, що дозволяє серверу пришвидшити досвід користувача на клієнті, тобто деякі ресурси вже мають бути підготовлені для отримання без додаткових зусиль жодної зі сторін.

Тож особливість роботи систем, які побудовані на REST-архітектурі, полягає в реагуванні серверу на різні запити за стандартними методами HTTP (Hypertext Transfer Protocol) від клієнту, а зокрема таких: GET, POST, PUT та DELETE, з урахуванням вище згаданих принципів. Усі такі запити часто визначають окрему дію в БД сервером, тобто додавання, отримання, редагування або видалення наявних у ній даних [24]. Головним чинником, який вплинув на популярність принципів REST, є передусім явність даних, які передаються мережею під час взаємодії клієнту та серверу. На відміну від, наприклад, протоколу SOAP, у котрого дані мають бути обов'язково «запаковані» в XML (Extensible Markup Language), підхід REST передбачає передачу даних мережею у їхньому «чистому» вигляді без додаткового обгортання [25]. Тож є можливість передавати навіть звичайний текст або код. Більше того, за окремими URL-зверненнями до API можна отримати доступ до будь-яких файлів, зокрема різного медіа, якщо відповідно до них був відкритий доступ сервером.

Комунікація між сервером та клієнтом не обмежується забезпеченням вище згаданого API, котре базується на принципі запит-відповідь. Також слід згадати й про надання сервером постійного двостороннього «full-duplex» підключення для певних функцій застосування. Таке підключення стало можливим завдяки протоколу WebSocket, котрий, як і HTTP, за мережевою моделлю OSI (Open System Interconnection) працює на його найвищому рівні – рівні застосунку. Саме він і є «найближчим» до користувача, адже впливає на те, що бачить людина у інтерфейсі застосування [26]. Таке підключення використовується для миттєвого обміну різними даними. Саме завдяки йому з'явилася можливість для відображення повідомлень в чатах, спостереженні змін в редагуванні кооперативних файлів, наявності міток онлайн-офлайн в соціальних мережах і т.п. Кожна з перерахованих функцій має відображатися постійно й у реальному часі. Це неможливо здійснити без додаткових періодичних запитів з клієнту в разі використання чогось на кшталт REST API або SOAP, а тому у нагоді став саме протокол WebSocket.

2.2.2 Вибір мови/середовища для розробки бекенду

З огляду на вище описане, на бекенд покладено забезпечення різних аспектів, що передусім стосуються того, яким чином буде здійснено обмін даними між ним та його клієнтами. Тому для написання серверної частини треба обрати інструмент, який дозволить її реалізувати якомога ефективнішою та працездатнішою. Коли йдеться про мову, на якій слід писати бекенд, все частіше увага приділяється саме фреймворкам, які та може запропонувати розробникам. Кожна з мов, яка використовується для написання серверних частин, може надати власні унікальні можливості, починаючи з того, чи є мова багатопоточною або асинхронною і закінчуючи особливостями, що стосуються парадигмою програмування та наявністю строгої типізації. За статистикою, що була зібрана в опитуванні від Statistics&Data, у п’ятірці найпопулярніших бекенд-фреймворків знаходиться Laravel, Flask, Django, Express та Rails [27]. Якщо говорити про мови, які вони «представляють», то це PHP, Python, JavaScript та Ruby відповідно. Таким чином, можна бути впевненим, що на форумах кожної з цих мов можна знайти багато інших аналогів для написання бекенду, які жваво обговорюються та розвиваються.

Тож для того, аби розробка всього застосування була зручнішою, слід обрати той каркас, котрий гарно взаємодіятиме з раніше обраним фреймворком для реалізації представлення. Оскільки більшість бекенд-каркасів на мові JavaScript чудово працюють з усім, що пов’язано з React, то вибір на користь розробок саме цієї мови буде достатньо виправданим. До того ж, не доведеться здійснювати зайвих перемикань між підтримуваними функціями та технологіями, що може статися під час використання інструментів, що написані на інших мовах. Також слід зазначити й результати тестувань серверу, що написаний на JavaScript. Майже всі показники демонструють те, що його середовище виконання NodeJS гарно підходить для швидкої обробки більшості запитів у великих кількостях. Більше того, саме бекенд на JavaScript демонструє відчутний приріст «швидкості» у порівнянні з іншими під час збільшення потужності самого сервера, зокрема в разі

покращення його процесорних якостей та об’єму оперативної пам’яті [28]. Це свідчить про непоганий потенціал для масштабування працездатності застосувань, що написані на цій мові. Як наслідок, вибір бекенд-фреймворку саме на JavaScript цілком доречний.

Також слід зазначити, що каркас NextJS, який був раніше обраний для рендерингу контенту, вже може відігравати роль повноцінного веб-серверу. На ньому може бути написане API, до якого він самостійно буде звертатися під час відмальовування сторінок. Це дозволить поєднати «бази» фронтенду та бекенду в одну, але якщо мова йтиме про структурованість та розподіленість коду, то це навряд чи гарно відобразиться на проекті. Більше того, NextJS не надає можливості забезпечення «сокетного» підключення, а також відсутня зручна підтримка проміжних обробників, що присутня зокрема в Express [29]. Такі можливості можуть бути в нагоді під час розробки різних проміжних функцій для застосувань, що схожі на соціальні мережі. Як наслідок, буде доречно обрати окремий бекенд-фреймворк для реалізації функцій, які притаманні серверній частині застосування, зокрема для написання API.

Арсенал бекенд-фреймворків для NodeJS налічує десятки різноманітних пропозицій. До найпопулярніших належить уже згаданий Express, а також Koa, Meteor та Sails [30]. Оскільки NextJS відіграватиме роль серверу для рендерингу контенту, то для сторення бекенду, який надаватиме API, необхідно обрати технологію, яка матиме найкращі показники в сенсі обробки запитів. Серед вище згаданих фреймоворків гарні результати передусім демонструють Express та Koa [31]. Цікаво, що другий також базується на наробках Express, адже Koa – це новий проект розробників того ж каркасу [32]. Якщо казати про абсолютного переможця, то серед них найкращим виступає саме Koa, адже той встигає обробити більше п’яти тисяч запитів за секунду, коли навіть Express не долає позначку у чотири тисячі [31]. Як наслідок, обрання сучасного фреймворку Koa, котрий славиться загальною мінімалістичністю, стане гарним інструментом для створення

ненагромадженого API. Більше того, для застосування, котре налічуватиме багато користувачів, які робитимуть численні звернення, буде потрібний потенціал швидкості, котрий Koa може запропонувати.

2.2.3 Типи баз даних для використання у проектах

Сьогодні існує багато DBMS (Data Base Management System), тобто систем керування для створення різних баз даних та роботи з ними. Усі вони дозволяють працювати з БД, що належать до одного з двох основних типів: реляційного або нереляційного. Реляційні бази даних складаються з «реляцій», тобто табличок, котрі можуть бути взаємопов'язані між собою за певними полями-ключами. Кожна з таких табличок відповідає окремій сутності, кожний рядок якої – це екземпляр такої сутності. Кожна БД, що належить до такого типу, використовує певну SQL-мову, котра й дозволяє працювати з записами в ній. Саме SQL-подібні мови дозволяють скористатися «реляційністю» таких баз даних шляхом підтримання різних спеціальних операції. До таких передусім належить JOIN, котра об'єднує дані з двох таблиць за полями, які їх пов'язують, що дозволяє ефективно створювати складні запити. Особливість нереляційних баз даних навпаки полягає у відсутності обов'язкової схеми для структурування даних, вони не мають чітких табличок, які перебувають у зв'язку за якимись ключами, а володіють різними іншими якостями. До того ж, цей тип має різноманітні підтипи для вибору в тих чи інших умовах. До них належать стовпчикові, графові, документно-орієнтовані та ті, що базуються на основі наборів пар ключ-значення.

Усі пропозиції як серед реляційних, так і будь-яких нереляційних баз даних мають власні сфери для застосування. Кожна БД може запропонувати певну особливість, яка здатна вплинути на розробку будь-якого проекту. Наприклад, якщо мова йде про реляційні бази даних, як і про окремі нереляційні, то серед їхніх переваг часто виокремлюють саме сувору відповідність принципам ACID (Atomicity Consistency Isolation Durability):

a) atomicity полягає в недопущенні часткового виконання транзакції, що вміщує в собі певний набір операцій. Кожна окрема операція у транзакції має виконатися успішно, аби та рахувалася як така, що була здійснена. Інакше, коли принаймні один запит не пройшов, то взагалі жодний із них не буде виконаний над даними в БД;

b) consistency відповідає за забезпечення цілісності даних у відповідності до встановлених проектувальником обмежень та налаштувань;

c) isolation каже про те, що жодна з транзакцій не може зчитувати окремі дані з інших, що ще не були остаточно виконані, тобто жодна з транзакцій не може вплинути на іншу;

d) durability полягає у тому, що кожна успішно завершена транзакція має бути обов'язково збережена у БД [33].

Як наслідок, все ці принципи забезпечують цілісність і надійність даних, що зберігаються в реляційних БД. Коли ж йдеться про те, що привертає увагу розробників до нереляційних пропозицій, то це передусім їхня гнучкість та швидкість у тих чи інших аспектах. Наприклад, якщо йдеться про збереження великої кількості записів, що часто оновлюються, то досить доречно звернутися саме до нереляційної БД. Адже відсутність чіткого слідування принципам ACID, зокрема атомарності в здійсненні транзакцій, дозволяє швидше вносити зміни, хоч це й може призвести до загальної втрати коректності даних. У таких базах даних це можливо завдяки тому, що кожний новий запис – це унікальний за ідентифікатором екземпляр. Як наслідок, є можливість одночасно здійснювати маніпуляції над тими ж самим сутностями великій кількості користувачів, адже немає обмеження на паралельність операцій, що присутня в реляційних БД. У них, наприклад, для того, аби змінити запис-рядок, тобто екземпляр, в таблиці, необхідно зачекати поки таку операцію з такою самою таблицею завершить інший користувач, якщо той почав її раніше [34].

Отже, деякі особливості реляційних баз даних призвели до збільшення попиту на переваги «нереляційності» серед розробників. Більше того, майже всі ці аспекти пов’язані з тим, що робить реляційні БД зручнішими та надійнішими в інших умовах. Часто саме під час роботи з великими обсягами даних головним недоліком визначають саме те, що вони мають зберігатися у взаємопов’язаних таблицях, які побудовані у відповідності до бізнес-моделі. Таким чином, якщо екземпляр однієї сутності не може дозволити в собі вмістити всі необхідні дані, і для цього необхідно залучати окрему пов’язану з нею таблицю, то це може призвести до загального збільшення складності в роботі з даними. Тож для того, аби підсумувати загальні особливості реляційних та нереляційних баз даних, можна провести аналіз основних відмінностей між ними. Наприклад, у проведеному порівняні для наукового журналу з інженерії визначають 10 головних аспектів [35], що різняться. Якщо їх проаналізувати, то основна відмінність полягає в толеруванні цілісності даних, котре одночасно впливає і на швидкість, і на надійність використання тої чи іншої БД. Таким чином, якщо необхідна надійність збережуваних даних, їхня структурованість у відповідності до бізнес-потреб, відсутність дублікатів та відносно рідке внесення змін, то найкращим вибором буде будь-яка реляційна БД. Якщо ж навпаки існує потреба в роботі з великими обсягами даних, які часто оновлюються, швидкості зчитування в окремих випадках та паралельності здійснення операцій, то обирати слід із нереляційних БД, враховуючи тип даних для вибору вже конкретного підтипу.

