Державний торговельно-економічний університет

Кафедра комп'ютерних наук та інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
на тему:

«Розробка Web-додатку для прослуховування музики»

Студента 1 курсу, 4м групи
спеціальності
122 «Комп'ютерні науки» _____ *підпис студента* Якименко Владислав Андрійович

Науковий керівник
кандидат педагогічних наук, доцент _____ *підпис керівника* Дивак Володимир Валерійович

Гарант освітньої програми
доктор фізико-математичних наук, професор _____ *підпис керівника* Пурський Олег Іванович

Київ 2023

Державний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра комп'ютерних наук та інформаційних систем
Спеціальність 122 «Комп'ютерні науки»
Освітня прорама «Комп'ютерні науки»

Затверджую
Зав. кафедри _____Пурський О.І.
«9» грудня 2022р.

Завдання
на випускну кваліфікаційну роботу студенту

Якименку Владиславу Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної роботи
«Розробка Web-додатку для прослуховування музики»
Затверджена наказом ректора від *«06» грудня 2022 р. № 3284*

2. Строк здачі студентом закінченої роботи *24 листопада 2023 року*

3. Цільова установка та вихідні дані до роботи
Мета роботи: теоретично обґрунтувати, розробити та перевірити Web-додаток для прослуховування музики.
Об'єкт дослідження: процеси розробки web-додатку для прослуховування музики.
Предмет дослідження: методи та технології розробки Web-додаток для прослуховування музики..

4. Перелік графічного матеріалу_____
_____
_____
_____

5. Консультанти по роботі із зазначенням розділів, за якими здійснюється консультування:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Дивак В.В.	09.12.2022 р.	09.12.2022 р.

2	Дивак В.В.	09.12.2022 р.	09.12.2022 р.
3	Дивак В.В.	09.12.2021 р.	09.12.2022 р.

6. Зміст випускного кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ

1.1. Аналіз предметної області

1.2. Аналіз методів та технологій розробки Web-додатку для прослуховування музики

1.3. Аналіз методів та технологій розробки Web-додатку для прослуховування музики

РОЗДІЛ 2. МОДЕЛЬ Web-додатку для прослуховування музики

2.1. Технології Веб-Розробки

2.2. Програмно технічні засоби розробки Web-додатку для прослуховування музики

2.3. Опис програмного продукту

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ Web-додатку для прослуховування музики

3.1. Тестування та аналіз результатів

3.2. Інструкція користувача

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

7. Календарний план виконання роботи

№ Пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	фактично
1	2	3	4
1	*Вибір теми випускної кваліфікаційної роботи*	*01.11.2022*	*01.11.2022*
2	*Розробка та затвердження завдання на випускну кваліфікаційну роботу*	*09.12.2022*	*09.12.2022*
3	*Вступ*	*01.05.2022*	*01.05.2022*
4	*РОЗДІЛ 1. Теоретичні аспекти*	*14.06.2022*	*14.06.2022*
5	*Підготовка статті у збірник наукових статей магістрів*	*20.06.2022*	*20.06.2022*
6	*РОЗДІЛ 2. МОДЕЛЬ Web-додатку для прослуховування музики*	*08.09.2022*	*08.09.2022*
7	*РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ Web-додатку для прослуховування музики*	*20.10.2022*	*20.10.2022*
8	*Висновки*	*02.11.2022*	*02.11.2022*
9	*Здача випускної кваліфікаційної роботи*	*07.11.2022*	*07.11.2022*

	на кафедру науковому керівнику		
10	*Попередній захист випускної кваліфікаційної роботи*	*17.11.2022*	*17.11.2022*
11	*Виправлення зауважень, зовнішнє рецензування випускної кваліфікаційної роботи*	*22.11.2022*	*22.11.2022*
12	*Представлення готової зшитої випускної кваліфікаційної роботи на кафедру*	*24.11.2022*	*24.11.2022*
13	*Публічний захист випускної кваліфікаційної роботи*	*За розкладом роботи ЕК*	

8. Дата видачі завдання «5» грудня 2021 р.

9. Керівник випускного кваліфікаційного проєкту Дивак В.В.
(прізвище, ініціали, підпис)

10. Гарант освітньої програми Пурський О.І.
(прізвище, ініціали, підпис)

11. Завдання прийняв до виконання студент Якименко В.А.
(прізвище, ініціали, підпис)

12. Відгук керівника випускної кваліфікаційної роботи

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

Керівник випускної кваліфікаційної роботи

_____

(підпис, дата)

13. Висновок про випускну кваліфікаційну роботу

Випускна кваліфікаційна робота студента _____

(прізвище, ініціали)

може бути допущена до захисту в екзаменаційній комісії.

Гарант освітньої програми_____ Пурський О.І.

(підпис, прізвище, ініціали)

Завідувач кафедри _____ Пурський О.І.

(підпис, прізвище, ініціали)

«_____»_____2023 р.

Анотація

У випускній кваліфікаційній роботі здійснено розробку функціоналу та дизайну web-додатку для прослуховування музики. В даній роботі були проаналізовані та розглянуті аналогічні системи, інструментальні засоби розробки програмного продукту, матеріали для коректної розробки web-додатку, різноманітні мови програмування, та проведено аналіз предметної області, логічної структури програми. Проведено практичну реалізацію web-додатку для прослуховування музики. Обрані інструментальні засоби розробки програмного продукту дадуть можливість розробити інтуїтивно зрозумілу, зручну web-орієнтовану систему для користувача.

Ключові слова: WEB, HTML, CSS, JavaScript, БД, СУБД, веб-додаток, браузер, архітектура, стримінговий сервіс, програмування, фреймворки.

Anotation

In the final qualification work, the functionality and design of the web application for listening to music was developed. This work analyzed and developed similar systems, tools for developing a software product, materials for the correct development of a web application, various programming languages, and analyzed the subject area, the logical structure of the program. A practical implementation of a web application for listening to music has been carried out. The selected software product development tools provide an opportunity to develop an intuitive, convenient web-oriented system for the user.

Keywords: WEB, HTML, CSS, JavaScript, DB, DBMS, web application, browser, architecture, streaming service, programming, frameworks.

ЗМІСТ

ВСТУП

Музика є невід'ємною складовою нашого життя. Ми чуємо її скрізь, на концертах, по телевізору, на радіо, в автомобілі, при поїздках чи подорожах, навіть в метро не буває і дня щоб не почути якогось вуличного музиканта, ну і мабуть найактуальніше це на наших смартфонах, в навушниках, люди постійно слухають музику. Можна навіть з впевненістю заявити, що музика з нами більшу половину нашого життя, в будь яких її проявах, будь то спів пташок чи ваша звичка наспівувати собі щось [1]. На сьогоднішній день, існує величезна кількість різноманітних жанрів і типів музики, з будь якою стилістикою і тональністю, що дає змогу, нам, споживачам, обирати те, що нам ближче до смаку. На цю тему є величезна кількість статей та довідників, як закордонних так і вітчизняних дослідників, таких як: Jim Samson, Roger B. Dannenberg, Douglass M. Green, Allan Moore, Franco Fabbri, Kelly Schwartz, Fouts Gregory, Bonneville-Roussy, Arielle Rentfrow, Peter J.Xu, Man K., Potter Jeff, Vincenzo Caporaletti, Майбурова К.В, Юрій Юцевич, Е.В. Назайкинский, М. К. Михайлов, С. Павлишин та інших [2-15].

З постійним розвитком інформаційних технологій за останній період, та з функціональним зростанням користувачів мережі Інтернет, музичні додатки, стали помічниками для прослуховування музики в будь-який час і найголовніше в будь-якому місці. Досліджуючи питання різноманіття жанрів і смаків в цілому, враховуючи, що кількість пісень на сьогоднішній день обчислюється мільярдами, саме для того, щоб не заплутуватись у цій різноманітності контенту створюють спеціальні музичні додатки [16-17], які дають нам змогу шукати саме ту музику, яка нам подобається, якісь конкретні пісні чи виконавців, специфічні жанри чи подкасти, можливо навіть прослуховувати радіо, або переглянути теперішні музикальні тренди. В середині цих додатків є можливість розподіляти музику по автору, жанру, особистих смаках, тощо. Тобто робити так звані папки, добірки чи бібліотеки, в яких користувачі можуть шукати, зберігати та прослуховувати музику.

Безпосередню роль у розвитку таких додатків взяв на себе розвиток інформаційних технологій та штучного інтелекту, який допомагає користувачам цих сервісів підшуковувати музику, схожу до тої, яку вони прослуховують, на основі накопичення, обробки та аналізу даних. Сукупність цих факторів заохочують створити найефективніший та найголовніше зручний сервіс, так як, я сам являюсь користувачем подібних, і як наслідок хочу поєднати найкраще, і втілити все в одному, який може стати каталізатором фурору на ринку музикальних додатків. Таким чином, постає необхідність розробки методів і засобів автоматизації, обробки даних та транслювання музикального контенту, з метою створення комфортного і ефективного сервісу по прослуховуванню музики, що і зумовило **актуальність** обраної теми дослідження, враховуючи кількість людей які слухають музику, його мету і завдання.

Мета і завдання дослідження. Метою цього дослідження є розробка веб-додатку для прослуховування музики, який надає зручну та недорогу можливість насолоджуватися музикою через Інтернет. Для досягнення поставленої мети необхідно було вирішити наступні **завдання:**

- Проаналізувати теоретичні та практичні підходи розробки web-додатку для прослуховування музики.
- Побудувати модель web-додатку для прослуховування музики.
- З’ясувати особливості використання програмно-апаратних засобів та розробити web-додаток для прослуховування музики.
- Перевірити технологію та, у разі необхідності, внести зміни у web-додаток для прослуховування музики.

Об’єкт дослідження:процеси розробки web-додатку для прослуховування музики.

Предмет дослідження: методи та технології розробки web-додатку для прослуховування музики.

Методи дослідження: теоретичною основою дослідження є емпіричний науковий метод, а також вивчення наукової літератури, статей,

публікацій та інших джерел, що стосуються теми прослуховування музики та розробки веб-додатків, аналіз конкурентних продуктів з метою з'ясування їхніх переваг, недоліків, унікальних особливостей та можливостей, Використання програмування та розробки для створення функціональності, інтерфейсу користувача, бази даних та інших складових Web-додатку. Інформаційну базу дослідження становлять джерела інформації, які використовуються для проведення дослідження розробки Web-додатку для прослуховування музики. Для практичного вирішення поставлених задач використовувалися такі методи:

- літературний аналіз;
- аналіз конкурентів, що дасть можливість зрозуміти ринкові тенденції, визначити можливі прогалини та розробити унікальну пропозицію.
- програмування та розробка;
- тестування та зворотний зв'язок.

Наукова новизна одержаних результатів полягає в використанні передових технологій, таких як хмарні сервіси, швидкому завантаженні даних та вдосконалені алгоритми для розпізнавання музичних жанрів, рекомендацій музичних треків та побудови персоналізованих плейлистів.

Практичне значення. Отримані результати, можуть бути використані для розширення можливостей взаємодії з музичним контентом. Програмна реалізація надає різноманітні функціональні можливості, що дозволяє користувачам активніше взаємодіяти з музикою, насолоджуватися нею з будь-якого пристрою та мати до неї вільний доступ.

Публікації. Результати дослідження опубліковано у збірнику наукових статей студентів, які здобувають освітній ступінь магістра за спеціальністю «Комп’ютерні науки» ДТЕУ на тему: «Розробка Web-додатку», 2023 р.

Структура та обсяг випускної кваліфікаційної роботи. Випускна кваліфікаційна робота складається із вступу, трьох розділів, висновків, списку використаних джерел із 53 найменувань, додатків і містить 86 сторінки основного тексту і 63 рисунка.

РОЗДІЛ 1.
ТЕОРЕТИЧНІ АСПЕКТИ

1.1. Аналіз предметної області

Перед тим, як розібратись з сервісом для прослуховування музики, розглянемо сам процес розвитку цієї сфери і як вона досягла цього етапу, етапу цифровізації, тож почнемо з музичної індустрії [18] загалом.

Музична індустрія це окрема галузь, можна сказати сектор, діяльність якого полягає у виробленні, запису, маркетингової складової, продажу та розповсюдженні музику. Вона включає різні аспекти, такі як студійна робота, живі виступи, музичне видавництво, музичний маркетинг та менеджмент артистів, а також різні аспекти студійної роботи.

Загальною метою музичної індустрії є виробництво, просування та розповсюдження музики. Вона відіграє важливу роль у суспільстві, дозволяючи музичним артистам виразити свою творчість та надихати людей.

Структура індустрії та її функціонування доволі складний та розгалужений процес, оскільки охоплює велику кількість різноманітних галузей та професій, які тісно з нею пов’язані. Основні з них розглянемо для повноти картини і чіткого уявлення про предметну область.

Розпочати можна з музичних лейблів[19], вони є ключовими гравцями в цій індустрії. Відіграючи роль посередників між музичними виконавцями та аудиторією ці компанії забезпечують фінансову та професійну підтримку для розвитку продукції, просування та продаж музики. Але в свою чергу виконавець і лейбл заключають договір, договори можуть відрізнятись, проте грають важливу роль у взаємодії між цими сторонами, встановлюючи права та обов'язки кожної сторони, умови співпраці та розподіл прибутку.

Іншою ключовою ланкою є самі виконавці – особа, чи група, яка виконує музику перед певною аудиторією. В цьому аспекті коротко розглянемо написання та створення музики, студійну роботу, виступи та

концерти. Створення музики це індивідуальний творчий процес, та як правило все починається з ідеї через яку автор бажає висловити себе у музичному плані. Написання тексту, який виражає емоції, розповідає історію або передає концепцію. Створення мелодії та гармонії, що включає в себе гру на інструментах, або використання цифрових програм та звукових бібліотек. Аранжування та звукове оформлення, процес вирішення які саме інструменти та ефекти будуть використані для передачі бажаного звучання. Запис та обробка, студійний запис, використання технічних засобів для оброблення та запису звуку та його обробки. Наступним етапом є зведення - процес об'єднання всіх записаних треків в один звуковий файл. І кінцевим етапом є випуск та промоція, композиція може бути випущена на різних музичних платформах. Продаж, стримінг та скачування роблять музику доступною для слухачів, в той час як лейбл працює над просуванням музики через різні канали: соціальні мережі, музичні відеокліпи, телерадіо, концерти та інші засоби.

Також варто сказати про існування так званих Рекорд-лейблів. Це лейбли, які зазвичай мають власні студії запису або співпрацюють з професійними студіями. Вони забезпечують виконавцями доступ до обладнання та інженерів, які допомагають в записі і обробці звукового матеріалу, надають фінансову підтримку та відіграють ключову роль у маркетингу музики та виконавця. Виступи та концерти є ключовою частиною музичної індустрії і важливим аспектом кар'єри багатьох виконавців. Це не лише джерело доходу, але й спосіб спілкування з аудиторією та побудови фан-бази. Виступи можуть включати широкий спектр подій, від клубних виступів до великих музичних фестивалів.

1.2. Аналіз методів та технологій розробки Web-додатку для прослуховування музики

У зв’язку з стрімким розвитком технологій, вони почали широко розповсюджуватись у музичній індустрії, та на сьогоднішній день, відіграють важливу роль у її розвитку та трансформації, впливаючи на всі аспекти від створення музики до її розповсюдження та споживання. Починаючи створенням музики, з’явились спеціальні робочі станції[20] (англ. DAW, digital audio workstation) такі як Ableton Live, Logic Pro, FL Studio, або Pro Tools, вони представляють собою програмне забезпечення, спеціально розроблене для запису, редагування та обробки аудіо та музичних файлів. Включають в себе широкий спектр функцій та інструментів, спрямованих на полегшення творчого процесу, таких як віртуальні інструменти, засоби редагування та обробки, бібліотека звуків, мультитрековий запис, різноманітні плагіни та автоматизацію ефектів.

Окрім цього, стрімко розвивається штучний інтелект та машинне навчання, на сьогоднішній день, штучний інтелект почали широко використовувати в будь-якій сфері, тому що зараз він на своєму піку розвитку і впровадження в процеси створення різноманітного контенту, його використовують задля збільшення ефективності та зменшенні витрат на персонал, в основному в технічних галузях, але навіть у творчості він отримав змогу показати себе[21]. Технології штучного інтелекту використовують для генерації музичних ідей, аранжування, виявлення настрою слухачів, та навіть написання тексту. Вже існує безліч композицій повністю створених та написаних штучним інтелектом, це значно прискорює процес створення і звучить досить креативно, адже штучний інтелект самотужки створює унікальні музичні фрагменти, емуляції звучання інструментів та вражає своїм високоякісним звучанням. Також основною перевагою штучного інтелекту є те, що він вивчає структури популярних пісень, завжди в курсі всіх трендів, а також аналізує дані із соціальних мереж та стримінгових сервісів для визначення тенденцій, що дає змогу розуміти попит та вдало маркетингувати свою музику.

Варто звернути увагу на використання VR та AR технологій у контексті музики[22]. Віртуальна та розширена реальність дає змогу слухачам взаємодіяти з музикою в тривимірному просторі, а також грати на музичних інструментах. Артисти можуть проводити віртуальні концерти, потрапити на які, можна за допомогою спеціальних VR-окулярів, експериментувати зі звуками та навіть створювати музику в ігровій формі. Таким самим чином можуть проводитись навіть віртуальні тури, де фанати можуть відвідати різні локації не покидаючи своїх домівок.

Однією з останніх тенденцій є Blockchain та криптовалюта[23], ці технології також почали використовуватись у музичній індустрії, в основному, у контексті авторських прав, дистрибуції музики та взаємодії між артистами та слухачами. Системи блокчейну служать для визначення авторських прав на музичні твори, одна з його технологій, а саме смарт контракти[24], автоматизують розподіл винагороди за використання музики, забезпечуючи справедливі винагороди для виконавців та композиторів. Децентралізовані музичні платформи на основі технології блокчейн надають артистам та слухачам більший контроль над своїм контентом, а також спростять взаємодію х правовласниками та монетизацією. Використання криптовалют для здійснення платежів зробить швидшим та дешевшим здійснення оплати за доступ до музичного контенту. Це може бути особливо корисним для підтримки невеликих виконавців та новаторських проектів. Інтеграція блокчейну та криптовалют ще не цілком відбулась у музичній індустрій, але вже починає адаптуватись, створюючи більш прозору, справедливу та динамічну систему як для авторів так і для слухачів.

Не менш захоплюючим способом взаємодіяти з музикою як артистам так і слухачам є музичні ігри, в яких користувач має змогу зробити це через інтерфейс гри. Одним з найяскравіших прикладів такої гри є GarageBand[25] – це програмне забезпечення для macOS та пристроїв системи iOS для створення музики або подкастів. Додаток має інтуїтивний і легкий у використанні інтерфейс, його візуальна структура робить його доступним для

початківців, що дає змогу людям без професійного музичного досвіду розвивати власні навички, адже GarageBand має вбудовані навчальні матеріали та уроки, спрямовані на навчання основ музичного творення та редагування, але в той же час він має достатньо функцій для задоволення потреб більш досвідчених користувачів. Налічує велику бібліотеку вбудованих інструментів та звуків, включаючи гітари, клавішні, ударні, струнні та багато інших що надає можливість музикантам експериментувати з різними звуками та стилями без необхідності власного обладнання, а також включає в себе різноманітні засоби редагування для аудіо та MIDI, присутня можливість використання власних інструментів чи підключення електронних музичних інструментів, а також ефекти, такі як реверберація, дисторшн та еквалізація. Завдяки таким додаткам звичайні смартфони перетворюються не просто на музичні інструменти а на повноцінні студії, за допомогою яких можна створювати власну музику.

1.3. Особливості розробки Web-додатку для прослуховування музики

Враховуючи швидкий розвиток цифрових технологій та їх інтеграцію у інтернет сервіси, з’явилась потреба створення інноваційних Web-додатків, які можуть не тільки підтримувати сучасні тенденції, але й запроваджуватимуть нові технології та розробки для розвитку індустрії, адже прослуховування музики, на сьогоднішній день, набуває зовсім нового значення. Для розуміння поточного стану ринку, необхідно проаналізувати вже існуючі рішення, що дасть змогу визначити стандарти якості та отримати інформацію щодо потенційних напрямків для інновацій власного продукту. Для уникнення типових помилок у процесі розробки власного додатку, необхідно чітко вивчити наявні платформи та додатки, оцінити їх функціональні можливості, інтерфейс, технологічні особливості та стратегії залучення аудиторії, що допоможе зробити новий продукт конкуренто спроможним на ринку. Основним критерієм буде популярність, адже саме

врахування кількості користувачів та рівень їх задоволення допоможе зрозуміти чи є ця платформа популярною серед аудиторії. Другим показником є оцінка різноманітності функцій, які пропонує додаток – від пошуку та відтворення музики до створення персональних плейлистів та персоналізованих рекомендацій. Зручність інтерфейсу є одним з найважливіших критеріїв, адже це важливо для забезпечення хорошого користувацького досвіду, його інтуїтивність та простота в комплексі з великим функціоналом, дасть змогу користуватись додатком великій кількості людей, незалежно від віку. Розгляд можливостей інтеграцій з іншими сервісами є надважливою складовою для розробників таких додатків, адже крім інтеграції з іншими програмами та сервісами, розробники можуть додавати нові функції або вдосконалювати існуючі завдяки API[26] (Application Programming Interface), не переробляючи весь додаток.

API це набір правил та специфікацій, які дозволяють різним програмним додаткам спілкуватися між собою. В основі ідеї API закладається у тому, щоб надати стандартизований спосіб для обміну даними та взаємодії різних систем, без необхідності розкривати деталі внутрішньої реалізації цих систем. Простіше кажучи, API визначає, яким чином програми можуть взаємодіяти одна з одною, визначаючи запити, які можуть бути зроблені, як вони повинні бути зроблені, формати даних, які використовуються. Тобто одна програма може використовувати функціональність або дані іншої, не залучаючись до розуміння внутрішньої реалізації цієї програми. Якісний API супроводжується детальною документацією, яка описує, як з ним працювати, які функції він надає та які дані він може повертати. У контексті веб-додатків для прослуховування музики, API може використовуватись для отримання доступу до музичних каталогів, управління плейлистами, здійснення платежів, інтеграції з соціальними мережами, і багато іншого. Це робить API невід'ємною частиною сучасних веб-сервісів.

1.3.1 Spotify

Spotify [27] — стримінговий сервіс потокового аудіо, що дозволяє прослуховувати музичні композиції та подкасти. Надає послуги легального онлайнового стримінгу аудіозаписів основних світових і незалежних лейблів, в тому числі BBC, Sony, EMI, Warner Music Group та Universal. Запущений у жовтні 2008 року шведським стартапом «Spotify AB». «Spotify» є першим стримінговим сервісом, який надає можливість слухати музику онлайн, не завантажуючи її на пристрій. Сервіс доступний у 236 країнах світу й перекладений на більше, ніж 36 мов, серед яких є й українська. Після повномасштабного російського вторгнення в Україну компанія припинила надавати свої послуги в Росії і закрила свій офіс у Москві. У 2023 році Spotify має приблизно 551 мільйон місячно активних користувачів по всьому світу, з яких близько 220 мільйонів є платними передплатниками. Загальна кількість користувачів у цьому році на 12,68% вища, або на 62 мільйони більше, у порівнянні з минулим роком що свідчить про успішність і розвиток цієї платформи.

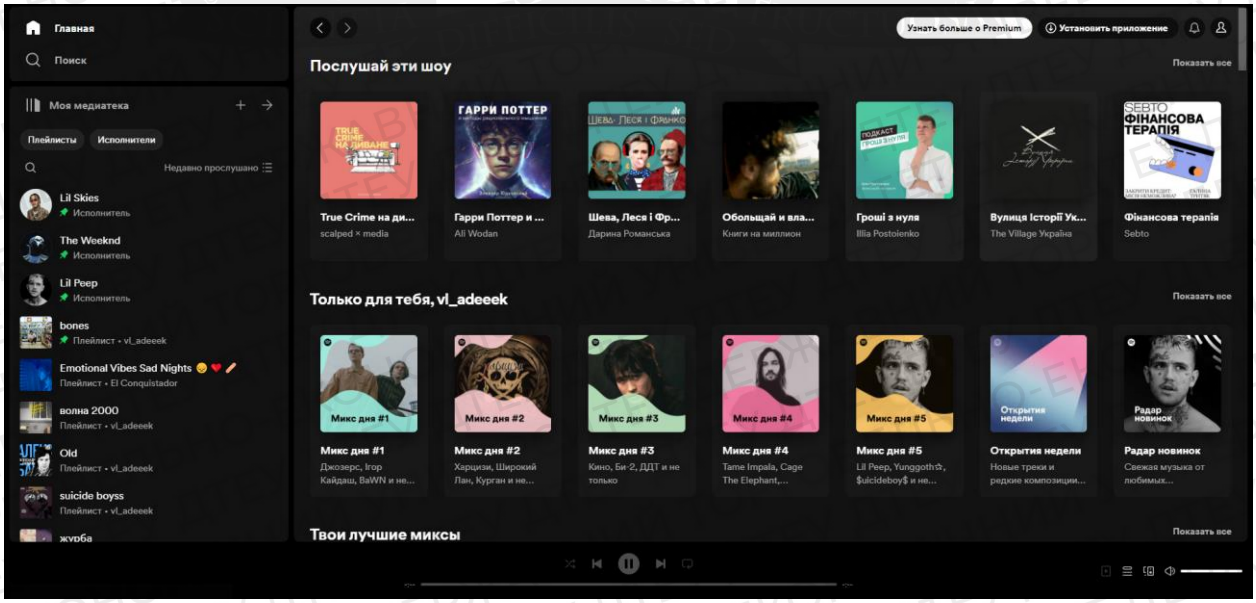

Рисунок 1.1 – Головна сторінка Spotify

Як вже зазначалось вище, майже половина користувачів є платними передплатниками сервісу, у користувачів є два варіанти користування:

безкоштовна підписка та Premium. Преміум-підписка Spotify пропонує ряд переваг порівняно зі стандартною, безкоштовною версією і одна з найбільших переваг Spotify Premium — це відсутність реклами. У безкоштовній версії користувачі, час від часу, слухають рекламні вставки між піснями, тоді як у Premium версії музика звучить без перерв. Spotify Premium пропонує покращену якість звуку. Це особливо важливо для аудіофілів та тих, хто слухає музику на високоякісному обладнанні. Також преміум-користувачі можуть завантажувати пісні, альбоми, плейлисти та подкасти для офлайн прослуховування, що є ідеальним для подорожей або місць з обмеженим доступом до Інтернету. Безкоштовна версія Spotify має певну кількість обмежень в порівнянні з Premium версією і одне з найголовніших це обмеження на кількість пісень, які можна пропустити за годину. Тобто у безкоштовній версії Spotify існують обмеження на кількість треків, які можна пропустити за певний період часу, користувачі безкоштовної версії мають право пропустити не більше шести пісень на годину. Це обмеження введено Spotify з метою заохочення користувачів перейти на платну підписку Premium, яка знімає це обмеження, дозволяючи користувачам вільно вибирати та пропускати пісні без будь-яких обмежень. У безкоштовній версії, після вичерпання ліміту пропусків, користувачам доведеться слухати музику, яка відтворюється автоматично, до того часу, коли обмеження буде скинуто. Spotify Premium дозволяє вільно вибирати та відтворювати будь-яку пісню з каталогу, не залежно від того, на якому пристрої ви її слухаєте та пропонує різні тарифні плани для Premium підписки, включаючи індивідуальні, сімейні, студентські та інші плани, які пропонують гнучкість та додаткові переваги. Враховуючи це можна зробити висновок, що для повноцінного користування необхідно купувати платний тарифний план, адже саме Spotify Premium є ідеальним варіантом для тих, хто шукає якісний музичний досвід без перерв на рекламу, з можливістю вільного вибору та прослуховування музики в будь-який час і будь-де.