2.2.4 Обрання оптимальних DBMS

Зараз існує багато різних БД, які були створенні з думкою про задоволення тих чи інших потреб. Кожне сучасне застосування вміщує в собі багато різних можливостей, кожна з яких полягає в роботі з різними даними. Для того, аби здійснювати їхнє ефективне збереження та обробку, якраз і слід обрати оптимальну БД, аби потім не стикнутися з труднощами в працездатності застосувань. Коли мова йде про веб-застосування, що матиме ознаки соціальної мережі з деякими можливостями майданчику для прослуховування музики, то буде необхідність надавати більш ніж одну функцію, кожна з яких працюватиме з різними даними. Таке застосування обов’язково матиме справу зі збереженням інформації зареєстрованих користувачів, зокрема як із чутливою для автентифікації, так і з публічно доступною в мережі.

Дані, що мають бути засекречені в цілях безпеки ідентичності користувача, мають зберігатися у надійному середовищі. Для цього найкраще підійде будь-яка з реляційних пропозицій, котрі широко підтримуються різними мовами програмування, а також мають великий рівень довіри на ринку систем керування. Їхнє слідування ACID-принципам дозволить бути впевненим у цілісності таких важливої інформації, без яких неможливе користуватися застосуванням зі сторони користувача в цілому. Також слід зазначити, що вони підтримують написання дозволів доступу, котрі покращать безпеку секретних даних на більш низькому рівні RDBMS. Більше того, пари паролів та логінів будуть обов’язковими для заповнення і вони завжди знаходитимуться поруч. Тож це структуровані дані, що зберігатимуться та зчитуватимуться з відповідної таблиці, структура якої не зазнаватиме змін. До того ж, чутлива інформація не належить до тої, що безперервно оновлюється, що також свідчить про доречність обрання реляційної БД. Під час остаточного вибору реляційної БД слід розглянути MariaDB. Вона побудована на базі всім відомої системи керування MySQL, але, якщо порівнювати, отримала зручнішу документацію та стала більш оптимізованою у швидкості

роботи [36]. Більше того, MariaDB чудово піклується про безпеку сховища та доступу до даних завдяки постійним оновленням. Це буде неабияким плюсом під час її використання в ролі сховища чутливої інформації.

Для збереження загальнодоступної інформації користувача, зокрема його псевдоніму, картинки-аватару, біографії, дописів та т.п., слід звернутися до нереляційної БД. Це все здебільшого неструктуровані дані, деякі поля яких можуть бути взагалі відсутні, зокрема той самий аватар, який користувачі інколи не додають в соціальних мережах. До того ж, такі дані не є надто важливими у сенсі їхньої цілісності, якщо порівнювати з паролями, а це свідчить про набагато меншу важливість слідування ACID. Слід також зазначити, що описові дані та картинки користувачі змінюють значно частіше, що також спричинятиме зайве навантаження для пошуку і оновлення записів у реляційних баз даних, сховище яких має бути передусім завжди доступним для швидкого зчитування чутливої інформації. Саме тому для збереження та обробки схожих «нечутливих» даних, що можуть бути численними, можна покласти саме на окрему нереляційну БД, зокрема на документно-орієнтовні MongoDB або DynamoDB, обидві з яких також підтримують здійснення транзакцій [37][38]. Вони дозволять з легкістю опрацьовувати неструктуровану інформацію у відповідних колекціях за тематикою, використовуючи зручні для роботи формати BSON чи JSON відповідно. Вибір на користь першої буде навіть більш доречним, оскільки вона не є пропрієтарною розробкою, може бути розгорнута на будь-який хостинг, а також підтримує більш гнучку та зручну мову запитів [39].

Можливість надання рекомендацій у соціальних мережах вимагає залучення окремих технологій для їхнього ефективного створення. Саме тому використання зв’язків між вершинами у графових БД стало невід’мною частиною для розробки такої функції. Навіть звичні реляційні бази даних, які головним чином побудовані на взаємозв’язках між таблицями-сутностями, не дозволяють ефективно скористатися таким зв’язком. Причина полягає в тому, що вони для самої

реляційної БД не є чимось явним. Адже їй спочатку необхідно «поєднати» відокремлені дані шляхом проведення операції JOIN, а вже потім отримати повноцінний зв'язок, за котрим можна отримати все необхідне. Це може неабияк вплинути на швидкість отримання потрібної інформації, особливо якщо відбуваються постійні дії-оновлення, які впливають на такі зв'язки. У графових БД всі дані навпаки вже знаходяться у такому «готовому» стані, тобто не потрібно робити зайвих операцій, аби з БД можна було отримати інформацію за потрібним взаємозв'язком [40]. Усі дані в графових БД, зокрема в Neo4j, складаються з вузлів-вершин, кожний із яких описує екземпляр певної сутності, та орієнтованих шляхів-ребер із певним описом, які їх і пов'язують. За допомогою цих шляхів можна знаходити найвіддаленіші зв'язки між різними екземплярами. Наприклад, коли мова йде про рекомендацію нових друзів у соціальних мережах, то віддаленість зв'язків може бути найрізноманітнішої глибини. Завдяки цим явним шляхам між вузлами, графові бази даних демонструють неймовірну швидкість знаходження «відносин» між двома різними екземплярами. За проведеним порівнянням ефективності знаходження «спорідненості» друзів у соціальній мережі на реляційній та графовій БД, можна побачити великий відрив у швидкості на користь другої.

Depth	RDBMS execution time(s)	Neo4j execution time(s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

Рисунок 2.3 - Показники швидкості реляційної БД та Neo4j [40]

Таким чином, коли ми намагаємося у соціальній мережі знайти для користувача всіх друзів якогось одного конкретного друга, тобто здійснити пошук на глибині третього рівня, то Neo4j вириває вперед у кілька сотень разів за

швидкістю. Це неймовірний результат у порівнянні, коли мова йде про надання рекомендацій у реальному часі. Як наслідок, вибір на користь саме такої графової БД, яка демонструє кращі результати навіть із-поміж власних аналогів [41], буде виправданим для створення рекомендаційної системи застосування.

2.2.5 Розгляд існуючих рекомендаційних систем

Сьогодні важко уявити успішний сервіс, який би не мав змоги порекомендувати користувачеві певні типи послуг або продуктів. Більше того, користувачі різних майданчиків часто не здогадуються, що вони бачать у своїй стрічці медіа лише те, що платформа вирішила їм показати на базі аналізу їхніх дій та даних. Таким чином, різні Інтернет-магазини, соціальні мережі та сервіси неявно змушують людей повертатися до їхніх застосувань за новою «порцією» підібраної інформації. Рекомендації стали рушійною силою, що приваблює користувачів і змушує їх надалі користуватися сервісами компаній. Наприклад, той самий успіх Spotify, який був згаданий раніше, завдячує саме покращеннями власної рекомендаційної системи.

Існує два головних методи для побудови систем рекомендацій: Collaborative Filtering та Content-Based Filtering. Вони різняться стратегією того, за чим і що буде пропонувати система користувачам. Обидві побудовані на різних алгоритмах та фільтраціях для надання якомога доречніших пропозицій послуг та продуктів.

Collaborative Filtering полягає в аналізі попередніх дій людей на платформах, за ним потім здійснюється підбір відповідних рекомендацій. Ця система також поділяється на дві підсистеми [42], головна відмінність між якими в тому, що одна базується на порівнянні вподобань та дій між користувачами, а інша - між продуктами, що були однаково вподобані користувачами. У першому випадку відбувається збір інформації про дії користувачів для подальшого порівняння з іншими користувачами, коефіцієнт збігу за якими і визначатиме спорідненість між ними. За цією наближеністю історії дій користувачів їм будуть пропонуватися речі,

які, наприклад, вже подобаються одному, але ще не були вподобані іншим. Схожий підхід стосується й підсистеми на продуктах, наприклад, композиціях, якщо користувач високо оцінив якусь пісню, то йому будуть запропоновані інші пісні, які були вподобані людьми, що також високо оцінили згадану композицію [43]. Таким чином, головна різниця полягає в тому, що в першій підсистемі ми відштовхуємося від рівня схожості вподобань між користувачами для створення нових пропозицій, тоді як в другій – від інших вподобань користувачів, що однаково оцінили один і той самий продукт.

Content-Based Filtering на відміну від вище згаданої системи базується лише на особливостях продуктів, які вже були вподобані одним користувачем, для рекомендації схожих за якостями речей [42]. Тож ця система не залучає порівняння даних дій або вподобань з іншими користувачами, користуючись лише відомостями про те, що було вподобано конкретною людиною. Таким чином, якщо користувач, наприклад, вподобав фільм 70-х років минулого століття, де грає Алек Гіннесс, то людині будуть рекомендовані й інші фільми цих років за участі цього актора.

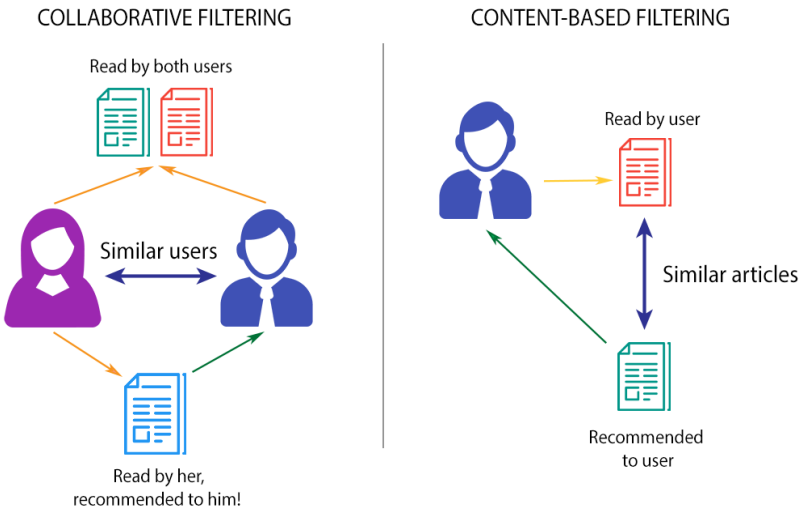

Рисунок 2.4 - Схема двох методів рекомендаційних систем [43]

Тож вибір тої чи іншої системи лежить лише на бажаннях замовника продукту, котрий має визначити те, яким чином буде формуватися рекомендаційна мережа для кожного користувача. Слід вирішити лише те, чи необхідно сервісу

пропонувати нових друзів або «споріднених душ», залучаючи аналіз дій та вподобань інших людей для відповідних порівнянь і знаходжень найкращих пропозицій. У такому разі неможливо уникнути залучення підсистеми Collaborative Filtering. Якщо ж необхідно створювати рекомендації якихось продуктів лише на базі їхніх якостей, шукаючі схожість з тими, які вже вподобав користувач, то необхідно скористатися саме методом Content-Based Filtering.

У випадку застосунку, створення якого є метою даної роботи, слід передусім залучити рекомендаційну стратегію на основі Collaborative Filtering для реалізації пропозицій користувачів за спільними музичними інтересами, знаходячи найбільш схожі профілі для побудови пропозицій.

2.3 Висновки до розділу 2

У цьому розділі надані теоретичні відомості про складові частини веб-застосування, а саме такі: клієнтську та серверну. Здійснено огляд функціональної відповідальності, що покладено на кожну з них в роботі проекту. Також були згадані популярні інструменти та стратегії для розробки як фронтенду, так і бекенду. Був виконаний аналіз різних пропозицій та обґрунтовано вибір кожної з технологій для використання під час реалізації застосування. До того ж, оглянуті існуючі пропозиції серед систем керування БД. Проведено вибір конкретних DBMS для забезпечення роботи з різними даними, котрі будуть присутні в застосуванні. Згадані типи рекомендаційних систем та їхні відмінності в роботі.

Тож маємо обґрунтований набір технологій для розробки всіх складових веб-застосування. Клієнтська частина буде реалізована за допомогою використання бібліотеки React із залученням фреймворку NextJS, який дозволить оптимізувати роботу застосування у сенсі стратегій відмальовування сторінок інтерфейсу. Для реалізації бекенд-функцій буде використано фреймворк Koa на середовищі NodeJS. Цей каркас дозволить реалізувати швидкісне й ненагромаджене API. Обробку даних буде покладено на кілька систем БД: MariaDB, MongoDB та Neo4j. Кожна з

них дозволить належно оптимізувати роботу з різними типами даних, які матимуть певні вимоги до цілісності, а доступ до них повинен мати різний рівень захищеності та швидкості. Рекомендації будуть побудовані на базі Collaborative Filtering для знаходження найбільш «рідних» за інтересами користувачів.