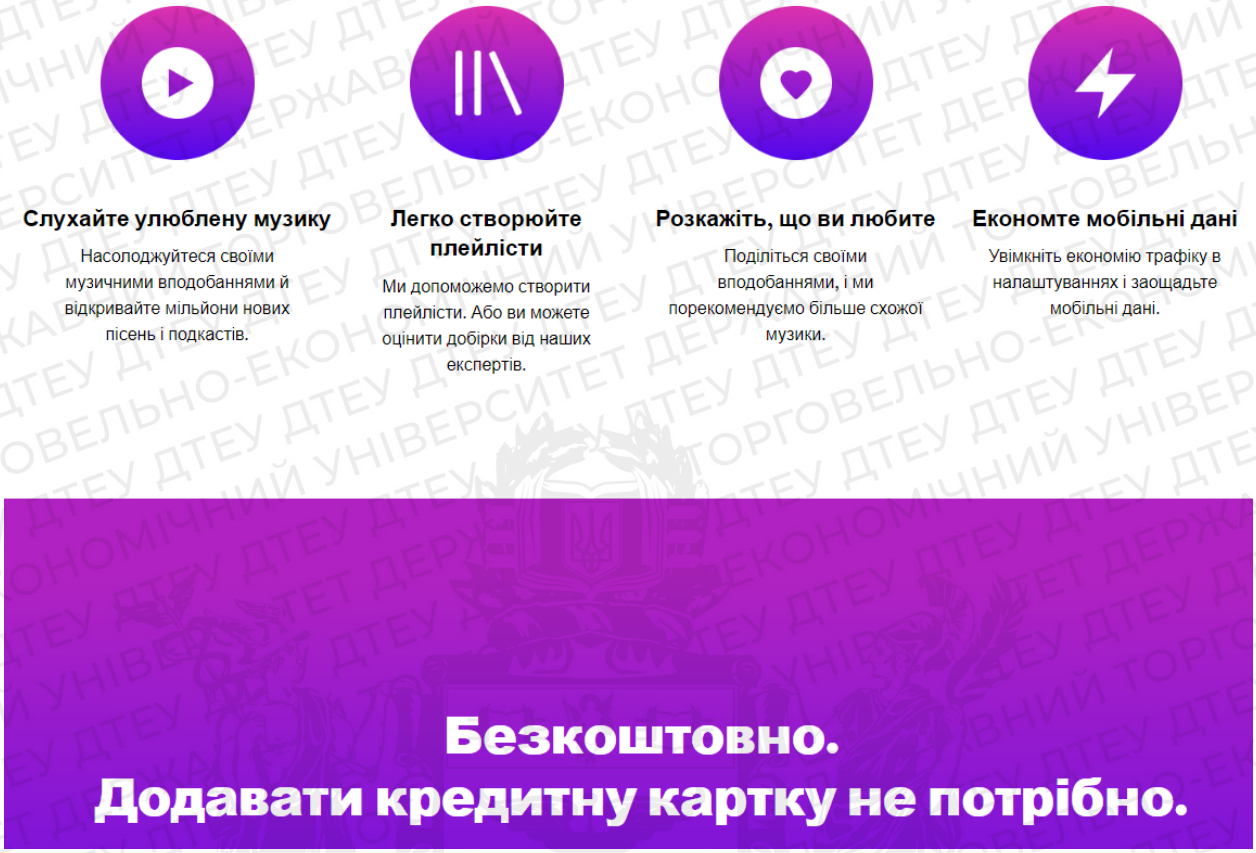

Рисунок 1.2 – Переваги Spotify

Spotify, як одна з найпопулярніших музичних стрімінгових платформ, має свої переваги та недоліки, а також відмінні характеристики, які виділяють її серед інших платформ, серед переваг можна виділити:

- Величезний Каталог Музики

Каталог музики Spotify є одним із найбільших серед музичних стрімінгових сервісів, він надає доступ до мільйонів пісень і альбомів, включаючи широкий спектр жанрів та артистів та охоплює музику з усього світу. Від популярних хітів до нішевих жанрів, від класичної музики до сучасного хіп-хопу, платформа пропонує широкий вибір для всіх смаків. Варто виділити, що сервіс регулярно оновлює свій каталог, включаючи нові релізи від відомих артистів, а також пропонуючи ексклюзивний контент, який недоступний на інших платформах.

- Персоналізовані Рекомендації

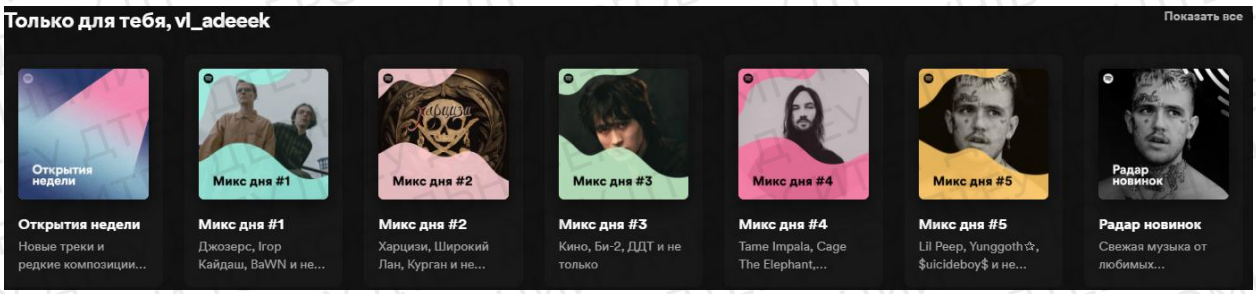

Рисунок 1.3 – Приклад персональних підбірок

Spotify відомий своїми сильними алгоритмами рекомендацій, які допомагають користувачам відкривати нову музику відповідно до їхніх вподобань. Це є однією з ключових особливостей, яка виділяє цей сервіс на тлі інших музичних стрімінгових платформ та робить кожен музичний досвід унікальним. Spotify використовує складні алгоритми машинного навчання, які аналізують музичні вподобання та поведінку користувачів. Це включає в себе треки, які вони слухають, пропускають, зберігають у своїх бібліотеках або додають у плейлисти. На основі цих даних Spotify створює унікальний профіль слухання для кожного користувача, ідентифікуючи їхні улюблені жанри, артистів, пісні та інші музичні уподобання. А вже на основі цього профілю Spotify генерує персоналізовані плейлисти, такі як "Discover Weekly" і "Daily Mix", які представляють підбірку пісень, вибраних спеціально для кожного користувача. "Discover Weekly" відомий тим, що кожен тиждень пропонує нові пісні, які, як передбачається, сподобаються користувачу. Ключовим є те, що ці рекомендації постійно оновлюються, враховуючи зміни в музичних вподобаннях користувачів та нові тренди у музиці. Ці рекомендації сприяють залученню користувачів, пропонуючи їм відкривати нову музику та розширювати свої музичні горизонти, що в свою чергу збільшує час, проведений ними на платформі. Персоналізовані рекомендації Spotify зробили його не просто сервісом для прослуховування музики, а потужним інструментом для відкриття нових музичних вражень, що відповідають унікальним смакам кожного користувача.

- Користувацький Інтерфейс

Користувацький інтерфейс Spotify відрізняється своєю інтуїтивністю та зручністю використання, що робить його одним із найбільш зручних музичних стрімінгових сервісів на ринку. Інтерфейс Spotify відзначається чітким дизайном з легкодоступними елементами управління. Це дозволяє користувачам легко навігувати по сервісу, шукати музику, переглядати плейлисти та керувати своєю бібліотекою.

Рисунок 1.4 –Spotify User Interface

Контент у Spotify добре структурований. Пісні, альбоми, виконавці, плейлисти та подкасти розділені на різні розділи, роблячи пошук і вибір музики простим і швидким. Головна сторінка показує персоналізовані рекомендації, засновані на минулих прослуховуваннях користувача, що дозволяє легко відкривати нову музику. Система пошуку ефективна та гнучка, вона дозволяє шукати треки за назвами, виконавцями, альбомами або навіть жанрами. Мобільний додаток також має привабливий та функціональний інтерфейс, забезпечуючи зручний доступ до музики на ходу. Загалом, користувацький інтерфейс Spotify орієнтований на забезпечення плавного та приємного досвіду користування, який сприяє зручному доступу до величезного каталогу музики та подкастів.

- Підтримка Різних Пристроїв

Сервіс відзначається своєю широкою підтримкою різних пристроїв, забезпечуючи користувачам можливість слухати музику майже де завгодно. Доступний на iOS та Android пристроях, що дозволяє користувачам слухати музику на смартфонах та планшетах. Мобільний додаток пропонує повний

спектр функцій, включаючи збереження музики для офлайн прослуховування, а також є можливість завантажити та встановити додаток Spotify на Windows, macOS та Linux системи, отримуючи доступ до більшого екрану та можливостей настільних комп'ютерів. Підтримує домашні аудіосистеми та смарт спікери, розумні годинники та фітнес браслети, телевізори та ігрові консолі, а також підтримує інтеграцію з автомобільними системами. Завдяки цій багатоплатформеній підтримці, Spotify забезпечує легкий доступ до музики незалежно від того, де знаходиться користувач чи який пристрій він використовує.

- Соціальні Функції

Серед переваг, які виділяють цей сервіс серед інших також можна виділити низку соціальних функцій. Користувачі можуть створювати спільні плейлисти, до яких інші користувачі можуть додавати пісні. Це ідеально підходить для організації заходів, робочих колективів або просто для спільного вибору музики з друзями. Треками, альбомами, плейлистами та подкастами на платформі можна легко ділитися в соціальних мережах, або через прямі повідомлення, а також відслідковувати музичну активність ваших друзів у реальному часі, яких можна знаходити у Facebook[28], адже платформа інтегрована та дозволяє переглядати їх плейлисти та вподобання. Користувачі можуть підписуватися на профілі улюблених артистів та плейлисти інших користувачів, щоб бути в курсі нових релізів та оновлень. Всі ці функції не лише сприяють залученню та утриманню користувачів на платформі, але й створюють відчуття спільноти, об'єднуючи людей через спільні музичні вподобання.

Звісно попри ці переваги й інновації, існують і недоліки, деякі з них були описані, але для структуризації виділимо їх ще раз, серед основних недоліків:

- Реклама у Безкоштовній Версії

Користувачі безкоштовної версії Spotify стикаються з аудіорекламою, яка відтворюється між треками, та візуальною рекламою, яка відображається в

додатку. Реклама з'являється регулярно, але не після кожної пісні, деякі рекламні блоки неможливо пропустити або вибирати типи реклами, який користувач хоче бачити або чути. Така реклама насильно стимулює переходити на платну версію, яка пропонує безрекламний досвід прослуховування.

- Висока Вартість Преміум Підписки

Ціна підписки Spotify Premium може варіюватися в різних країнах, залежно від місцевих економічних умов, в Україні стандартний преміум план коштує $4.99 на місяць після безкоштовного місяця випробувального періоду, звісно Spotify пропонує декілька видів підписок, включаючи індивідуальні, сімейні, студентські та дуетні плани. Кожен план має свою вартість, при цьому сімейні та студентські плани часто пропонують знижки. Для студентів, які навчаються у визнаних вищих навчальних закладах, підписка коштує $2.49 на місяць, план для двох осіб, які проживають за однією адресою, коштує $6.49 на місяць, а план для до шести осіб, які проживають разом, коштує $7.99 на місяць, але для деяких користувачів вартість Spotify Premium може здатися високою у порівнянні з іншими стрімінговими сервісами, особливо якщо вони пропонують схожі послуги за нижчою ціною.

- Обмежена Доступність Деяких Пісень

Це в першу чергу пов’язано з доступністю музики в різних регіонах та для різних користувачів. Найбільша причина обмеження пов'язана з авторськими правами та угодами про ліцензування. Spotify має угоди з лейблами та правовласниками, які дозволяють стримінг музики. Однак, ці угоди можуть варіюватися в різних країнах, в залежності від місцевих законів про авторські права. Через ліцензійні обмеження деяка музика може бути доступна лише в певних країнах або регіонах. Це означає, що деякі треки або альбоми, які доступні користувачам у одній країні, можуть бути недоступні в іншій. Музика певних артистів або жанрів може бути більш популярною в певних регіонах, що також впливає на доступність і вибір пісень у каталозі Spotify для цих регіонів. Обмежена доступність деяких пісень є загальною

проблемою для всіх музичних стрімінгових платформ і для вирішення цієї проблеми необхідно удосконалювати угоди з правовласниками.

- Відсутність Високоякісного Звуку

У стандартній версії Spotify, якість звуку зазвичай є достатньою для більшості користувачів. Безкоштовна версія пропонує бітрейт[29] до 160 kbps у форматі Ogg Vorbis, тоді як Premium версія підвищує цей бітрейт до 320 kbps. Деякі інші музичні стрімінгові сервіси, такі як Tidal або Amazon Music HD, пропонують звук високої роздільної здатності (Hi-Res Audio), який може перевищувати якість, яку пропонує Spotify. Це може бути важливим фактором для користувачів, які мають високоякісне аудіообладнання, адже якісь звуку залежить саме від обладнання на якому відтворюється музика.

1.3.2 Apple Music

Apple Music[30] — це музичний стрімінговий сервіс, розроблений та запущений компанією Apple Inc у 2015 році. Є одним з провідних музичних стрімінгових сервісів, по популярності займає друге місце після Spotify. Станом на 2023 рік, Apple Music має понад 101 мільйон користувачів по всьому світу, що відображає зростання на 14.7% або 13 мільйонів користувачів порівняно з попереднім роком. Цей стрімінговий сервіс є найпопулярнішим у Сполучених Штатах, де налічується 33 мільйони передплатників, перевищуючи кількість користувачів Spotify та Amazon Music у цьому регіоні. Важливо зазначити, що на відміну від Spotify, який пропонує як безкоштовні, так і платні підписки, Apple Music пропонує тільки платні передплати. Це підкреслює його модель бізнесу, яка зосереджена на наданні якісного контенту та послуг за передплату. З моменту свого запуску Apple Music стабільно збільшує кількість своїх передплатників, що свідчить про зростаючу популярність сервісу серед шанувальників музики.

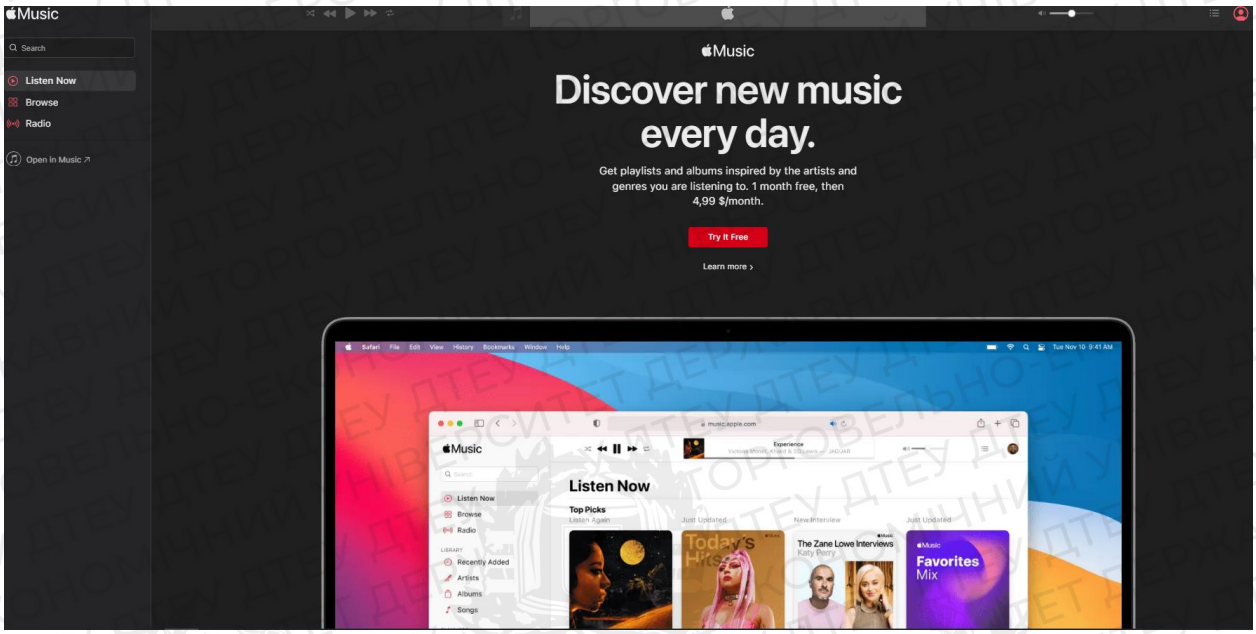

Рисунок 1.5 – Головна сторінка Apple Music на платформі Windows

Цей сервіс швидко здобув популярність, завдяки своїй сильній брендовій ідентичності, так як, є частиною екосистеми Apple, Apple Music відразу ж залучив велику аудиторію завдяки високій відомості та довірі до бренду Apple. До того ж Apple Music було інтегровано з вже існуючою базою користувачів iTunes[31] , що дало сервісу значну кількість потенційних передплатників. Тісна інтеграція з iOS[32], особливо з iPhone та iPad, також сприяла швидкому розповсюдженню сервісу. Пропозиція ексклюзивного контенту та ранніх релізів від популярних артистів стала однією з ключових стратегій Apple для приваблення та утримання користувачів. Apple Music часто укладає ексклюзивні угоди з артистами для релізу їх нових альбомів або синглів. Це означає, що нова музика від цих артистів з'являється спочатку на Apple Music, перш ніж стати доступною на інших платформах. рім ексклюзивних угод, Apple Music іноді пропонує ранній доступ до нових альбомів та синглів, тобто користувачі отримують можливість слухати нові треки або альбоми до їх офіційного виходу. На початку свого існування, сервіс надавав безкоштовний пробного період, що допомогло новим користувачам оцінити сервіс перед тим, як платити за підписку. Протягом цього періоду, користувачі отримують повний доступ до всіх функцій Apple

Music, включаючи весь каталог музики, персоналізовані плейлисти та радіо, проте надалі, якщо користувач не відмінить підписку до кінця пробного періоду, вона автоматично перетворюється на платну підписку, і почне знімати кошти самостійно, вартість підписки на Apple Music в Україні за Індивідуальна Підписка: $4.99 на місяць та сімейний $7.99 на місяць. Звичайно користувачі можуть відмовитися від продовження підписки в будь-який час протягом пробного періоду без будь-яких зобов'язань або фінансових витрат, лише тоді автоматичне продовження перестає існувати. Така практика є ефективним інструментом для залучення нових користувачів, оскільки він дозволяє їм без ризику спробувати сервіс та переконатися в його цінності. Це вигідно як для користувачів, так і для самого сервісу, оскільки він стимулює інтерес до продукту та може призвести до збільшення кількості платних передплатників.

Всі чудово знають як працює Apple і які ефективні маркетингові кампанії ця корпорація проводить, маючи безліч серйозних партнерств з артистами та брендами, вона сприяла обізнаності та популярності свого сервісу. Це не тільки телевізійні реклами, партнерства з артистами та співпраця з лейблами, а й також інтеграція зі спортивними та культурними подіями. Apple Music часто виступає як спонсор або партнер музичних фестивалів, спортивних подій та інших культурних заходів, що забезпечує велику видимість бренду. А також партнерства з технологічними компаніями, корпорація активно пропагує свою продукцію та сервіси задля досягнення так званої екосистеми, тобто один продукт продає інший. Сервіс Apple Music, звісно не міг обійти стороною й студентів, тож регулярно пропонує спеціальні пропозиції та знижки, а також інтегровані підписки з іншими сервісами Apple, щоб залучити нових користувачів.

Первинно Apple Music було інтегровано з екосистемою продуктів Apple, включаючи iPhone, iPad, iPod touch, Mac і Apple TV, використовуючи iTunes як основний інтерфейс для десктопних користувачів. Згодом Apple Music розширив свою доступність і на сьогоднішній день кількість пристроїв

збільшилась в рази, адже крім того, що додались продукти компанії, такі як Apple Watch та HomePod, Apple випустила додаток Apple Music для Android пристроїв у листопаді 2015 року, що дало можливість користувачам Android отримати доступ до сервісу. Щодо користувачів PC на Windows, то вони могли користуватись Apple Music через iTunes, а сьогодні вже музичний плеєр Music, який замінив iTunes.

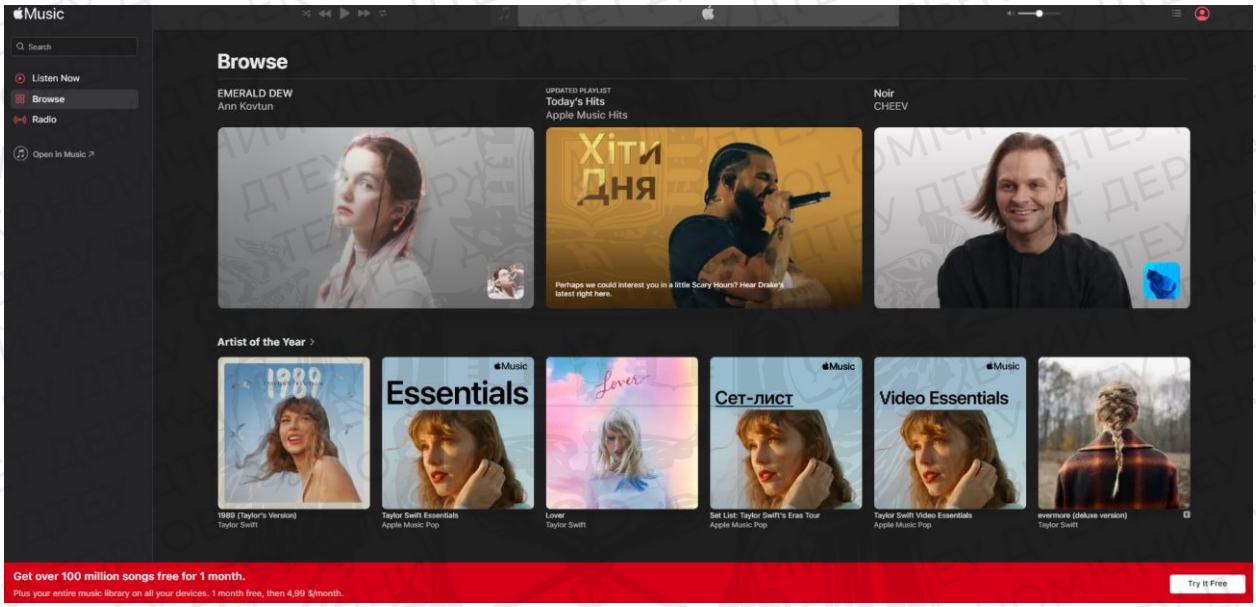

Рисунок 1.6 – Підбірка музики у веб-плеєрі Apple Music

Apple Music також став доступний на різних смарт-спікерах, включаючи продукти, що підтримуються Siri[33] та іншими голосовими помічниками, а також на інших пристроях, таких як смарт-годинники (Apple Watch) та інтегровані автомобільні системи. Таке розширення на різних платформах та пристроях було ключовим кроком у стратегії Apple для забезпечення широкого охоплення та зручності для користувачів, незалежно від їхніх уподобань в обладнанні.

Серед основних переваг Apple Music, можна виділити такі:

- Великий Каталог Музики

Каталог Apple Music включає мільйони пісень, охоплюючи широкий спектр жанрів, від популярних до нішевих. Користувачі мають доступ до великої

різноманітності музики від всесвітньо відомих артистів до незалежних виконавців. Він постійно оновлюється і включає музику з усього світу. Крім автоматизованого персоналізованого підбору музики, в Apple Music також є ретельно відібрані плейлисти від експертів і відомих артистів, що відображають музичні тенденції, жанри, настрої та інші теми.

- Висока Якість Звуку

Apple Music використовує формат AAC (Advanced Audio Codec) з бітрейтом 256 kbps, який забезпечує відмінну якість звуку. Хоча цей бітрейт нижчий, ніж деякі конкуренти пропонують у своїх високоякісних або "Hi-Fi" підписках, формат AAC відомий своєю ефективністю та високою якістю звуку при даній швидкості передачі даних. Цей формат також ефективно зменшує втрату даних, що важливо для збереження якості звуку, особливо при прослуховуванні через бездротові навушники або динаміки. Однією з недавніх інновацій в Apple Music є впровадження Spatial Audio та підтримка Dolby Atmos[34], що пропонує більш об'ємний звуковий досвід.

- Інтеграція з Екосистемою Apple

Інтеграція Apple Music з екосистемою Apple є однією з ключових особливостей цього музичного стрімінгового сервісу, що значно підвищує його привабливість для користувачів продуктів Apple. Сервіс оптимізований для безшовної роботи з iPhone, iPad, iPod Touch, Mac, Apple Watch, Apple TV та HomePod. Це означає, що користувачі можуть легко синхронізувати свої плейлисти, налаштування та вподобання музики між усіма своїми пристроями. Вони можуть використовувати голосового асистента Siri для керування відтворенням музики в Apple Music, включаючи відтворення пісень, пошук артистів, створення плейлистів та навіть отримання рекомендацій на основі своїх вподобань та керувати відтворенням музики прямо зі своїх Apple Watch, що є особливо зручним під час тренувань та інших активностей. Apple Music також включений у пакет Apple One, який об'єднує кілька послуг Apple, включаючи iCloud, Apple TV+, Apple Arcade та інші, в одну підписку.

- Ексклюзивний Контент

Ексклюзивний контент є важливою частиною стратегії Apple Music, що допомагає йому вирізнятися серед конкурентів на ринку музичного стрімінгу. Користувачі Apple Music мають доступ до ексклюзивних музичних відео, документальних фільмів, інтерв'ю з артистами та іншого відеоконтенту, який не доступний на інших платформах, а також доступ до оригінальних шоу, радіопередач та програм, включаючи власну радіостанцію Beats 1, де ведучі та зіркові гості представляють унікальний контент. Apple Music іноді пропонує ексклюзивні живі виступи артистів, а також унікальні записи концертів або музичних сесій. Ці елементи не тільки підвищують привабливість Apple Music для шанувальників музики, але й сприяють залученню нових передплатників, які шукають унікального музичного контенту та досвіду, якого не можна знайти на інших платформах.

- Персоналізація

На основі історії прослуховувань та вподобань користувача, Apple Music створює персоналізовані плейлисти, такі як "For You" та "New Music Mix", які регулярно оновлюються і відображають музичні переваги користувача. Застосування складних алгоритмів дозволяє Apple Music аналізувати вподобання та вибір користувачів, щоб рекомендувати нові пісні, артистів та жанри, які можуть їм сподобатися. Збереження та модифікація плейлистів і підписка на артистів та жанри, дозволяє Apple Music більш точно відповідати на музичні переваги та інтереси своїх користувачів та лідирувати серед конкурентів.