РОЗДІЛ 3. ОПИС РЕАЛІЗАЦІЇ ВЕБ-ЗАСТОСУВАННЯ

3.1 Аналіз технічного завдання

Відповідно до аналізу предметної області та особливостей цільової аудиторії, яка може цікавитися музикою в ролі як слухача, так і виконавця, слід звернути увагу на те, чого їм бракує в сучасних музичних сервісах. Багато хто на ринку музики зрозумів важливість та невід'ємність рекомендаційного аспекту таких платформ, але мало хто надав користувачам те, що зробило б взаємодію між ними більш інтерактивною та явною. Більшість сервісів дозволила користувачам легко знаходити нову музику, але зовсім не зробила легшим знаходження нових людей за спільними інтересами. Тож для того, аби зацікавити користувачів необхідно скористатися досвідом раніше згаданого застосунку Vampr, який завдяки об'єднанню деяких особливостей музичних сервісів і соціальних мереж задовольнив потреби багатьох користувачів, яким було замало можливостей майданчиків Spotify та Soundcloud. Тож слід перелічити аспекти функціоналу веб-застосування, яке б вдало поєднало знаходження нової музики та друзів на базі рекомендацій.

Передусім необхідно реалізувати такі можливості веб-застосування:

a) створення нового профілю за однією з двох ролей: музиканта та слухача;
b) вказання жанрових музичних вподобань для кожного із зареєстрованих користувачів;
c) обрання власних окремих музичних навичок для музикантів на сторінці профілю;
d) створення дружніх зв'язків з іншими користувачами на базі двостороннього підтвердження з можливістю вилучення в майбутньому;
e) формування індивідуальної стрічки дописів від друзів;
f) прив'язування акаунту з сервісу Spotify, якщо в користувача такий є, щоб ділитися з іншими створеними на згаданій платформі плейлістами (від англ. playlist – список відтворення);

g) прослуховування композицій із прикріплених плейлістів Spotify: короткими уривками для тих, хто не має прив’язки до згаданого сервісу, та повними для тих, хто має;
h) прослуховування власних файлів аудіо, які були завантажені користувачами на сторінку профілю;
i) отримування рекомендацій людей за спільними жанровими інтересами, а також спільними друзями та діями на платформі, з окремими рекомендаціям за музичними навичками для музикантів;
j) здійснення пошуку музикантів/слухачів за ім’ям користувача або електронною поштою.

Тож перейдемо до безпосереднього опису реалізації частин веб-застосування, яке б втілило можливості зазначені вище.

3.2 Створення бекенд-частини застосування

Під час створення серверної частини застосування, яке б відповідало за надання API для клієнту, було реалізовано проект із залученням фреймворку Koa. Створений бекенд базується на основних принципах, які були згадані в попередньому розділі стосовно архітектури REST API, яке дозволило реалізувати станово незалежні один від одної частини застосування. Таким чином, у клієнта немає жодної потреби в «розумінні» того, що відбувається на стороні серверу, а серверу відповідно немає необхідності «знати» щось додаткове про кожний підключений клієнт для надання їм повноцінного функціоналу. Така розподіленість дозволить із легкістю оновлювати й змінювати кожну з частин в індивідуальному порядку, адже єдине, чого необхідно дотримуватися, – це непорушність усталеного API, за яким відбувається взаємодія між ними.

Структура бекенд-проекту реалізована на базі рекомендацій архітектурних принципів та у відповідності до практик розподілу відповідальності модулей.

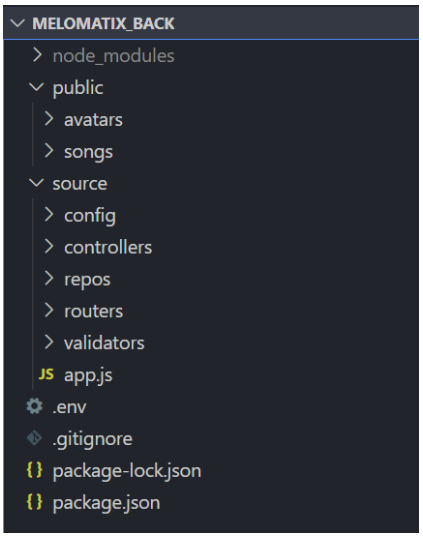

Рисунок 3.1 – Структура бекенд-проекту

Структура головним чином розподілена на допоміжні файли та дві теки, кожна з яких відіграє окрему роль. Папка source розташовує в собі всі важливі компоненти для роботи нашого серверного застосування, зокрема окремі прошарки роботи із отриманими запитами, рахуючи обробники даних, валідатори, контролери та маршрути, ланцюжок яких починається з вхідної точки застосунку Koa в файлі app.js, у якому й реєструються основні шляхи API. Папка public відповідає за надання публічно доступних підтек статичних файлів медіа, яка відкривається за допомогою присвоєння її ролі статичної директорії через Koa. Таким чином, до файлів усередині можливо доступитися за допомогою відповідної URL-стрічки для запиту розташування ресурсу. Також усі необхідні npm-модулі та плагіни для коректної роботи застосування, їхній перелік з указанням відповідних налаштувань та версій для ініціалізації проекту можна переглянути в файлах package.json та package-lock.json. Також врахована важливість безпеки змінних індивідуального середовища розробки, тому записи секретних стрічок та конфігураційних кодів зберігаються в файлі .env [44], котрий прихований за замовчуванням та не відображається в репозиторії проекту.

Завдяки використанню особливостей фреймворку Koa вдалося реалізувати зручну для виводу стану та підтримки систему. Через те, що цей каркас залучає

унікальний для середовища NodeJS підхід до конвеєрної (каскадної) обробки запитів, було можливим реалізувати різні middleware (від англ. – проміжна програма, обробник), які в разі переходу до іншого прошарку обробників не покидаються назавжди. Таким чином, після завершення відпрацювання будь-якого middleware, є можливість повернутися до попереднього для відповідного виводу якоїсь події або перехоплення помилки. Приклад такого проміжного обробника можна переглянути на рисунку нижче.

```
app.use(async (context, next) => {
  try {
    await next();
  } catch (error) {
    context.status = 500;
    context.body = error.message;
  }
});

const AuthRouter = require("./routers/authRouter");
app.use(AuthRouter.routes());
app.use(AuthRouter.allowedMethods());
```

Рисунок 3.2 – Приклад гнучкості проміжного обробника Koa

На прикладі вище зображено обробник у першому виклику метода Koa-серверу app.use(). Він, користуючись блоком перехоплення помилок, обгортає виклик наступного middleware через await next(), який в даному випадку буде пов’язаний з обробкою шляхів запиту авторизації. Таким чином, якщо щось помилкове, що не було перехоплено нижче, трапиться в будь-якому наступному обробнику, ми будемо здатні пропрацювати цю помилку на найвищому рівні.

Збереження файлів медіа, які завантажують користувачі, було реалізовано шляхом використання пакету Multer від Koa, який дозволив зручно валідувати отриманий файл та зберігати його у відповідну теку. Він дозволяє перевірити розширення файлу, відформатовати назву, а також встановити обмеження на обсяг отриманих даних.

```
const storageSongs = KoaMulter.diskStorage({
  destination: function (_, _, callback) {
    callback(null, "./public/songs");
  },
  filename: function (_, file, callback) {
    callback(null, uuidv4() + "." + file.originalname.split(".")[1]);
  },
});
const filtererSong = (_, file, callback) => {
  file.mimetype === "audio/mpeg" ||
  file.mimetype === "audio/mp3" ||
  file.mimetype === "audio/ogg"
    ? callback(null, true)
    : callback(() => {
        throw new Error("Strange extension, is it really an audio file?");
      }, false);
};
const gbQuarterSize = 268435456;
const uploadSong = KoaMulter({
  storage: storageSongs,
  fileFilter: filtererSong,
  limits: { fileSize: gbQuarterSize },
});
```

Рисунок 3.3 – Налаштування Multer для валідованого збереження аудіо

Для демонстрації перебігу взаємодії модулів серверної частини візьмемо приклад реалізації збереження допису користувача із прикріпленим медіа-файлом. Для початку відбувається реєстрація відповідного маршруту API у раніше згаданому файлі app.js. Також необхідно зазначити те, що всі шляхи й методи є прийнятними під час звернення за цим маршрутом. Далі згідно з зареєстрованим маршрутом реалізуємо обробку HTTP-методу POST з одночасним залученням проміжних обробників авторизації та обробки помилок третьої сторони.

app.js

source > app.js > app.use() callback

```
43  const PostRouter = require("./routers/postRouter");
44  app.use(PostRouter.routes());
45  app.use(PostRouter.allowedMethods());
```

Рисунок 3.4 – Реєстрація маршрутів, що стосуються роботи з дописами

```
PostRouter.post(
  "/postWithSong",
  AuthController.authorizeMiddlewareController,
  AuthController.handleThirdPartyError,
  uploadSong.single("song"),
  async (context) => {
    const post = {
      path: context.request.file.path,
      text: context.request.body.text,
      playlistURL: context.request.body.playlistURL,
    };
    await PostValidator.validateAsync(post);
    await PostController.makePost(
      context.post.path,
      context.post.text,
      context.post.playlistURL,
      context
    );
  }
);
```

Рисунок 3.5 – Приклад обробника POST-запиту за маршрутом /postWithSong

У зазначеному вище прикладі також використовується метод збереження файлу аудіо за отриманими даними з тіла запиту. Це здійснюється за допомогою імплементованого раніше методу Multer, який й дозволить здійснити збереження в публічну папку. У тілі обробки запиту POST відбувається створення об’єкту допису за отриманими даними з їхньої подальшою валідацією. Уже після цього перевірені дані передаються до окремого контролера (див. рисунок 3.6), в якому відбуватиметься обробка й надсилання результату клієнтові.

```
makePost: async (songLink, text, playlistURL, context) => {
  const id = context.state.userId;
  const { inMongoCreation, inNeoCreation } =
    await PostRepo.makePost(id, text, songLink, playlistURL);

  if (inMongoCreation.dbSuccess && inNeoCreation.dbSuccess) {
    context.status = 200;
  } else {
    context.status = 503;
    context.body = { msg: "Error during posting request. Sorry." };
  }
```

Рисунок 3.6 – Приклад контролера, що опрацьовує створення допису за даними з POST-запиту

Тут відбувається отримання id користувача за пройденою авторизацією для подальшого збереження даних посту (від англ. post - допис) в базах даних. У разі, якщо створення нового допису було успішним, то ми формуємо відповідь зі

статус-кодом HTTP, що дорівнює 200. В іншому разі відбувається формування тексту помилки з вказанням статус-коду 503, що знаменує несправність сервісу. Останнім, що необхідно показати, є метод роботи з базами даних, що знаходиться у відповідному модулі, що стосується маніпуляцій над дописами. Саме тут відбувається формування підготовлених запитів до БД, з використанням змінних, аби уникнути ін’єкцій та зловживань.

Рисунок 3.7 – Приклад методу, що займається збереженням даних у БД

Подібним до показаного чином відбувається обробка всіх інших звернень від клієнта. Таким чином, реалізована зручна архітектура, яка дозволить прослідковувати за перебігом пропрацювання отриманих запитів сервером. Це дозволить якомога легше відслідковувати джерело помилок та підтримувати вдосконалення бекенд-частини в майбутньому.