Мінуси Apple Music:

- Відсутність Безкоштовної Версії

Однією з особливостей Apple Music є відсутність традиційної безкоштовної версії, яка є доступною у багатьох інших музичних стрімінгових сервісах, таких як Spotify. Для деяких людей відсутність безкоштовної версії може стати перешкодою для вибору сервісу, особливо якщо вони не готові відразу платити за підписку, адже без безкоштовної версії, потенційні користувачі не

можуть спробувати Apple Music перед тим як вирішити чи варто користуватись цим сервісом. Це створює бар'єр для входу, тому що на ринку є багато безкоштовних версій з рекламою. Але цей крок чітко показує стратегію компанії і її підхід зосередитися на якісному платному контенті та сервісах, а не на масовому охопленні через безкоштовний доступ з рекламою.

- Обмежені Соціальні Функції

Обмежені соціальні функції в Apple Music є однією з аспектів сервісу, які часто порівнюються з можливостями інших платформ, як-от Spotify. На відміну від деяких конкурентів, Apple Music не має інтегрованих соціальних мереж або платформ, що дозволяють користувачам легко поділитися своїми музичними вподобаннями або дізнатися, що слухають їхні друзі.

- Обмежена Сумісність з не-Apple продуктами

Хоча Apple Music доступний на Android та інших платформах, деякі функції можуть бути обмежені або не такі зручні, як на продуктах Apple, а також взагалі відсутня інтеграція з деякими системами та пристроями.

- Вартість Підписки

Ціни на Apple Music порівнянні з іншими провідними стрімінговими сервісами, такими як Spotify та Amazon Music, але вони пропонують додаткові опції або безкоштовні версії з рекламою.

1.3.3 Amazon Music

Amazon Music — це платформа для потокового відтворення музики та онлайн-магазин музики, яким управляє Amazon[35]. Сервіс був запущений в вересні 2007 року як Amazon MP3, стаючи однією з перших великих музичних служб, яка продавала пісні та альбоми без цифрового управління правами[36] (DRM). Це означало, що користувачі могли завантажувати музику та відтворювати її на будь-якому пристрої, що підтримує формат MP3. Це було важливим відмінним фактором у ті часи, коли багато музичних сервісів використовували DRM для обмеження використання музики.

Служба пропонувала понад 2 мільйони пісень від 180 тисяч виконавців та 20 тисяч лейблів, включаючи всі крупні лейбли.

Введення Amazon Cloud Player в 2011 році було важливим кроком у розвитку Amazon Music. Це був хмарним сервісом для зберігання та відтворення музики, який дозволяв користувачам зберігати музику в хмарі. Завдяки хмарному зберіганню, користувачі могли слухати свої улюблені пісні з будь-якого пристрою, що мав підключення до Інтернету, включаючи комп'ютери, смартфони та планшети. Цей сервіс став одним з перших, хто запропонував інтегроване рішення для покупки, зберігання та відтворення музики в хмарі, що було новаторським підходом на той час, адже дозволяло відтворювати музику без необхідності завантаження файлів на пристрій, забезпечуючи зручність і гнучкість.. Він поклав початок переходу від традиційних способів купівлі та зберігання музики до більш гнучкого та доступного цифрового формату.

У 2014 році був запущений Prime Music, це стало ще одним значним кроком у розвитку музичних послуг Amazon. Prime Music став частиною Amazon Prime, популярної підписки, яка вже включала безкоштовну доставку, доступ до Prime Video та інші переваги:

1. Prime Music було безкоштовним доповненням для користувачів Amazon Prime, що додало велику цінність до вже існуючого пакету послуг Prime.
2. На момент запуску Prime Music пропонував доступ до понад мільйона пісень та сотень плейлистів без додаткової плати.
3. Служба пропонувала необмежене відтворення музики без будь-яких додаткових витрат та реклами.
4. Prime Music було легко інтегрувати з Amazon Echo[37] та іншими пристроями, що підтримуються Amazon, що дозволяло користувачам з легкістю відтворювати музику через голосові команди Alexa[38].

Запуск Prime Music допоміг Amazon зміцнити свої позиції на ринку музичних стрімінгових сервісів, залучаючи користувачів через інтеграцію з популярною підпискою Amazon Prime. Це також підкреслило стратегію компанії щодо надання додаткових переваг для членів Prime, збільшуючи вартість їхньої щорічної або місячної плати за підписку.

У жовтні 2016 року відбувся запуск Amazon Music Unlimited, що стало ще одним важливим глобальним розширенням музичних послуг Amazon. Цей сервіс був відповіддю на зростаючу популярність стрімінгових музичних платформ і мав на меті конкурувати з іншими провідними сервісами, такими як Spotify та Apple Music. Це був платний сервіс з більшою бібліотекою пісень та додатковими функціями порівняно з Prime Music, вона налічувала десятки мільйонів пісень і значно збільшила вибір музики. Сервіс включав покращені функції персоналізації, такі як рекомендації пісень та плейлистів, засновані на індивідуальних перевагах користувачів, підтримував голосові команди та був тісно інтегрований з голосовим помічником. Amazon запропонував конкурентоспроможні тарифи на підписку, включаючи знижки для членів Prime та власників пристроїв Echo, що дозволило зайняти більш значну частку на ринку стрімінгових музичних сервісів.

Останнім масштабним оновленням було впровадження Amazon Music HD у вересні 2019 року, який забезпечував покращену якість звуку порівняно зі стандартними стрімінговими сервісами. Він пропонував музику в HD (високої чіткості) та Ultra HD (ще вищої чіткості), що забезпечувало вищу якість звуку порівняно зі стандартними стрімінговими форматами.

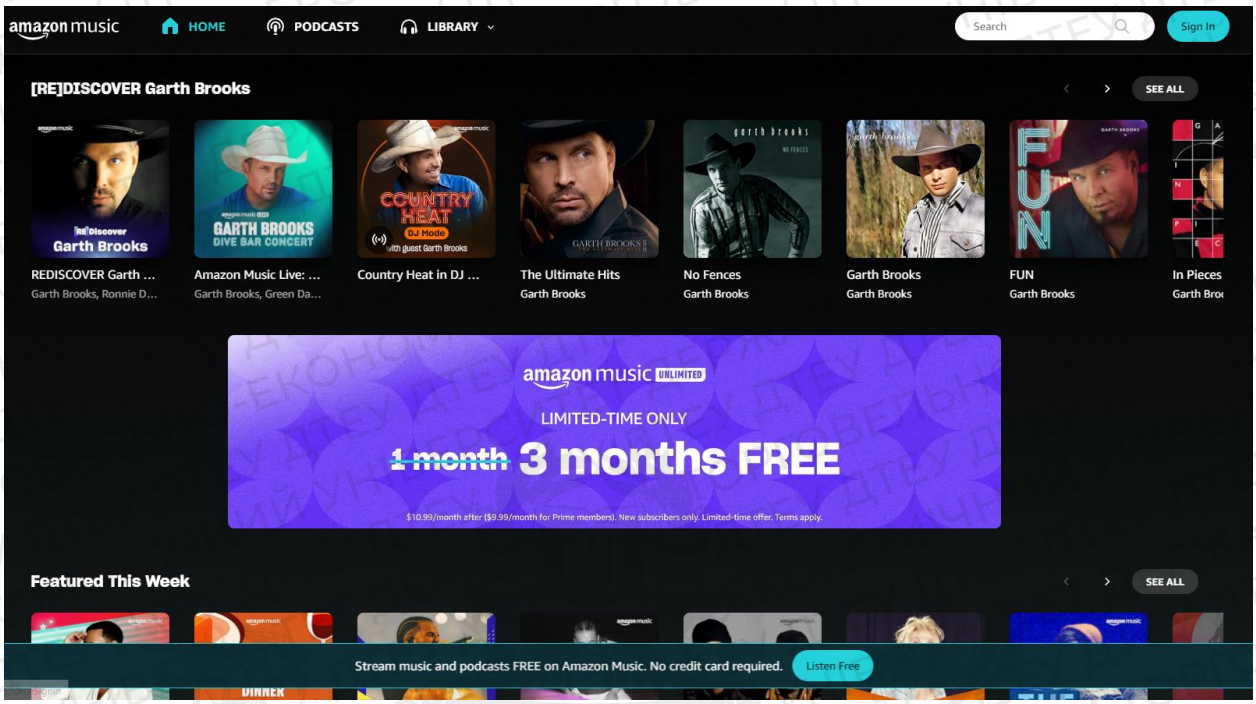

Рисунок 1.7 – Головне меню Amazon Music

Таке аудіо має більший діапазон бітрейтів, що дозволяє слухачам відчути більш деталізований та чистий звук і пісень у такій якості було досить багато, попри ці зміни та оновлення, ціновий діапазон не сильно відрізнявся від конкурентів, так як цей сервіс не є доступним в Україні, ціна в середньому сягає $12.99 на місяць, що є цілковитою нормою для регіонів, де його ним користуються, враховуючи, що безкоштовний період сягає цілих 90 днів, що в тричі більше аніж в конкурентів.

Рисунок 1.8 – Порівняльна характеристика цін Amazon Music та Spotify

Станом на 2023 рік, Amazon Music має понад 68 мільйонів передплатників по всьому світу, більша частина слухачів, а саме 48,3 мільйона із Сполучених Штатів, іншу частку займає Великобританія, Німеччина, Японія, Франція, Італія, Мексика та Іспанія. Amazon Music доступний лише у 49 країнах по всьому світу, що є значно меншою кількістю порівняно з конкурентами, такими як Apple Music, який надає свої послуги у 167 країнах. Amazon Music, як сервіс потокового відтворення музики, має ряд переваг і недоліків у порівнянні з конкурентами, такими як Spotify, Apple Music, і YouTube Music, наприклад, серед переваг:

- Великий музичний каталог

Завдяки наявності понад 100 мільйонів пісень, Amazon Music пропонує один з найбільших каталогів серед музичних стрімінгових сервісів.

- Модель Freemium

Модель включає безкоштовну версію, яка пропонує користувачам основний доступ до музичного каталогу сервісу. Ця версія має обмеження, відсутня можливість вибору конкретних пісень та присутність реклами. Однак, вона дає користувачам можливість ознайомитися з сервісом без витрат.

- Високоякісне аудіо

Сервіс пропонує мільйони пісень у високоякісному та ультрависокоякісному форматах з високим бітрейтом.

- Розширений не-музичний каталог

Amazon Music включає також подкасти та інший ексклюзивний чи оригінальний контент від відомих зірок, а також музичні відео.

- Міцна інтеграція з продуктами Amazon

Серед мінусів можна виділити:

- Відсутність інтернет-радіо
- Обмеження безкоштовної та Prime версій

У цих версіях немає можливості вибору пісень, і вони працюють в режимі випадкового відтворення. Функція відтворення на вимогу доступна лише для підписників Unlimited

- Обмежена регіональна доступність

Amazon Music не настільки широко доступний, як його конкуренти.

- Обмеження високоякісного аудіо

Обмеження високоякісного аудіо в Amazon Music полягають у тому, що його не можна стрімити через веб-браузер. Таким чином, користувачі, які хочуть слухати музику у високій якості, мають її завантажувати.

1.3.4 YouTube Music

YouTube Music — це музичний стрімінговий сервіс, розроблений компанією Google[39]. Він був створений і запущений Google у листопаді 2015 року. Цей сервіс забезпечує стрімінговий доступ до широкого спектру музики, включаючи офіційні треки, музичні відео, ремікси, кавери та живі виступи. Повністю інтегрований з величезною бібліотекою YouTube, що дозволяє користувачам легко знаходити та досліджувати різноманітну музику. YouTube Music пропонує як безкоштовну версію з рекламою, так і платні підписки, що надають доступ до додаткових функцій, таких як безрекламне прослуховування, відтворення музики в фоновому режимі та можливість завантаження музики для офлайн прослуховування.

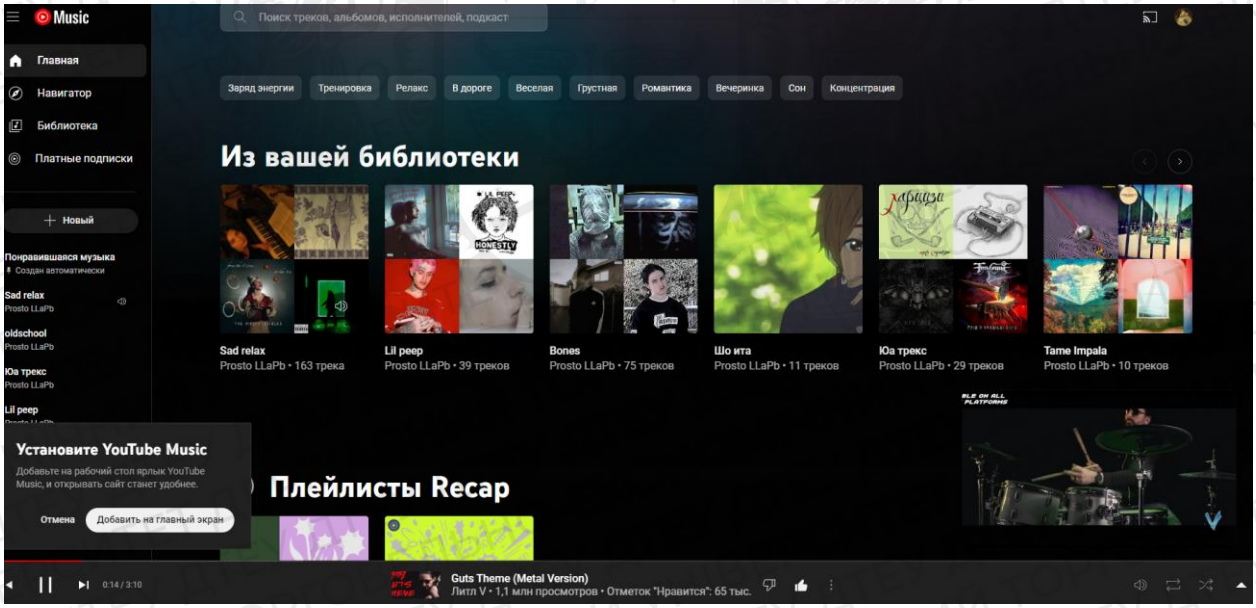

Рисунок 1.9 – Головна сторінка YouTube Music

Станом на 2023 рік, YouTube Music має значну користувацьку базу, яка нараховує 2.6 мільярда користувачів, з яких 80 мільйонів є платними передплатниками. Сервіс доступний у понад 100 країнах по всьому світу, що робить його одним з найбільш широко доступних музичних стрімінгових сервісів.