3.3 Створення рекомендаційної системи

Для створення рекомендаційної системи на базі музичних інтересів та навичок слід скористатися однією зі стратегій фільтрації, які було зазначено в попередньому розділі. Для реалізації такої системи було вирішено скористатися графовою БД під назвою Neo4j, адже за своїми показниками вона дозволяє швидше за інших знаходити спорідненість та віддалену пов’язаність між екземплярами. Оскільки нам необхідно зробити рекомендаційний «двигун», який дозволить знаходити нових людей за схожими інтересами або зв’зками,

скористаємося саме підсистемою Collaborative Filtering з нахилом на підхід User-to-User. Це дозволить будувати майбутні пропозиції на основі інтересів, що мають найбільш схожі між собою користувачі. Для того, аби така система Collaborative Filtering була якомога точнішою у своїх пропозиціях, необхідно визначати спорідненість між двома користувачами за кількома різними параметрами. У нашому веб-застосуванні маємо кілька таких параметрів, адже передбачена поява кількох корисних зв'язків, зокрема внаслідок вибору нового музичного інтересу, навички та вподобання допису якогось користувача. Унаслідок таких дій у Neo4j від вузла User створюються такі зв'язки:

a) INTERESTED_IN до потрібного вузла Genre.
b) KNOWS до відповідного вузла Skill.
c) LIKED до вподобаного вузла Post.

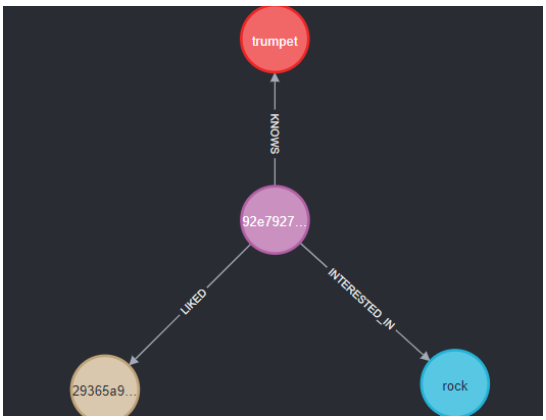

Рисунок 3.8 – Приклад вигляду згаданих зв'язків у Neo4j

Завдяки тому, що в нашому веб-застосуванні також передбачена можливість створення дружніх зв'язків, які мають мати двосторонню орієнтованість між користувачами, можна реалізувати пропозиції нових людей, які відповідно є друзями одного з користувачів. Скориставшись вище згаданими зв'язками, можна створювати змістовні рекомендації. Розглянемо запит до Neo4j, який дозволить нам підбирати перелік людей для пропозиції окремому користувачеві.

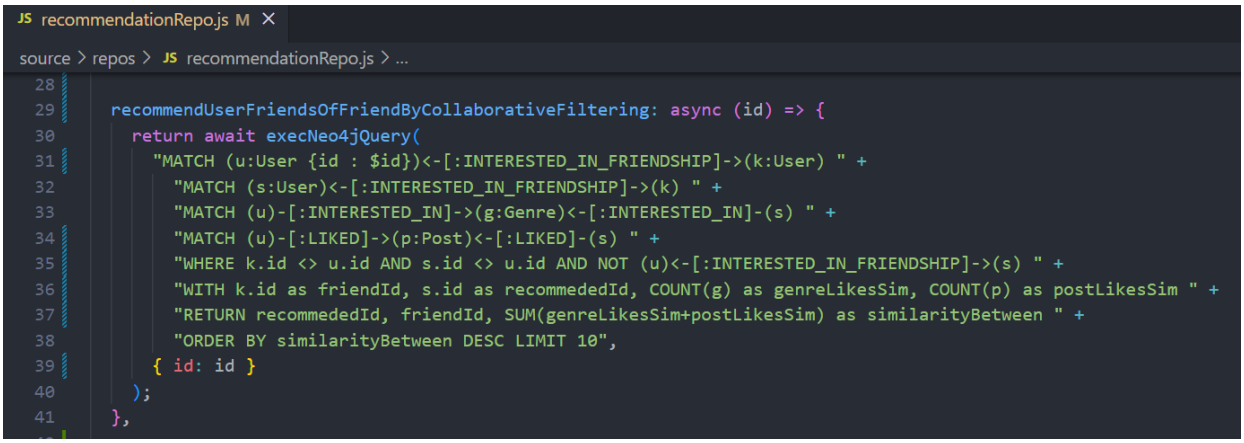

Рисунок 3.9 – Запит до Neo4j, що забезпечує надання рекомендацій за Collaborative Filtering на базі User-to-User

Вважатимемо, що ті люди, які вже перебувають один з одним у дружньому зв’язку, являються спорідненими екземплярами за підходом User-to-User в Collaborative Filtering. Тож на початку відбувається знаходження вузла користувача A, для якого треба зробити підбір рекомендацій, із супутнім знаходженням вузлів усіх тих користувачів, які вже є для нього друзями. Потім відбувається відповідне знаходження незнайомих для A користувачів, які є навпаки знайомим для його друзів, тобто відбувається пошук друзів другого рівня. Коли знайшли всіх необхідних людей, починаємо знаходити всі схожі інтереси між A та ними. Після того, як необхідні перетини схожих зв’язків були зібрані, відбувається їхній відповідний підрахунок. У кінцевому результаті формуємо список тих користувачів, сумарний показник схожості з якими у користувача A є найвищим. Більше того, для того, аби рекомендації були точнішими й доречнішими зробимо сортування за вище згаданим показником схожості, потім встановимо обмеження на вивід у 10 екземплярів. Таким чином, отримали рекомендації ще незнайомих для A людей, які є друзями друзів користувача A, за найбільш схожими музичними інтересами. Аналогічний підхід було реалізовано й до рекомендацій нових користувачів для музикантів за спільними музичними навичками, що були вказані ними в їхніх профілях.

Як наслідок маємо складну мережу взаємозв’язків між людьми в Neo4j (див. додаток А), за допомогою яких користувачі матимуть змогу отримувати різні рекомендації для своїх акаунтів. Більше того, з часом поява ще більшої кількості зв’язків за згаданими параметрами дозволить робити ще точніші підбірки пропозицій, адже саме в цьому й полягає як перевага, так і недолік стратегії Collaborative Filtering [43]. Хоч без даних пропозиції слабкі та не такі змістовні, з поступовим зростанням обсягу відомостей про користувачів вони ставатимуть все кращими.

3.3 Реалізація клієнтської частини веб-застосування

Створення фронтенду відбувалося за допомогою раніше обраних технологій, а саме фреймворку React та допоміжного каркасу NextJS для управління стратегіями рендерингу сайту, які були розглянуті в попередніх розділах.

Під час роботи з NextJS були використані кілька різних стратегій, зокрема CSR та SSR. Цей фреймворк дозволив поєднати дві стратегії в одну у відповідності до того, як було описано в другому розділі. Таким чином, застосунок має як швидке перше відмалювання, так і швидку навігацію по сторінках сайту. До того ж, була використана функція статичного рендерингу сторінок, які не мають зазнавати змін свого вмісту та які мають бути відмальованими в одному й тому ж самому вигляді для всіх користувачів. Такий підхід був застосований до загальнодоступних сторінок веб-застосування, а саме головної, описової та автентифікаційної сторінки. Таким чином, всі вони відмальовуються ще під час побудови проекту [45] й зберігаються на веб-сервері NextJS. Тож тепер без додаткових маніпуляцій зі сторони фреймворку згадані представлення одразу надсилаються користувачам, коли ті до них звертаються в браузері. Отже, завдяки цьому покращуються як показники швидкості отримання сторінок клієнтами, так і розвантажується сам веб-сервер.

Загальна структура фронтенд-проекту побудована на засадах декомпонування сторінок на окремі React-компоненти, які перевикористовуються на різних сторінках застосування. Прикладом такого компоненту може послугувати як картка відомостей про користувача, так і картка представлення допису, адже обидва такі компоненти зустрічаються дуже часто й їхнє перевикористання з різними даними на окремих сторінках буде дуже дорочним. Більше того, завдяки використанню NextJS була можливість легко облаштувати навігацію за визначеними компонентами в папці pages не втручаючись у її перебіг (див. рисунок 3.10). Через це ім'я кожного такого файлу з розміткою сторінки відіграватиме роль прямого продовження URL-шляху, за яким відбуватиметься доступ до представлення окремої сторінки. Таким чином, якщо наприклад матимемо файл під назвою login в папці pages, то зможемо до неї доступитися у браузері за покликанням, що вміщує адресу запущеного веб-серверу фреймворка та відповідну назву файлу. Тож можемо отримати згадану сторінку за запитом на кшталт http://domain-address/login [46]. Більше того, аби додати вкладені шляхи, наприклад, як вказано на рисунку нижче з текою authed, потрібно створити окрему папку з відповідною назвою, яка також стане частиною URL до відповідних сторінок уже в ній.

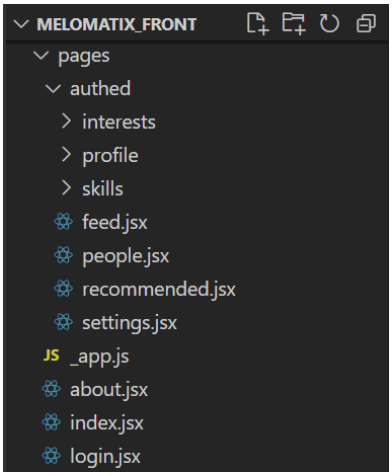

Рисунок 3.10 – Структура вкладень веб-сторінок NextJS

Папка pages вміщує не лише файли, що розташовують в собі розмітку сторінок, а й файл _app.js з кореневим компонентом, в контейнері якого розгортаються всі інші представлення застосування. Явне створення такого компоненту не є важливим, якщо немає необхідності впливати на вкладені представлення з самого кореня, аби зміни застосовувалися й до них. Таким чином, завдяки перевизначенню цього компоненту є можливість обгорнути всі майбутні дочірні компоненти в обробник помилок, певний контекст або візуальну тему, в якій будуть визначені кольори застосунку.

```
<ThemeProvider theme={MeloTheme}>
  <ErrorCatcher>
    <CurrentLayout>
      <Component {...pageProps}></Component>
    </CurrentLayout>
    <Box>
      {spotifyStreaming ? (
        <SpotifyWebPlayer
          play={true}
          initialVolume={10}
          magnifySliderOnHover={true}
          token={cookies.spotifyAccessToken}
          uris={[spotifyStreaming]}
        />) : ( <></> )}
    </Box>
  </ErrorCatcher>
</ThemeProvider>
```

Рисунок 3.11 – Приклад обгортання дочірніх React-компонентів темою та обробником помилок

Тож завдяки особливості React, що полягає в підтримці створення обгорток над компонентами, можна з легкість керувати станом всього фронтенд-застосування.

Слід також згадати, що в React існує можливість створення спеціальних обгорток-функцій, які називаються HOC (Higher Order Component) [47], вони дозволяють ще зручніше здійснювати обробку дочірніх компонентів із впливом на їхнє відображення. Таким чином, вдалося реалізувати зручний перебіг авторизації в застосуванні, успішність якої напряму впливала на те, який компонент буде повернено клієнтові в разі виконання тої чи іншої умови. Таким чином, для того, аби доступитися до сторінок, що розгортатимуться всередині автентифікованого макету, необхідно буде пройти відповідні перевірки у ProtectionHOC. Усередині

цього компоненту вищого поряду відбувається процеси як авторизації з бекенд-частиною на Koa, так і авторизація з сервісом Spotify. Ці процеси реалізовані в хуках (від англ. hook – гачках) під назвою useEffect. Особливість таких функцій в фреймворці React полягає в тому, що вони здатні «чіплятися» за стан компоненту й впливати на нього [48]. Сам useEffect викликається після нового перемалювання компонента, тобто кожний раз, коли він відмальовується на сторінці. Цей хук призначений саме для виконання в ньому окремих операцій на клієнті, зокрема тих, що стосуються запитів до API [49]. Таким чином, можливо реалізувати перевірку й авторизацію користувача кожний раз, коли він переміщується по різних сторінках сайту, що відмальовуються всередині захищеного макету. Тому всі компоненти, які мають бути доступні лише автентифікованим користувачам, будуть у безпеці.

```
const ALayout = ({ children }) => {
  const [isDesktop] = useMediaQuery("(max-width: 768px)");
  return (
    <Box>
      <Head>
        <title>Melomatix</title>
        <meta name="description" content="music" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <link rel="icon" href="/favicon.ico" />
      </Head>
      <ALayoutBox
        id="authed-layout"
        flexDirection={isDesktop ? "column" : "row"}
      >
        <ANavbar bgColor="fourth" />
        <AContent>{children}</AContent>
      </ALayoutBox>
    </Box>
  );
};

export default ProtectionComponent(ALayout);
```

Рисунок 3.12 – Приклад захищеного макету з використанням ProtectionHOC

Для додаткової безпеки збереження авторизаційних ключів в cookie-документах використовується підхід httpOnly. Таким чином, ті токени (від англ. token – жетон-ключ) оновлення авторизації, що відповідають за ініціалізацію або відповідне оновлення основних токенів, які відіграють безпосередню роль в перевірці користувача, є недоступними для маніпуляції зі сторони клієнта.

Користувач не може доступитися до таких токенів, оскільки вони явно не зберігаються в сховищі браузера. Тож немає можливості за допомогою певних скриптів JavaScript дістати ключі, які мають встановлений параметр httpOnly для їхнього поля cookie в заголовках відповіді бекенду.

```
context.set("set-cookie", [
  `accessToken=${accessToken}; Path=/; HttpOnly; SameSite=None; Secure`,
  `refreshToken=${refreshToken}; Path=/; HttpOnly; SameSite=None; Secure`,
]);
```

Рисунок 3.13 – Приклад встановлення httpOnly cookie в заголовок відповіді

Аби запити з клієнта включали ці убезпечені cookie, необхідно вказати відповідний параметр під час створення запиту до API. Наприклад, у разі використання бібліотеки fetch, необхідно всього лише записати параметр credentials в опціях запиту зі значенням include.

```
const response = await fetch(url, {
  method: "GET",
  credentials: "include",
  headers: header,
});
```

Рисунок 3.14 – Приклад включення підтримки httpOnly даних в fetch-методі

Використання такого підходу до збереження чутливих даних є досить важливим моментом в сенсі забезпеченні безпеки як системи, так і користувача. Це один з найголовніших кроків на шляху до захисту від атак Cross-Site Scripting, які мають на меті крадіжку авторизаційних даних користувача для подальшої підміни його сесії, коли зловмисник видає себе в ролі іншого [50].

Отже, за допомогою згаданих технологій вдалося створити клієнтську частину проекту, яка гарно відповідає потребам й очікуванням від сучасного фронтенду. Було реалізовано зручну перевірку безпеки, оптимізувати як навантаження клієнта, так і веб-сервера NextJS, а також створити досить модульний проект, компоненти якого можна було б легко перевикористовувати та оновлювати в майбутньому.

3.4 Огляд реалізованого веб-застосування

Коли вже відомі основні аспекти реалізації частин клієнт-серверного застосування Melomatix й воно здатне надати користувачеві всі раніше перелічені функціональні можливості, слід продемонструвати його роботу. Для цього відтворимо шлях користування запропонованими функціями веб-застосунку. Користувач від початку має відвідати якусь публічну сторінку, а вже потім пройти процес реєстрації з подальшим входженням до власного акаунту. Тож коли неавтентифікований користувач відвідує кореневу сторінку сайту, йому пропонується екран-вітання, як це поширено на відомих веб-сервісах.

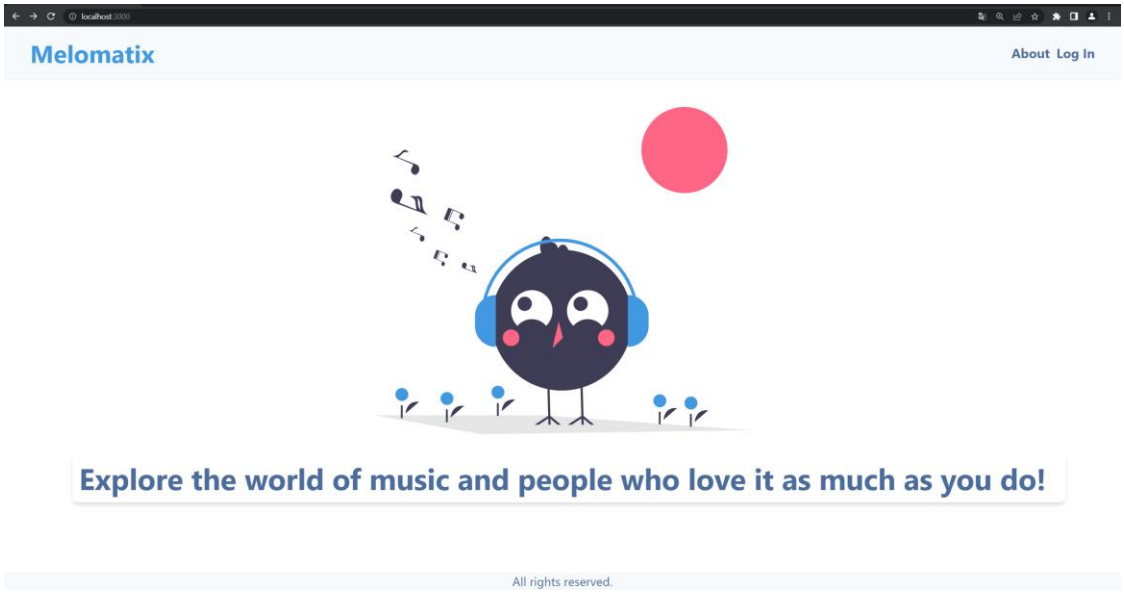

Рисунок 3.15 – Початкова сторінка-привітання веб-застосування

На цій сторінці можна побачити доступну для використання навігаційну панель, яка вміщує в собі покликання на інші публічні сторінки: опису сервісу та автентифікації користувача. У разі відвідування першої, буде запропонована пізнавальна сторінка про сервіс з певними картками-описами. В іншому – сторінка з формою автентифікації/реєстрації користувача.

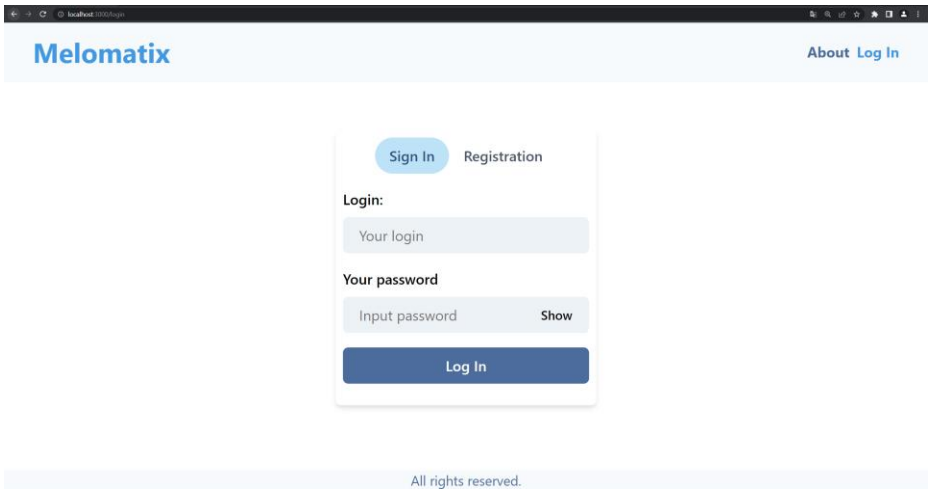

Рисунок 3.16 – Веб-сторінка з формою автентифікації

На цьому екрані у користувача є два шляхи відповідно, у залежності від того, чи він вже має акаунт на сервісі. Якщо немає, то йому необхідно буде пройти процедуру реєстрації, перш ніж отримати доступ до можливостей сервісу. Дані, що вводяться до форми проходять перевірку, особливо якщо мова йде про процес реєстрації нового профілю. Таким чином, будь-який неправильний ввід даних зі сторони клієнту перевіряється як на фронтенді, так і на бекенді. У разі надання якихось невірних даних або їхніх форматів буде відображене спливаюче повідомлення, яке повідомлятиме про неправильно заповнені поля.

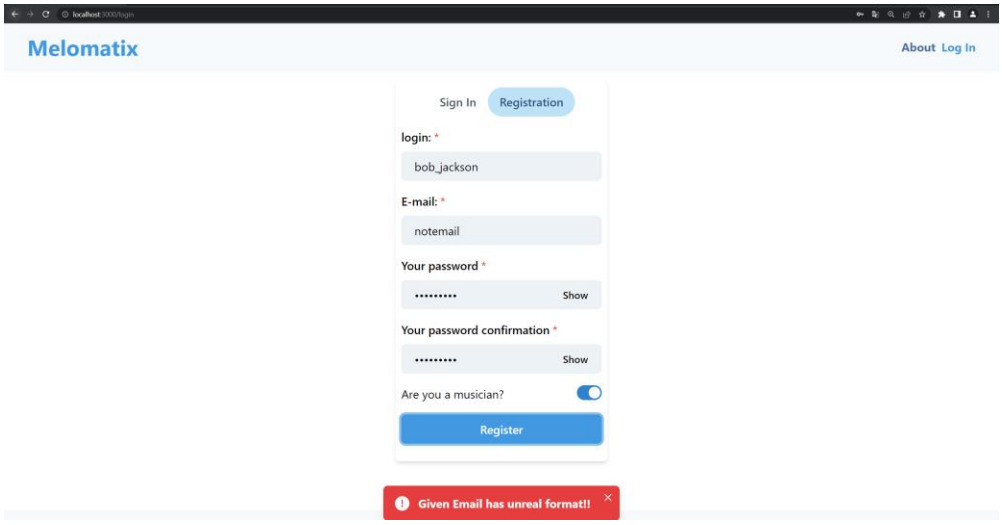

Рисунок 3.17 – Приклад відображення помилки в разі неправильного формату пошти

Також на формі реєстрації користувачу надається вимикач для визначення в межах сервісу його ролі, тобто або музиканта, або звичайного користувача відповідно. Це вплине на те, чи зможе користувач вказувати свої музичні навички та отримувати рекомендації інших музикантів за спільними вміннями. Для демонстрації згаданих функцій пройдемо реєстрацію в ролі митця. Після успішного проходження реєстрації користувачем його одразу буде направлено на сторінку вибору музичних інтересів за наданим переліком жанрів.

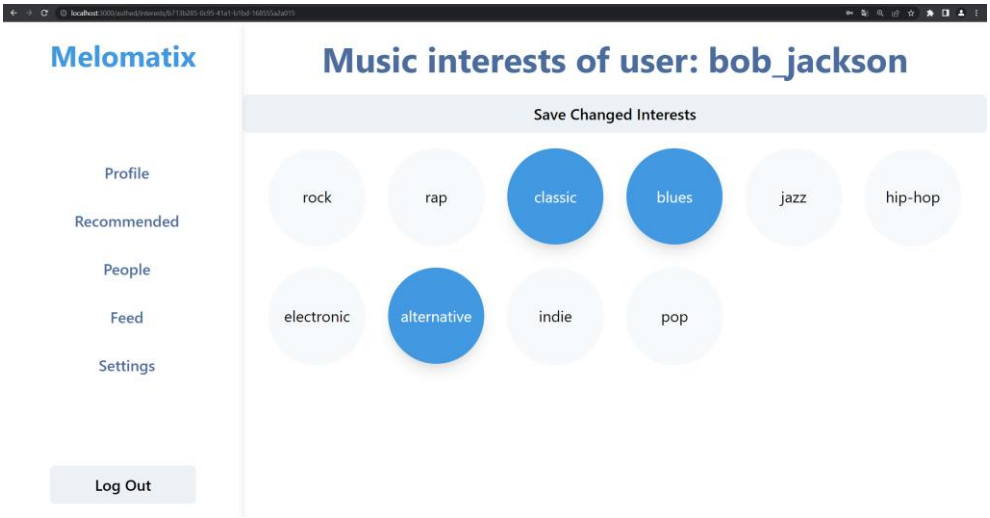

Рисунок 3.18 – Екран вказання музичних інтересів користувача за жанрами

Також можна одразу перейти до розділу профіля за покликанням Music Skills для вказання музичних навичок, якщо користувач створив відповідний профіль музиканта. Також слід зазначити, що потім можна буде вносити зміни до всього раніше обраного, за такої дії будуть переформовуватися рекомендаційні списки.