Рисунок 1.10 – Мапа доступності послуги YouTube Music у світі

Платна підписка YouTube Music Premium, дає змогу насолоджуватися музикою без перерв на рекламу, що забезпечує неперервне прослуховування. Також ця функція дозволяє користувачам продовжувати слухати музику, навіть коли вони використовують інші додатки або блокують екран свого пристрою. Користувачі підписки можуть завантажувати пісні, альбоми та плейлисти для прослуховування без Інтернету і мають доступ до ексклюзивного контенту. В Україні вартість підписки 79 гривень, що позитивно виділяє цей сервіс серед інших для нашого регіону. Серед інших переваг:

- Доступ до обширної бібліотеки

Бібліотека включає понад 50 мільйонів пісень це не тільки офіційні записи від артистів, але й музичні відео, живі виступи, кавери та ремікси. Великий вибір музичного контенту на YouTube Music дозволяє користувачам відкривати для себе нову музику, повертатися до улюблених класичних треків та насолоджуватися різноманітністю жанрів та стилів.

- Прослуховування в фоновому режимі

Ця функція дозволяє користувачам продовжувати відтворення музики, навіть коли вони використовують інші додатки або коли екран пристрою заблокований. Це особливо зручно, коли користувачі слухають музику на мобільному пристрої і хочуть використовувати інші додатки або функції пристрою без переривання музики. Наприклад, вони можуть переглядати соціальні мережі, читати електронні листи або використовувати навігацію, продовжуючи слухати музику в фоновому режимі. Така функціональність робить YouTube Music Premium більш гнучким та зручним для повсякденного використання.

- Завантаження музики для офлайн прослуховування

Це ще одна важлива перевага платної підписки YouTube Music Premium. Ця функція дозволяє користувачам зберігати пісні, альбоми та плейлисти на своїх мобільних пристроях, щоб вони могли слухати музику без необхідності підключення до Інтернету.

До мінусів YouTube Music можна віднести наступне:

- Обмежена якість аудіо

Одним із найбільших недоліків YouTube Music є те, що максимальна якість стрімінгу становить 128Kbps у форматах MP3 і AAC+, що є нижче, ніж пропонують інші музичні сервіси, такі як Spotify, Apple Music та Amazon Music.

- Обмеження функцій в безкоштовній версії

Користувачі безкоштовної версії не можуть відтворювати музику у фоновому режимі або завантажувати пісні для офлайн прослуховування, а також вона містить реклами, які можуть переривати прослуховування музики.

- Обмежений алгоритм рекомендацій

Алгоритм рекомендацій YouTube Music не такий точний та різноманітний, як у конкурентів.

1.3.5 SoundCloud

SoundCloud[40], заснований у 2007 році, є одним із провідних музичних стрімінгових сервісів, особливо відомим своєю спільнотою незалежних музикантів, продюсерів, DJ-ів та аудіофілів. Станом на 2023 рік, SoundCloud має понад 175 мільйонів щомісячних користувачів, з яких приблизно 76 мільйонів є платними передплатниками. Це робить SoundCloud однією з найбільших платформ для стрімінгу музики, особливо популярною серед незалежних артистів та шанувальників нової музики. Найвищий рівень популярності в Сполучених Штатах, де він становить близько 27% всіх відвідувань. Великобританія слідує за США з 6% відвідувань, а Німеччина та Франція вносять 5% і 3% відповідно до загального обсягу відвідувань.

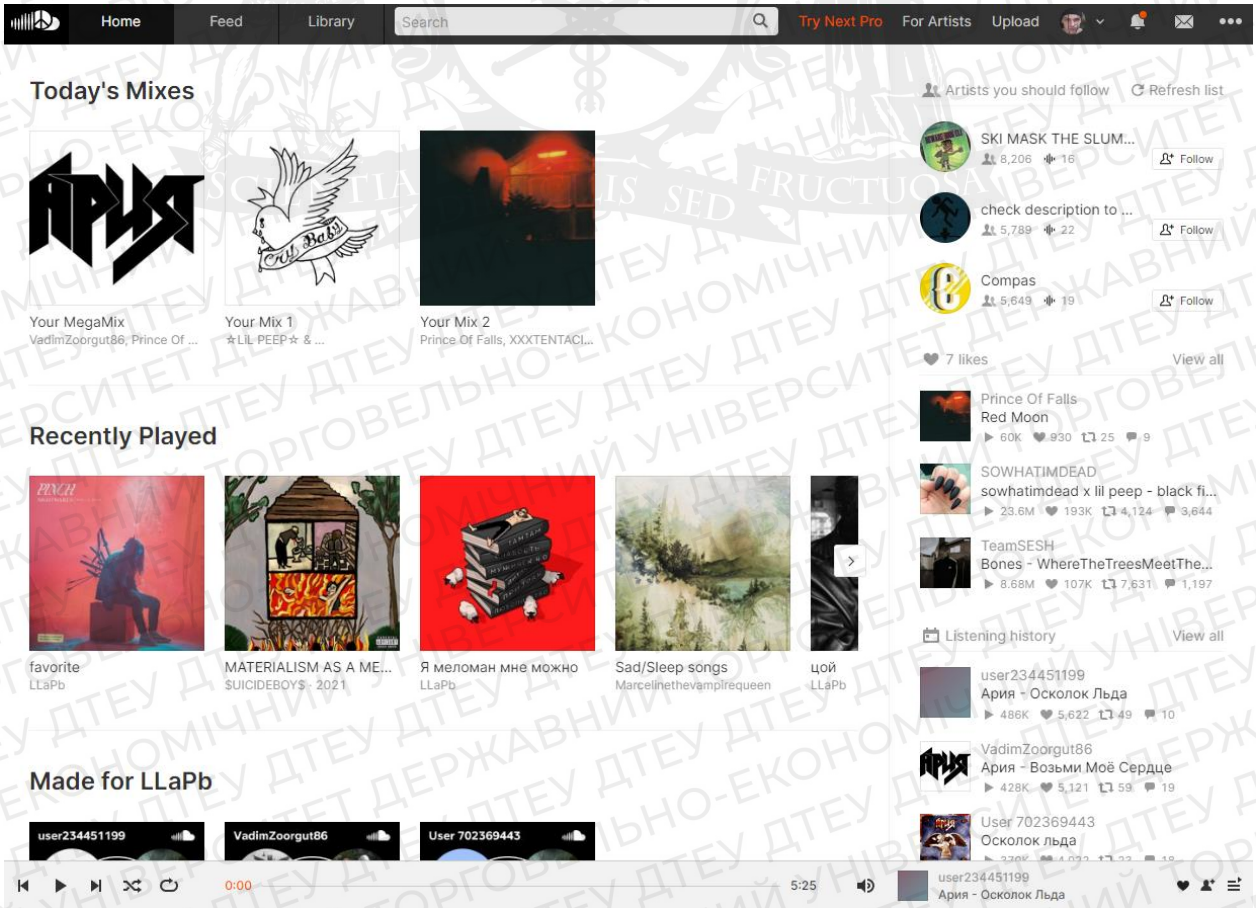

Рисунок 1.11 – Головна сторінка SoundCloud

Сервіс сприяє новій музиці, він відомий тим, що допомагає відкривати нових артистів та популяризувати нові музичні жанри, ставши платформою для пошуку та популяризації музичних новинок. Це популярне місце для незалежних музикантів, продюсерів та артистів, що використовують платформу для публікації своєї оригінальної музики, що створює унікальний каталог музики, який включає велику кількість невиданих, експериментальних та незвичайних треків. Одною з його особливостей, є можливість коментувати треки безпосередньо на часовій шкалі пісні, що створює унікальну взаємодію між артистами та їхніми слухачами. Артисти мають можливість дозволити користувачам безкоштовно завантажувати їхні треки, що не завжди можливо в інших стрімінгових сервісах.

Існує два основних варіанти використання своєї платформи: безкоштовний і платний. SoundCloud Go — це платна підписка, яка пропонує кращу якість звуку порівняно з безкоштовною версією, відсутність реклами, завантаження для офлайн прослуховування та надає доступ до всієї бібліотеки треків на платформі. SoundCloud Go+ — це розширена платна підписка, яка пропонує додаткові переваги порівняно з базовою версією SoundCloud Go. Вона звернена до більш вимогливих користувачів, які шукають найкращу якість звуку та хочуть мати доступ до всього обсягу музики на платформі. Ця підписка є вибором для аудіофілів та активних користувачів SoundCloud, які хочуть максимально використовувати можливості платформи. SoundCloud Go коштує $4.99 на місяць, вартість SoundCloud Go+ становить $9.99 на місяць.

Мінусами сервісу можна назвати якість звуку, оскільки платформа зосереджена на незалежних артистів та продюсерів, з цієї ж причини можуть бути відсутні деякі великі імена. Нестабільність доходу для артистів є однією з важливих проблем, яка турбує багатьох творців контенту платформи, для деяких артистів модель монетизації може бути нестабільною або невигідною. Дохід артистів на SoundCloud часто залежить від кількості стрімів їхніх

треків. Це може бути непросто для нових або менш відомих артистів, які ще намагаються залучити слухачів. Виплати за стрім на SoundCloud традиційно вважаються нижчими порівняно з іншими платформами, такими як Spotify або Apple Music та через велику кількість контенту на платформі є проблема високої конкуренції, що ускладнює артистам процес залучення аудиторії та генерування доходу. Також певним мінусом є інтерфейс з своєю складною навігацією, він здається заплутаним та перевантаженим, що погіршує зручність та точність навігації. Але SoundCloud все одно залишається важливою платформою для відкриття нової музики та підтримки незалежних артистів, хоча й має деякі обмеження в плані функціональності та якості звуку.

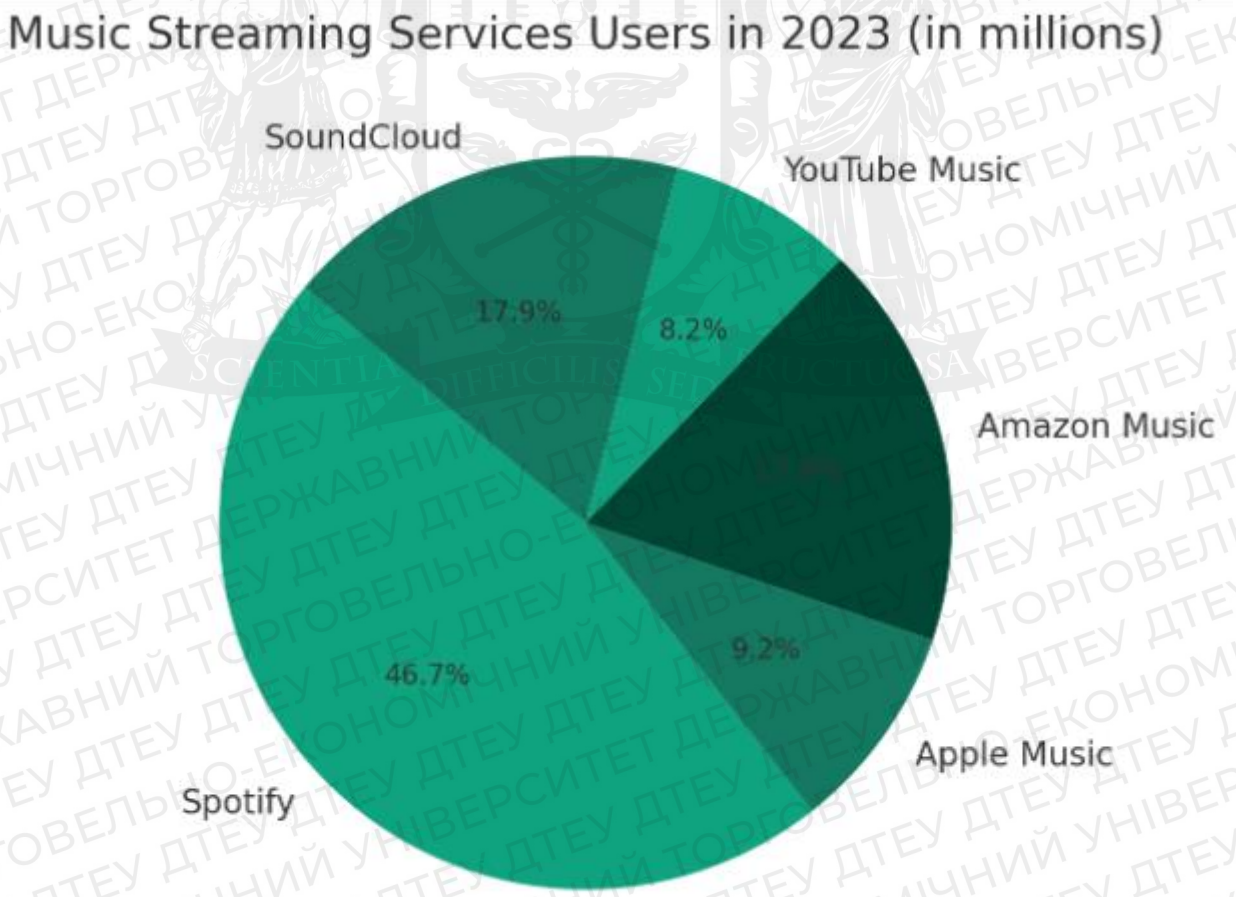

Рисунок 1.12 – Кругова діаграма розподілу користувачів основних музичних стрімінгових сервісів у 2023 році

РОЗДІЛ 2. Модель Веб-додатку для Прослуховування Музики

2.1 Технології Веб-Розробки

Основною технологією у розробці є HTML[41] (HyperText Markup Language) це фундаментальна мова розмітки, яка використовується для створення та структурування вмісту на веб-сторінках. Вона формує основу будь-якого веб-додатку, визначає основні елементи сторінки, заголовки, параграфи, списки та інші. Особливістю застосування HTML є використання "тегів" та "елементів" для організації та представлення різних частин веб-сторінки, включаючи текст, зображення, посилання і форми. Такі теги як :

- <header>, <footer>, <nav>, та <section> допомагають в структуруванні сторінки на логічні секції.
- , , <p>, та <h1> до <h6>, визначають різні частини вмісту.
- <img>, <audio>,<video> дозволяють вбудовувати зображення, аудіо, відео та інші мультимедійні елементи.
- елемент <a> дозволяє створювати гіперпосилання, які з'єднують різні веб-сторінки або розділи в межах однієї сторінки.
- елементи форм <form>, <input>, <textarea>, <button>, використовуються для збору даних від користувачів.