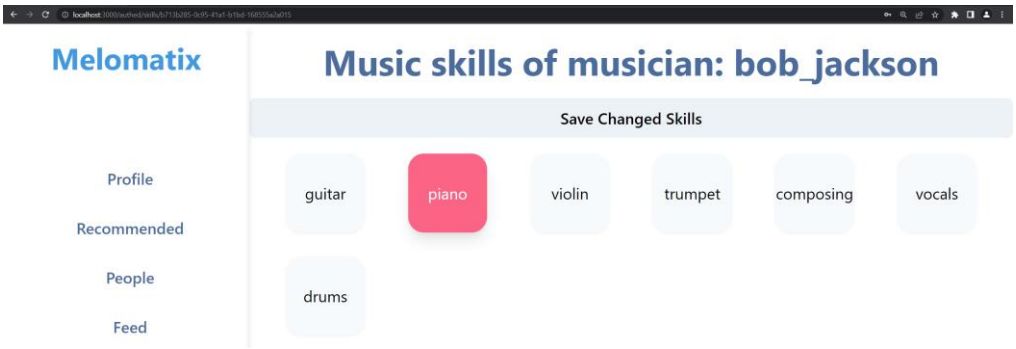

Рисунок 3.19 – Екран вказання музичних навичок музиканта

На екрані музичних жанрів користувач може одразу здійснити вибір улюблених смаків. Хоч це й не є обов’язковим, вибір цих інтересів дозволить сформувати для користувача найперші рекомендації інших людей. Для вибору необхідно натиснути на відповідні бульбашки з їхніми назвами та натиснути на кнопку збереження, що пропонується вище. Оскільки дані про вподобання були внесені, можемо одразу переглянути сторінку рекомендацій за переходом по пункту Recommended або ж почати шукати людей на сторінці People за поштою або ім’ям користувача. На першій же можна буде переглянути новосформовані пропозиції за спільними інтересами. Трошки пізніше, коли будуть вказані музичні навички й зроблені додаткові дії з вподобайками, буде сформовано індивідуальні рекомендації інших музикантів за відповідними критеріями схожості. Розділ By Relation почне працювати тільки після того, як користувач зробить зв’язок з іншою людиною, в якої будуть друзі з найбільш схожими інтересами та вподобаннями. Тож на першому рисунку нижче бачимо наявність трьох різних вкладень рекомендацій, але для інших двох нам буде необхідно принаймні трохи покористуватися веб-застосуванням, аби можна було сформувати списки й для них.

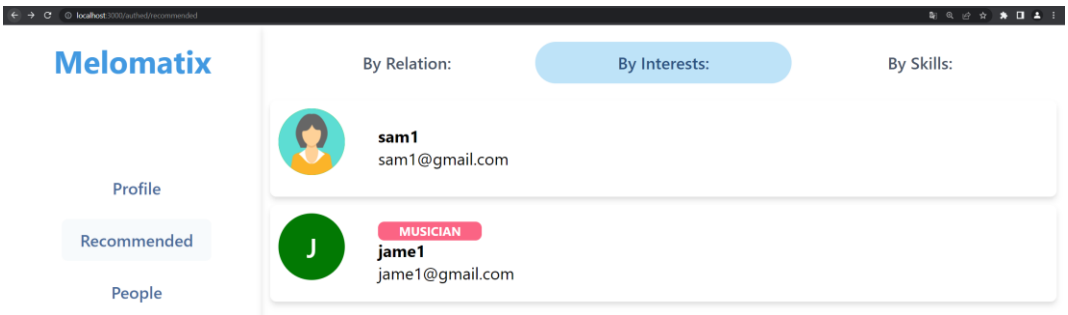

Рисунок 3.20 – Екран рекомендацій із супутніми вкладками пропозицій за різними критеріями

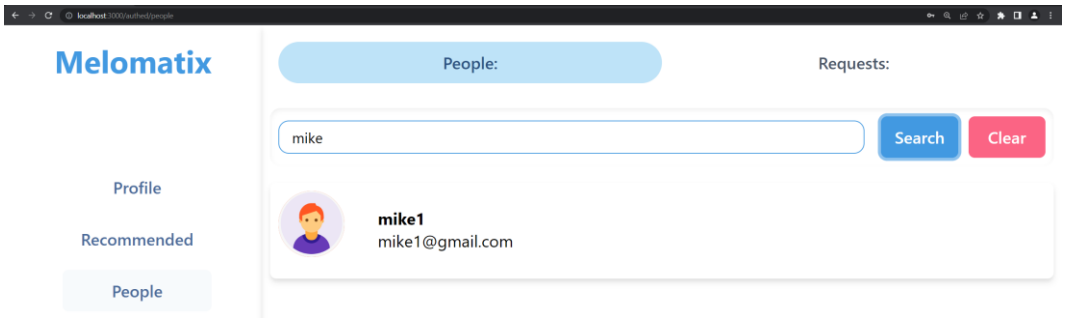

Рисунок 3.21 – Екран пошуку та перегляду користувачів веб-сервісу

На екрані пропозицій за спільними жанровими інтересами ми можемо бачити одразу кілька користувачів, один із яких також є музикантом. Можемо перейти на профіль цього користувача шляхом натискання на його картку.

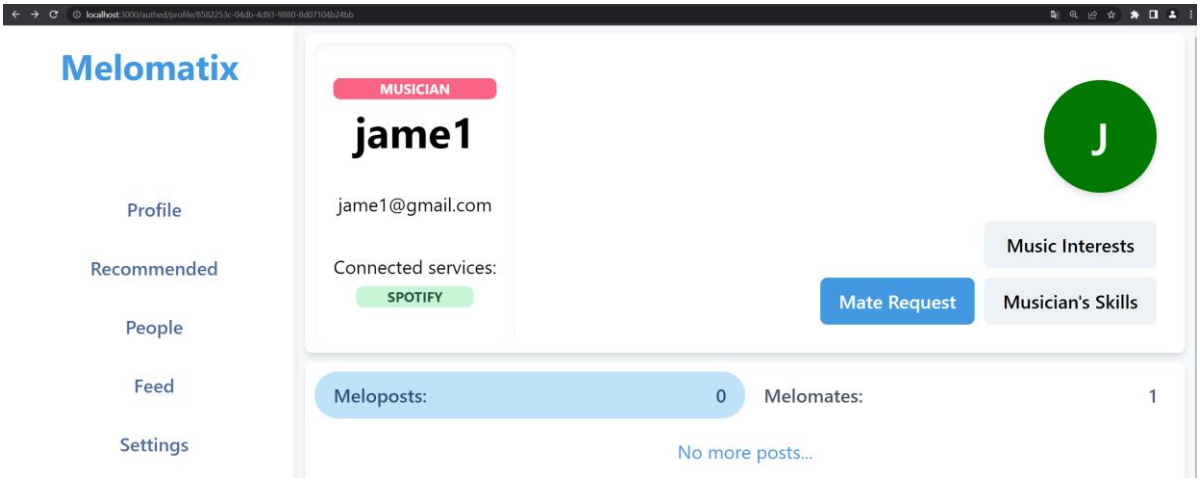

Рисунок 3.22 – Головний екран профілю користувача Melomatix

На сторінці профілю обраного музиканта в нас є можливість переглянути його дописи, якщо він такі робив, а також його існуючих друзів. Також на самій деталізованій картці користувача можна побачити додаткову інформацію про те, чи підключений в нього акаунт до сервісу Spotify. Наявні додаткові кнопки для перегляду й інших даних профілю, зокрема й перегляд обраних жанрових інтересів та музичних навичок. Так само тут відображається яскрава кнопка для запиту на дружній зв’язок з ним. У разі надсилання такого запиту, з’являється напис, що його було надіслано. Він відображатиметься доти, доки відповідний користувач не прийме або не відмовиться від зв’язку.

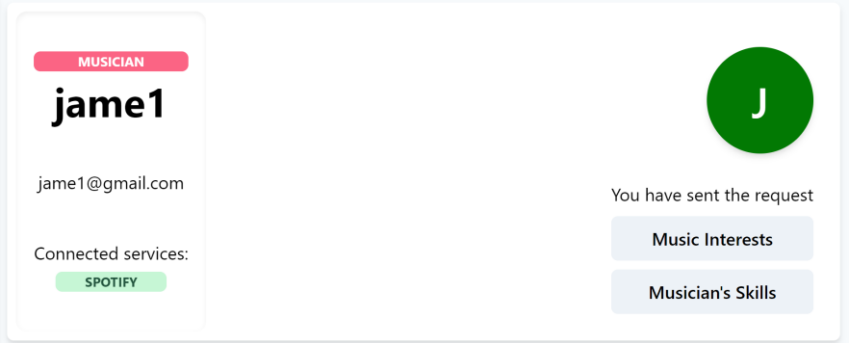

Рисунок 3.23 – Приклад зміни статусу заявки в картці профілю

У свій час користувач, котрому було надіслано запит, отримає відповідне повідомлення-картку на екрані перегляду людей. На вкладці запитів у нього буде можливість переглянути всі нові отримані запити разом з тими, на які він ще не відповів. Тож тут у отримувача буде можливість підтвердити запит або відмовитися від нього відповідного. Від цього залежатиме те, чи відображатимуться користувачі один в одного у відведеному для друзів місці на сторінці профілю. До того ж від такого зв'язку вже можуть почати формуватися рекомендації незнайомих друзів користувача, з яким було тільки-но створено зв'язок.

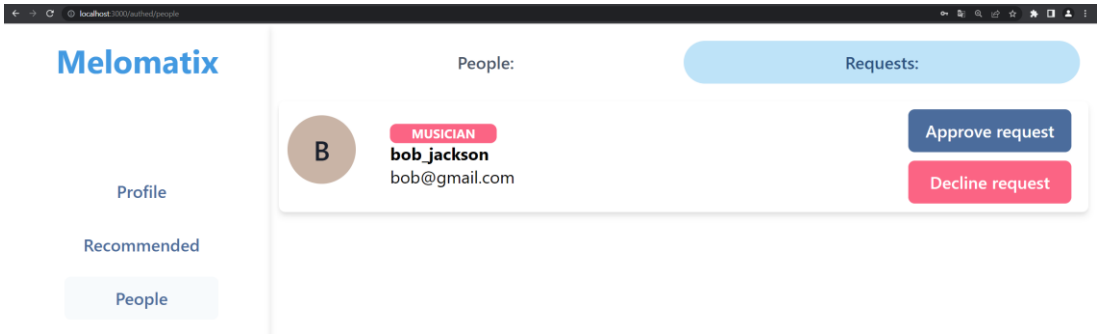

Рисунок 3.24 – Картка з інформацією про отриманий запит в друзі з можливостями відповіді

Як можна було побачити, в деяких користувачів на аватарах (від англ. avatar – іконка або зображення користувача) профілю є картинки, а в інших лише буква. Тож для того, аби завантажити власну картинку слід перейти до розділу налаштувань під назвою Settings у меню ліворуч. На цій сторінці користувачу буде

надана можливість як змінити або завантажити собі аватар, так і прив’язати відповідний акаунт сервісу Spotify, якщо такий є. Це дозволить користувачеві ділитися через дописи своїми плейлістами зі згаданої платформи.