Застосування HTML5[42] у веб-додатках для прослуховування музики, використовується для вбудовування аудіофайлів безпосередньо в веб-сторінку, що є ключовим для музичних веб-додатків. Атрибути, як-от controls, autoplay, та loop, дозволяють налаштувати поведінку аудіоплеєра. Також у HTML5 були введені семантичні теги, що допомагають у створенні більш доступного та логічно організованого вмісту та полегшують роботу з веб-сторінкою не тільки розробникам, але й пошуковим системам та читачам

екрану. Тобто можна зробити висновок, що HTML виконує основні завдання у контексті розробки веб-додатку для прослуховування музики:

- створення основи інтерфейсу користувача, де будуть відображатися музичні треки, плейлисти та елементи управління відтворенням.
- використання аудіо тегів для інтеграції музичного плеєра безпосередньо в сторінку.
- організація контенту таким чином, щоб він був доступний та зручний для користувачів

Він є відправною точкою для створення будь-якого веб-додатку і лежить в основі структури та контенту веб-сторінок. У сполученні з CSS та JavaScript, HTML стає потужним інструментом для розробки динамічних та інтерактивних веб-додатків.

Невід'ємною частиною веб-розробки є технологія CSS[43] як основний інструмент стилізації та оформлення веб-сторінок. CSS (Cascading Style Sheets) є мовою стилів, яка використовується для визначення візуального дизайну та макету веб-сторінок. Це ключовий інструмент у веб-розробці, який дозволяє розділити вміст та оформлення веб-сайту.

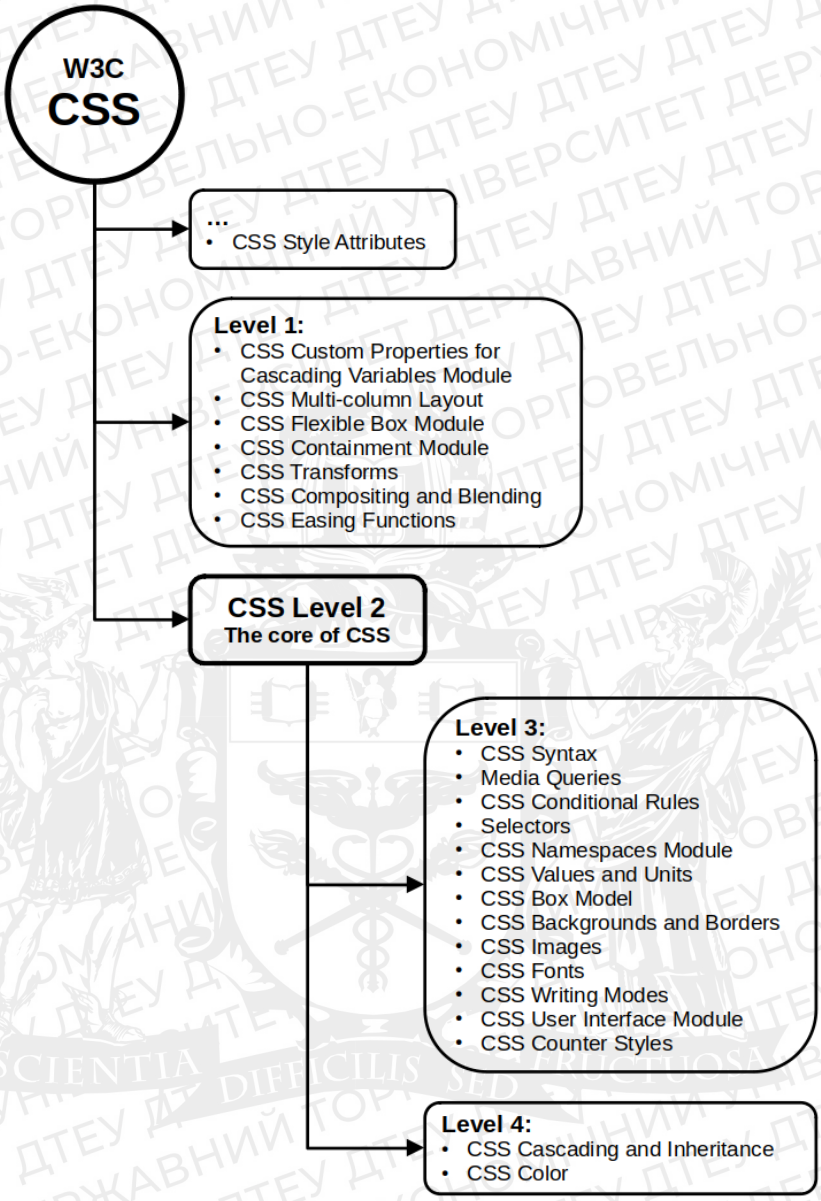

Рисунок 2.1 – CSS Snapshot 2021

Однією з основних властивостей та застосування CSS є структурування та стилізація. CSS дозволяє присвоїти стилі елементам HTML, включаючи кольори, шрифти, відступи, та макетування, що є фундаментальним для створення візуально привабливих веб-сторінок. Селектори, такі як класи та ідентифікатори, дозволяють цілеспрямовано застосовувати стилі до специфічних елементів на сторінці. Він використовує медіа-запити для

адаптації веб-сторінок до різних розмірів екранів та пристроїв, що є важливим для забезпечення гнучкості та доступності веб-додатку на мобільних пристроях. Способи розмітки CSS надають потужні інструменти для створення комплексних макетів, які динамічно адаптуються під розмір екрану. Одним з таких способів є Flexible Box Layout, головна його мета - надати контейнеру здатність змінювати ширину/висоту елементів, щоб найкраще заповнити доступний простір. Ще один спосіб Grid Layout, він є двовимірною системою розмітки, яка дозволяє створювати складні макети з використанням рядків та стовпців. Вона дозволяє більш детальне позиціонування елементів та краще контролюється, ніж інші методи розмітки. CSS покращує інтерфейс та інтерактивності за допомогою візуальних ефектів та анімації. Вони відіграють ключову роль у залученні уваги користувачів та створенні плавного та інтуїтивно зрозумілого досвіду взаємодії. Вони використовуються для акцентування змін у музичному плеєрі, підкреслення активних елементів, або як візуальні підказки для користувацьких дій.

CSS є важливою складовою в процесі веб-розробки, яка надає необхідні інструменти для створення естетично привабливих, зручних та функціональних веб-інтерфейсів. У музичних веб-додатках, CSS відіграє ключову роль у формуванні візуального досвіду та інтерактивності користувача. Його класи вдосконалюють взаємодії користувачів з інтерфейсом, динамічні елементи сповіщають про поточний стан відтворення або зміни плейлисту, стилізація елементів плеєра, включаючи кнопки відтворення, регулятори гучності, слайдери треків, та використання іконок і графічних елементів, які відповідають загальному дизайну додатку.

Останньою базовою технологією є JavaScrip[44] яка представляє собою центральну частину стеку веб-розробки, особливо у контексті створення інтерактивних веб-додатків, таких як платформи для прослуховування музики. Використання JavaScript в таких додатках дозволяє реалізувати

різноманітні функції, від управління інтерфейсом користувача до обробки аудіоданих. JavaScript дозволяє динамічно змінювати вміст та структуру веб-сторінок за допомогою маніпуляцій з документом об'єктної моделі (DOM). Це включає зміну тексту, додавання або видалення елементів, зміну стилів тощо. Він представляє структуру документа у вигляді деревоподібної моделі, дозволяючи програмам змінювати структуру, стиль та вміст веб-сторінок. JavaScript широко використовує DOM для маніпуляцій з веб-сторінкою, що робить можливим динамічне змінення контенту та інтерфейсу, у контексті веб додатку це:

- списки пісень, плейлистів та інформації про артистів без перезавантаження сторінки.
- зміна властивостей аудіо-елементів (наприклад, відтворення, пауза) та оновлення інтерфейсу плеєра (наприклад, бар прогресу треку).
- реалізація інтерактивних функцій, як-от відгуки на дії користувача, спливаючі вікна, динамічні анімації.

Крім цього, JavaScript використовується для обробки користувацьких подій, таких як кліки мишею, натискання клавіш, жести на сенсорних екранах, забезпечуючи реакцію додатку на дії користувача. У сучасній веб-розробці, JavaScript відіграє ключову роль у створенні аудіо-функціональності, відтворення аудіо, обробка аудіо, аудіо візуалізація та інтеграція з різними медіа. З приводу відтворення аудіо, то відбувається взаємодія з елементом HTML для контролювання відтворення, зупинки, перемотування та налаштування гучності аудіо. Щодо обробки аудіо, то використовується спеціальний інтерфейс Web Audio API, який дозволяє розробникам обробляти і маніпулювати аудіо в браузерах у реальному часі. Він надає широкі можливості для створення складних аудіо додатків від простого відтворення музики до складної обробки та інтерактивності, цей API

дозволяє розробникам підняти веб-додатки на новий рівень аудіо функціональності та креативності. Застосування Web Audio API:

1. Створення Музичних Веб-Додатків:
- Реалізація функціоналу музичних плеєрів, включаючи відтворення, паузу, перемотування та регулювання гучності.
- Створення інструментів для цифрового аудіо редагування та мікшування.
2. Обробка та Маніпуляція Аудіо:
- Додавання ефектів до аудіо сигналів, наприклад, реверберація, ехо, фільтри.
- Створення аудіо візуалізацій, аналізуючи аудіо сигнали для відображення їх у візуальному форматі.
3. Розробка інтерактивних аудіо додатків:
- Інтеграція аудіо з іншими веб-технологіями для створення інтерактивних аудіо, таких як ігри, музичні інсталяції.
- Створення освітніх аплікацій для вивчення музики та звуку.

JavaScript у поєднанні з Web Audio API може аналізувати аудіо сигнали для створення візуалізацій, наприклад, зображення частотного спектру або візуального відображення ритму музики. А його інтеграція з різними медіа дає змогу синхронізувати аудіо з відео контентом, та створювати інтерактивні музичні додатки, де користувачі можуть взаємодіяти з аудіо контентом, наприклад, вибір треків, налаштування плейлистів тощо.

2.2 Програмно-технічні засоби розробки веб-додатку для прослуховування музики

Крім основних фронтенд технологій, розробка веб-додатку для прослуховування музики вимагає використання різноманітних програмно-

технічних засобів, наприклад фреймворки та бібліотеки. Вони є незамінними інструментами в сучасній веб-розробці, оскільки вони спрощують процес створення додатків, підвищують продуктивність розробників та забезпечують більшу стабільність і безпеку продукту.

Фреймворк у веб-розробці — це комплексний набір інструментів, бібліотек та певних правил, які використовуються для розробки програмного забезпечення. Фреймворки надають структурований шаблон, який допомагає розробникам організувати свій код та полегшує процес розробки. Структура фреймворків надає певну архітектуру, якої слід дотримуватися, вона включає патерни проектування[45], моделі управління програмами тощо. Вони часто містять бібліотеки з повторно використовуваним кодом, що знижує необхідність писати багато стандартних елементів програми з нуля. Фреймворки зазвичай супроводжуються документацією, спільнотою користувачів та іншими ресурсами, які допомагають у вирішенні проблем.

Бібліотека в програмуванні — це збірка функцій або допоміжних інструментів, які розробник може використовувати для виконання певних завдань. На відміну від фреймворків, бібліотеки не накладають структуру на весь проект, а пропонують специфічний функціонал, який можна використовувати за бажанням. Бібліотеки надають певні функції, які можна "викликати" у власному коді, щоб виконати конкретні завдання (наприклад, аналіз даних, маніпуляції з текстом). Вони дозволяють більшу гнучкість, оскільки розробники вільні вибирати, які функції використовувати, а які ігнорувати. Бібліотеки можна легко інтегрувати у різні частини проекту без необхідності змінювати загальну структуру.

Однією з найпопулярніших відкритих JavaScript бібліотек є React[46], створена та підтримувана Facebook, використовується для побудови користувацьких інтерфейсів, особливо в складних веб-додатках з динамічними даними. React базується на ідеї компонентів — незалежних блоків, які утворюють інтерфейс додатку. Бібліотека дозволяє описувати, як компоненти виглядають у залежності від їхнього стану, не заморочуючись з

деталями реалізації. Вона чудово підходить для створення динамічних веб-додатків, де потрібно управляти складними станами та інтерактивними інтерфейсами завдяки своїй гнучкості, ефективності та потужним можливостям.