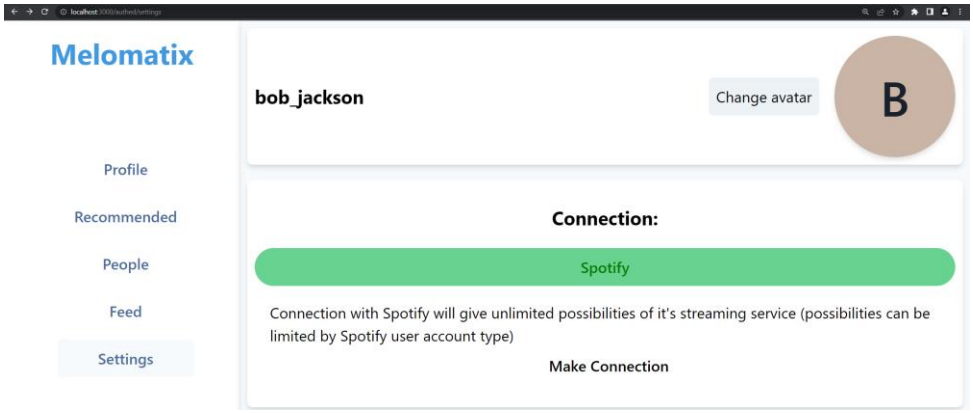

Рисунок 3.25 – Екран налаштувань профілю користувача

Для завантаження аватару необхідно буде обрати відповідний файл з дозволеними розширеннями, які будуть попередньо відображені у вікні-меню системи. У разі натиснення на приєднання до акаунту Spotify необхідно буде пройти автентифікацію в їхньому сервісі. Якщо користувач нещодавно входив до Spotify, то його буде автоматично під’єднано. У разі успішного процесу буде відображено напис «You are already connected» замість «Make connection». Після цього в користувача з’являється можливість прослуховування повних пісень із прикріплених у дописах плейлістах, а також власноручно створювати дописи з ними. Тож вже на екрані власного профілю у користувача буде можливість створити новий пост. Там буде запропоновано прикріпити наявні на прив’язаному акаунті Spotify набори пісень.

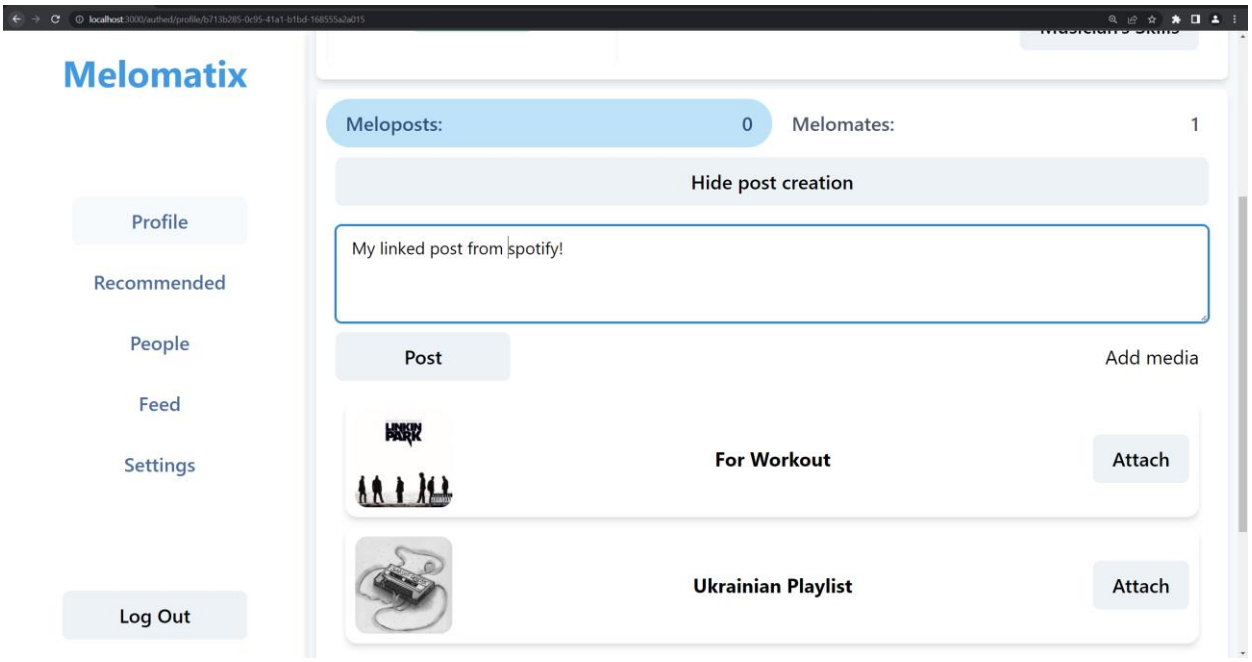

Рисунок 3.26 – Створення нового допису із прикріпленням доступних плейлістів зі Spotify

У разі успішного створення допису, він буде відображений на сторінці профілю користувача у відділі Meloposts. Також він з’являтиметься в особистих розділах Feed існуючих друзів. На відповідному дописі буде відображено його створювача, текст, весь вміст прикріпленого плейлисту, а також кнопку Likes для відповідного надання вподобайок з індикатором кількості вже наявних.
Якщо натиснути на запропоновані в піснях кнопки Play однієї з них, то одразу з’явиться відповідний програвач із можливістю переміщення за часовимм проміжком композиції. Також буде можливість змінювати гучність та зупиняти програвання відповідно.

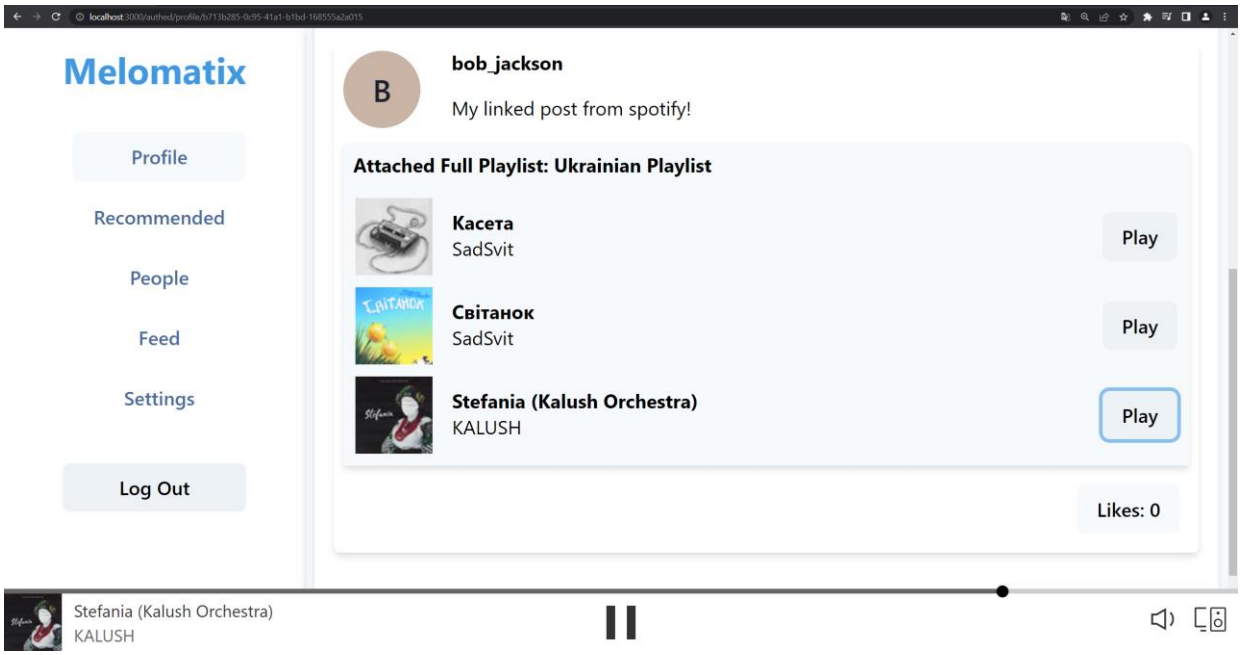

Рисунок 3.27 – Приклад картки допису із прикріпленим плейлістом з відповідним програвачем окремої пісні

Якщо в користувача не буде прив’язаного акаунту Spotify, то йому буде надана можливість прослуховування лише демонстраційних відрізків пісень. Такі уривки можна буде прослухати через спеціальний віджет від сервісу, який одразу вміщуватиме кнопку для реєстрації. До того ж слід зазначити, що в користувача є можливість прикріпляти й власні файли аудіо за відповідними форматами .mp3, .mpeg та .ogg в разі натиснення на напис «Add media» під час створення посту. Таке прикріплене медіа також відображатиметься на картці допису для всіх, хто його переглядатиме. Воно матиме окремий програвач, за допомогою якого можна буде керувати відтворенням аудіо.

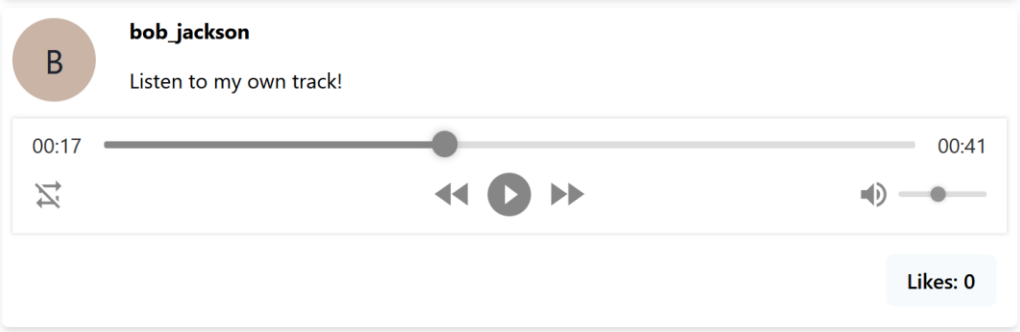

Рисунок 3.28 – Приклад картки посту із власним прикріпленням аудіо файлу

Отже, завдяки відтворенню такого шляху дій користувача був продемонстрований можливий сценарій користування сервісом. Під час проходження цього шляху були показані всі основні функціональні можливості веб-застосування Melomatix.

3.5 Висновок до розділу 3

У даному розділі були розглянуті головні аспекти створення кожної з частин клієнт-серверного застосування. Було визначено технічне завдання проекту з відповідним вказанням необхідних функціональних можливостей, які б зацікавили користувачів з цільової аудиторії. Було проведено опис різних особливостей як бекенду, так і фронтенду із наданням супутніх рисунків. Також було відтворено досвід нового користувача під час користування реалізованим веб-застосуванням. Були продемонстровані основі екрани та представлення веб-сторінок, за якими можна сформувати бачення доступних можливостей та користувацького інтерфейсу.

Висновки

Підбиваючи підсумки виконаної курсової роботи, було отримано повну реалізацію бекенд- та фронтенд-частини, котрі в поєднанні надають очікуваний веб-застосунок із рекомендаційними можливостями. У відповідності до пунктів постановки завдання вдалося планомірно підійти до розробки, врахувавши всі тенденції предметної області та обравши необхідний набір ПЗ. Таким чином, розуміючи потреби цільової аудиторії, стало можливим доречне втілення рекомендацій в межах продукту. Перед етапом опису розробки, були визначені всі необхідні поняття та стратегії використання технологій, які дозволили ефективно підійти до безпосередньої реалізації. У ході створення кожної з частин веб-застосування були враховані документаційні практики інструментів та дотримані головні принципи залучених архітектурних підходів, зокрема REST. У кінці роботи було продемонстровано сценарій користування реалізованим застосунком в ролі нового користувача, також були відтворені основні дії з його сторони та показані відповідні представлення.

Підсумовуючи результат, слід визначити можливі перспективи поліпшень та вдосконалень отриманого веб-застосування. Особливості обраних технологій та предметної області дозволяють додавати безліч додаткових функціональних можливостей. Наприклад, дуже доречним буде додавання підтримки інших музичних сервісів, окрім Spotify, зокрема Soundcloud та Deezer. Це б дозволило розширити зацікавлену аудиторію користувачів. Також можна розглянути впровадження аспекту прогресивного веб-застосунку, тобто PWA (Progressive Web App). Це дозволить користувачам завантажувати самостійну версію застосування на особистий пристрій, що зробить користування веб-сайтом аналогічним до використання додатку. Також можливо й надалі покращувати рекомендації, додаючи нові параметри, які на них впливають. Отже, обсяг можливих покращень та оновлень веб-застосування на базі рекомендацій залежатиме лише від зростаючих потреб користувачів та доступного бюджету.

Список використаних джерел

1. U.S. Bureau of Labor Statistics – Occupational Outlook Handbook : [Електронний ресурс] https://www.bls.gov/ooh/computer-and-information-technology/web-developers.htm (дата звернення: 29.01.2022)
2. Britannica – Technology : [Електронний ресурс] https://www.britannica.com/topic/ARPANET (дата звернення: 29.01.2022)
3. History – History Stories : [Електронний ресурс] https://www.history.com/news/the-worlds-first-web-site (дата звернення: 29.01.2022)
4. BBC – News : [Електронний ресурс] https://www.bbc.com/news/technology-43237139 (дата звернення: 30.01.2022)
5. Spotify – Investors News : [Електронний ресурс] https://investors.spotify.com/financials/press-release-details/2021/Spotify-Technology-S.A.-Announces-Financial-Results-for-Second-Quarter-2021/default.aspx (дата звернення: 30.01.2022)
6. Medium – UX Design – Why hasn't Spotify improved its social features? : [Електронний ресурс] https://uxdesign.cc/why-hasnt-spotify-improved-its-social-features-7fa47e746841 (дата звернення: 30.01.2022)
7. CNN Business – How iTunes changed music, and the world – By Brandon Griggs and Todd Leopold : [Електронний ресурс]

https://edition.cnn.com/2013/04/26/tech/web/itunes-10th-anniversary/index.html
(дата звернення: 10.03.2022)

8. GEMTRACKS – Vampr App Review – By Gemtracks Staff : [Електронний ресурс] https://www.gemtracks.com/guides/view.php?title=vampr-app-review&id=1657 (дата звернення: 10.03.2022)
9. Google Play – Apps – Vampr – Find & Meet Musicians : [Електронний ресурс] https://play.google.com/store/apps/details?id=me.vampr.android&hl=uk&gl=US
(дата звернення: 10.03.2022)
10. Medium – Server Side Rendering (SSR) vs. Client Side Rendering (CSR) vs. Pre-Rendering using Static Site Generators (SSG) and client-side hydration – By Prashant Ram : [Електронний ресурс] https://medium.com/@prashantramnyc/server-side-rendering-ssr-vs-client-side-rendering-csr-vs-pre-rendering-using-static-site-89f2d05182ef
(дата звернення: 15.03.2022)
11. Google – web.dev – Rendering on the Web – By Jason Miller and Addy Osmani : [Електронний ресурс] https://web.dev/rendering-on-the-web/
(дата звернення: 15.03.2022)
12. AltexSoft – What is Front-End Development: Key Technologies and Concepts : [Електронний ресурс] https://www.altexsoft.com/blog/front-end-development-technologies-concepts/

(дата звернення: 19.03.2022)

13. StackOverflow Insights – Developer Survey 2021 – Most dreaded and wanted webframeworks : [Електронний ресурс] https://insights.stackoverflow.com/survey/2021#most-loved-dreaded-and-wanted-webframe-want (дата звернення: 19.03.2022)
14. ClockWise Blog Technology – Front-end JavaScript Frameworks Showdown – By Andrew Hurskiy and Sofiya Merenych : [Електронний ресурс] https://clockwise.software/blog/angular-vs-react-vs-vue/ (дата звернення: 19.03.2022)
15. NgDevelop – Angular History : [Електронний ресурс] https://www.ngdevelop.tech/angular/history/ (дата звернення: 19.03.2022)
16. GitHub Gist – Front-end frameworks popularity (React, Vue, Angular and Svelte) – By Tanguy Krotoff : [Електронний ресурс] https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190 (дата звернення: 19.03.2022)
17. LogRocket – Angular vs. React vs. Vue.js: Comparing performance – By Piero Borrelli : [Електронний ресурс] https://blog.logrocket.com/angular-vs-react-vs-vue-js-comparing-performance/ (дата звернення: 19.03.2022)
18. Javatpoint – ReactJS – React Version : [Електронний ресурс] https://www.javatpoint.com/react-version (дата звернення: 19.03.2022)

19. Stack Overflow Insights – Stack Overflow Trends : [Електронний ресурс] https://insights.stackoverflow.com/trends?tags=reactjs%2Cvue.js%2Cangular%2Csvelte%2Cangularjs (дата звернення: 19.03.2022)
20. GitHub – vuejs/vue – Releases : [Електронний ресурс] https://github.com/vuejs/vue/releases?page=21 (дата звернення: 22.03.2022)
21. Node Package Manager – Vue Package : [Електронний ресурс] https://www.npmjs.com/package/vue (дата звернення: 22.03.2022)
22. DOM benchmark comparison of the frontend JavaScript frameworks React, Angular, Vue, and Svelte – Master's Thesis in Computer Science – By Mattias Levlin : [Електронний ресурс] https://www.doria.fi/handle/10024/177433 (дата звернення: 22.03.2022)
23. TechTerms.com – Software Terms – Backend : [Електронний ресурс] https://techterms.com/definition/backend (дата звернення: 28.03.2022)
24. IBM - Application Programming Interface (API)
    – By IBM Cloud Education : [Електронний ресурс] https://www.ibm.com/cloud/learn/api (дата звернення: 28.03.2022)
25. RapidAPI Blog Dev Room – Types Of APIs – By RapidAPI Staff : [Електронний ресурс] https://rapidapi.com/blog/types-of-apis/

(дата звернення: 28.03.2022)

26. Baeldung – Courses – WebSockets
https://www.baeldung.com/websockets-spring : [Електронний ресурс]
(дата звернення: 29.03.2022)
27. Statistics&Data – Most Popular Backend Frameworks – 2012 – 2021 Data: [Електронний ресурс]
https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2021/
(дата звернення: 29.03.2022)
28. Medium – Web REST API Benchmark on a Real Life Application
    - By Mihai Cracan : [Електронний ресурс]

https://medium.com/@mihaigeorge.c/web-rest-api-benchmark-on-a-real-life-application-ebb743a5d7a3
(дата звернення: 29.03.2022)
29. ExpressJS – Using Middleware – Docs Guide : [Електронний ресурс]
https://expressjs.com/en/guide/using-middleware.html
(дата звернення: 29.03.2022)
30. TopTal – Express, Koa, Meteor, Sails.js: Four Frameworks Of The Apocalypse – By Chuoxian Yang : [Електронний ресурс]
https://www.toptal.com/nodejs/nodejs-frameworks-comparison
(дата звернення: 29.03.2022)
31. Medium – The Best Node.js Framework: Koa VS Express VS Hapi [Detailed Comparison] – By Savvycom JSC : [Електронний ресурс]

https://medium.com/savvycom/the-best-node-js-framework-koa-vs-express-vs-hapi-detailed-comparison-46cc85207d65
(дата звернення: 01.04.2022)

32. KoaJS – Docs : [Електронний ресурс]
https://koajs.com/
(дата звернення: 01.04.2022)

33. Relational Vs. NoSQL databases: A survey – By Mohamed Ahmed Mohamed and Obay. G. Altrafi. C.598 : [Електронний ресурс]
https://www.researchgate.net/profile/Mohamed-Mohamed-314/publication/263272704_Relational_Vs_NoSQL_databases_A_survey/links/00b7d53a495312ad22000000/Relational-Vs-NoSQL-databases-A-survey.pdf
(дата звернення: 05.04.2022)

34. Integrate.io – Which Modern Database Is Right for Your Use Case? – By Donal Tobin : [Електронний ресурс]
https://www.integrate.io/blog/which-database/
(дата звернення: 05.04.2022)

35. A Survey and Comparison of Relational and Non-Relational Database – By Nishtha Jatana, Sahil Puri, Mehak Ahuja, Ishita Kathuria and Dishant Gosain. C.5 : [Електронний ресурс]
file:///C:/Users/ADMIN/Downloads/Telegram%20Desktop/10.1.1.678.9352.pdf
(дата звернення: 10.04.2022)

36. MariaDB – MariaDB vs MySQL Features : [Електронний ресурс]
https://mariadb.com/kb/en/mariadb-vs-mysql-features/
(дата звернення: 10.04.2022)

37. MongoDB – Why Use MongoDB : [Електронний ресурс] https://www.mongodb.com/why-use-mongodb (дата звернення: 12.04.2022)
38. Amazon AWS – About AWS – Anouncing Amazon DynamoDB Support for Transactions : [Електронний ресурс] https://aws.amazon.com/about-aws/whats-new/2018/11/announcing-amazon-dynamodb-support-for-transactions/ (дата звернення: 12.04.2022)
39. MongoDB – Comparing MongoDB and DynamoDB : [Електронний ресурс] https://www.mongodb.com/compare/mongodb-dynamodb (дата звернення: 12.04.2022)
40. Graph Databases – New Opportunities for connected data – By Ian Robinson, Jim Webber and Jim Eifrem. C.11-12 : [Електронний ресурс] https://books.google.com.ua/books?hl=uk&lr=&id=RTvcCQAAQBAJ&oi=fnd&pg=PR2&dq=advantages+of+using+graph+databases+and+recommendations&ots=fLaXDyt-kG&sig=SAm_psRX_QX_9mxVaa7RmrDadyM&redir_esc=y#v=onepage&q=advantages%20of%20using%20graph%20databases%20and%20recommendations&f=false (дата звернення: 12.04.2022)
41. A Performance Comparison between Graph Databases - By Alm Robert and Imeri Lavdim. C.40-41 : [Електронний ресурс] https://www.diva-portal.org/smash/get/diva2:1588722/FULLTEXT01.pdf

(дата звернення: 15.04.2022)

42. Medium – Types of Recommendation Systems & Their Use Cases –
By Maruti Techlabs : [Електронний ресурс]
https://medium.com/mlearning-ai/what-are-the-types-of-recommendation-systems-3487cbafa7c9
(дата звернення: 21.04.2022)
43. Towards Data Science – Brief On Recommender Systems – By Sanket Doshi : [Електронний ресурс]
https://towardsdatascience.com/brief-on-recommender-systems-b86a1068a4dd
(дата звернення: 21.04.2022)
44. Codementor Community – What is .env? How to Set up and rub a .env file in Node? – By Parthibakumar Murugesan : [Електронний ресурс]
https://www.codementor.io/@parthibakumarmurugesan/what-is-env-how-to-set-up-and-run-a-env-file-in-node-1pnyxw9yxj
(дата звернення: 02.05.2022)
45. NextJS Documentation – getStaticProps – By Vercel : [Електронний ресурс]
https://nextjs.org/docs/basic-features/data-fetching/get-static-props
(дата звернення: 10.05.2022)
46. NextJS Documentation – Basic Features: Pages – By Vercel : [Електронний ресурс]
https://nextjs.org/docs/basic-features/pages
(дата звернення: 10.05.2022)

47. React Documentation – Higher Order Components : [Електронний ресурс] https://reactjs.org/docs/higher-order-components.html (дата звернення: 10.05.2022)
48. React Documentation – Hooks : [Електронний ресурс] https://reactjs.org/docs/hooks-overview.html (дата звернення: 15.05.2022)
49. React Documentation – Hooks Effect : [Електронний ресурс] https://reactjs.org/docs/hooks-effect.html (дата звернення: 15.05.2022)
50. NNT – White Hat Security – HttpOnly Session Cookie : [Електронний ресурс] https://www.whitehatsec.com/glossary/content/httponly-session-cookie (дата звернення: 15.05.2022)

Додатки

Додаток А – Мережа взаємозв’язків у Neo4j між існуючими екземплярами