Одним з провідних фреймворків для розробки веб-додатків є Angular[47], розроблений та підтримуваний Google. Це повнофункціональний фреймворк, який використовує TypeScript (надмножина JavaScript) і призначений для розробки як простих, так і складних односторінкових додатків (SPA). Angular організує додатки як колекцію компонентів, кожен з яких є сукупністю HTML шаблону та класу TypeScript, що керує цим шаблоном. Компоненти можуть, налаштовані та комбіновані для створення складних інтерфейсів. Він розділяє функціональність на модулі, що спрощує управління кодом та має вбудований маршрутизатор, який дозволяє керувати навігацією між різними сторінками та виглядами в односторінковому додатку.

Бекенд розробка відноситься до серверної частини веб-додатку, де відбувається обробка даних, управління базами даних, автентифікація користувачів, і конфігурація серверних ресурсів. Це ключовий компонент веб-додатку, що забезпечує його функціонування та взаємодію з фронтендом. Серед основних технологій бекенд розробки можна виділити Node.js[48] — це потужне серверне середовище, яке використовує JavaScript, зазвичай використовуваний для фронтенд розробки, для створення бекенд додатків. Воно дозволяє розробникам використовувати єдину мову програмування для розробки всього веб-додатку.

RESTful API[49] (Representational State Transfer Application Programming Interface) — це стиль архітектури для розробки веб-сервісів. REST використовує стандартні HTTP методи і є одним з найпопулярніших підходів для створення веб-сервісів завдяки своїй простоті, легкості використання та сумісності з веб-технологіями.

Реляційні бази даних MySQL[50]. Вони є однією з найпопулярніших реляційних систем управління базами даних (СУБД). Використання MySQL у розробці додатків забезпечує надійне, зручне та ефективне управління даними, які організовані у вигляді таблиць і взаємодій з допомогою SQL (Structured Query Language). Широко використовується у веб-розробці для зберігання даних користувачів, контенту, транзакцій та часто поєднується з популярними веб-технологіями та фреймворками Node.js. MySQL ефективно справляється з великими обсягами даних та високою кількістю одночасних запитів, що робить його підходящим для великих та складних додатків. Дані зберігаються у вигляді таблиць, де кожен рядок представляє запис, а стовпці - атрибути цього запису. Модель дозволяє легко створювати взаємозв'язки між різними таблицями, що забезпечує інтеграцію та цілісність даних.

Системи контролю версій Git. Git є однією з найпопулярніших систем контролю версій, використовуваною у розробці програмного забезпечення. Це відкритий інструмент, який дозволяє розробникам ефективно управляти змінами у коді, співпрацювати з іншими та відстежувати історію проекту протягом часу.

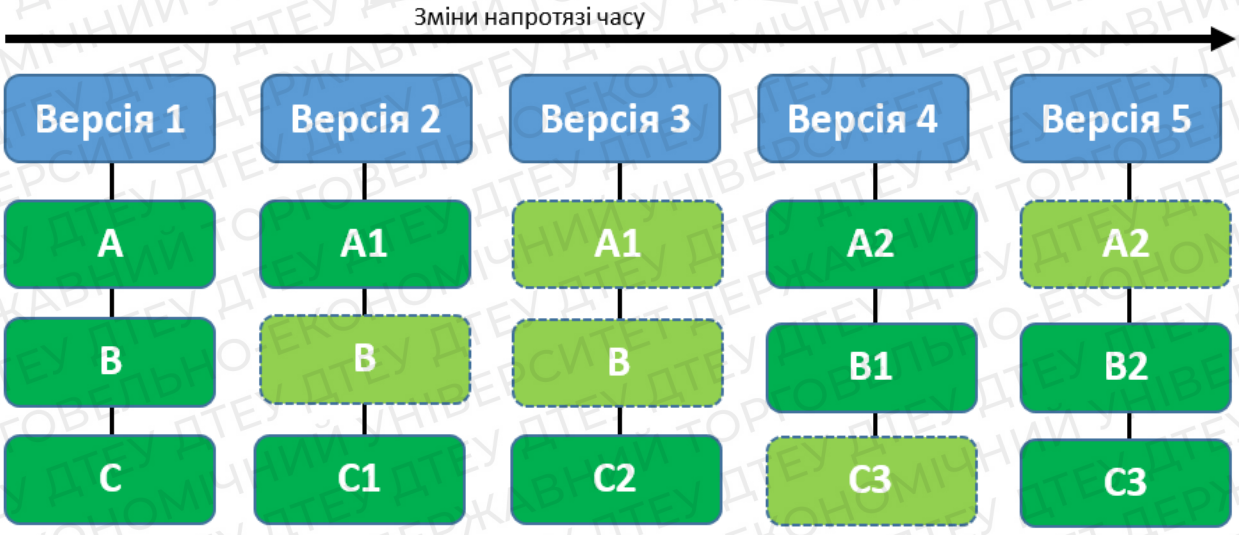

Рисунок 2.2 – Дані, як зліпки стану проєкту в часі

У відмінності від централізованих систем, Git є розподіленим, що означає, що кожен розробник має локальну копію всієї історії репозиторію. Це забезпечує високу швидкість роботи та незалежність від центрального сервера. Він дозволяє розробникам створювати різні гілки (branches) для розробки окремих функцій або експериментів, що спрощує процес розробки та знижує ризик для основного коду, а також зберігає повну історію змін, дозволяючи повернутися до будь-якої версії коду, порівняти зміни між різними версіями та виявити авторів конкретних змін.

2.3 Опис програмного продукту

Для розробки свого програмного продукту було використано фреймворк Nextjs[51]. Next.js — це популярний фреймворк для розробки серверно-рендерених JavaScript додатків, заснований на React. Розроблений компанією Vercel, Next.js надає гнучкість для створення як статичних, так і динамічних веб-додатків, полегшуючи процес розробки і забезпечуючи високу продуктивність. Основною особливістю фреймворку є здатність до серверного рендерення (Server-Side Rendering, SSR), що покращує швидкість завантаження сторінок та SEO. Серверний рендеринг є методом веб-розробки, де вміст веб-сторінки генерується на сервері, а не в браузері користувача. Це протилежно клієнтському рендеренню (Client-Side Rendering), де веб-сторінка будується в браузері користувача, зазвичай за допомогою JavaScript. SEO, або пошукова оптимізація, це процес покращення видимості веб-сайту або веб-сторінки в органічних (неоплачуваних) результатах пошукових систем, таких як Google, Bing, та інші. Мета SEO — збільшити кількість та якість трафіку на веб-сайт через пошукові системи. Nextjs дозволяє реалізувати повністю готовий веб-сайт, тому що підтримує статичну генерацію сторінок, дозволяючи попередньо генерувати сторінки під час побудови додатку. Надає просту систему

маршрутизації, засновану на файловій системі, яка спрощує налаштування маршрутів у додатку та автоматично оптимізує додатки для кращої продуктивності, включаючи розбиття коду, оптимізацію зображень та багато іншого.

Для стилізації компонентів веб-сайту використовується CSS-фреймворк «Tailwind», який використовує готові класи стилів, які можна використовувати прямо у розмітці без необхідності прямої роботи з CSS файлами. Tailwind CSS - це низькорівневий фреймворк. Це означає, що на відміну від інших CSS-фреймворків, таких як Bootstrap і Materialize, Tailwind не пропонує повністю стилізовані компоненти, такі як кнопки, випадаючі списки і навігаційні панелі. Натомість він пропонує утилітарні класи, що дає змогу створювати власні багаторазові компоненти.

У проекті було реалізовано можливість реєстрації та авторизації для користувачів, працюючи як “Backend-as-a-service”. Backend-as-a-Service (BaaS), іноді відомий як "cloud backend", це модель, за якою розробники делегують багато з задач бекенду до зовнішнього постачальника сервісів. Ці сервіси зазвичай включають управління базами даних, автентифікацію користувачів, хостинг файлів, push-сповіщення та інші функції.

Для даних цілей було використано сервіс “Supabase”[52], який базується на “PostgreSQL”. PostgreSQL є потужною відкритою системою управління реляційними базами даних. Він відомий своєю надійністю, гнучкістю та підтримкою розширених функцій, що виходять за рамки традиційних реляційних баз даних. PostgreSQL широко використовується у різноманітних додатках, від мобільних додатків до великих веб-сайтів і корпоративних систем.

Взаємодія із додатком відбувається за допомогою API, яке надає даний сервіс, за допомогою REST-запитів що є ключовим компонентом в архітектурі RESTful API. Також, є можливість реалізації реального часу через WebSocket з'єднання, що дозволяє отримувати миттєві оновлення даних у реальному часі. Це протокол, який забезпечує двосторонній канал зв'язку

через одне TCP-з'єднання. Він дозволяє взаємодію між веб-клієнтом (наприклад, браузером) та сервером в реальному часі, що є ідеальним для додатків, що потребують швидкого обміну даними.

База даних для проекту була створена прямо на сервісі Supabase. Було використано готовий шаблон БД, який містить готові сутності користувачів, продуктів, підписок, цін і клієнтів. Було додано таблицю “songs” та “liked_songs” для можливості зберігання пісень в базі даних.

На сервісі також було реалізовано можливість оформлення преміум підписки. Для її реалізацію використано сервіс “Stripe Dashboard”[53], який надає можливість управляти фінансами, аналізувати транзакції, налаштовувати оплати та отримувати інформацію щодо фінансової діяльності, пов'язаної з платежами. Інтеграція Stripe з веб-додатком відбувається за допомогою API, який надається Stripe SDK. Для того, щоб підписка з’явилась на сайті, вона спочатку створюється на сайті Stripe. Після цього, за допомогою REST-запитів з API, ці підписки відображаються на сайті і користувач має змогу їх оформити. Процес оформлення підписки відбувається у тестовому режимі, з використанням фейкових платіжних даних, для емуляції реального процесу оплати.

Після оформлення підписки, користувач має змогу завантажувати пісні, які зберігатимуться у базі даних на Supabase, прослуховувати їх чи добавляти до улюблених пісень.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ Web-ДОДАТКУ ДЛЯ ПРОСЛУХОВУВАННЯ МУЗИКИ

3.1. Тестування та аналіз результатів

Для початку необхідно налаштувати середовище додатку. У терміналі VS Code запускаємо наступні команди, упевнившись, що вказуємо назву папки, де будемо розроблювати свою програму.

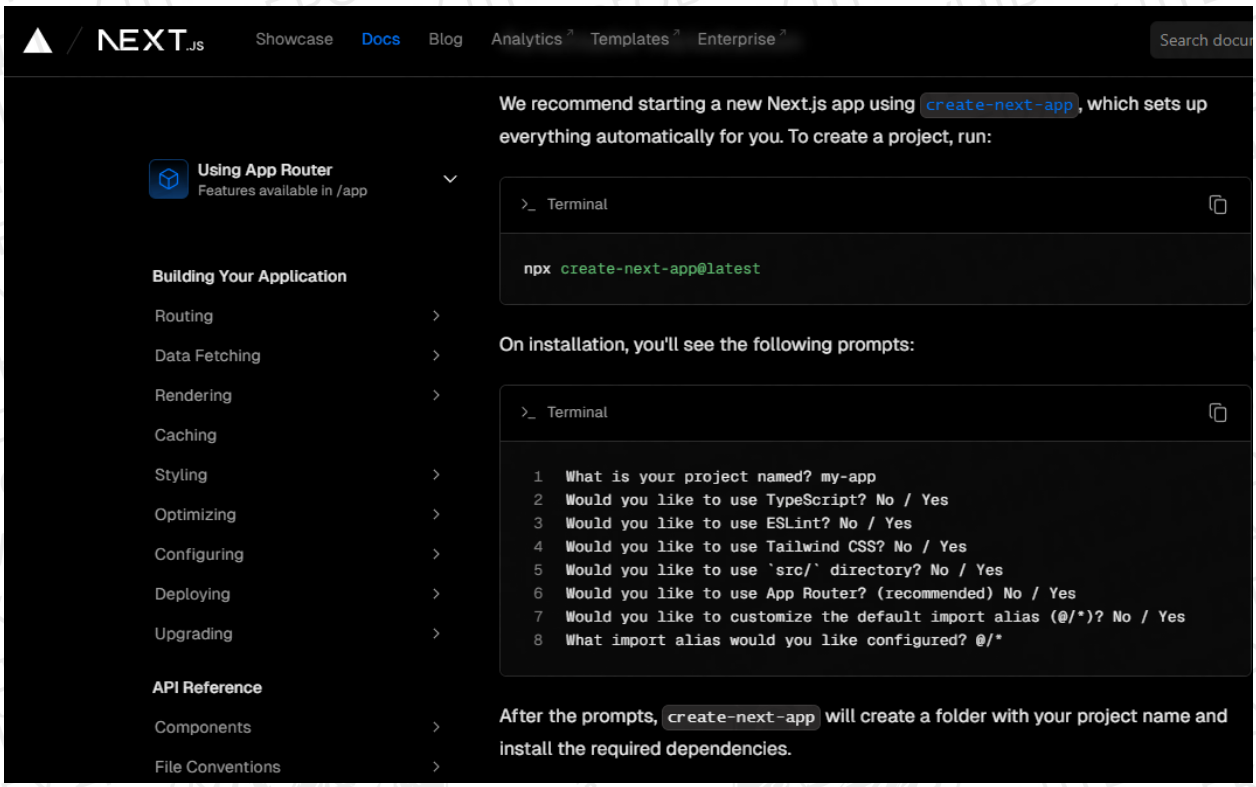

Рисунок 3.1 – Автоматичне встановлення Next.js

Next.js тепер поставляється з конфігурацією CSS TypeScript, ESLint і Tailwind за замовчуванням.

Додатково можна використовувати каталог src у корені проекту, щоб відокремити код програми від конфігураційних файлів.

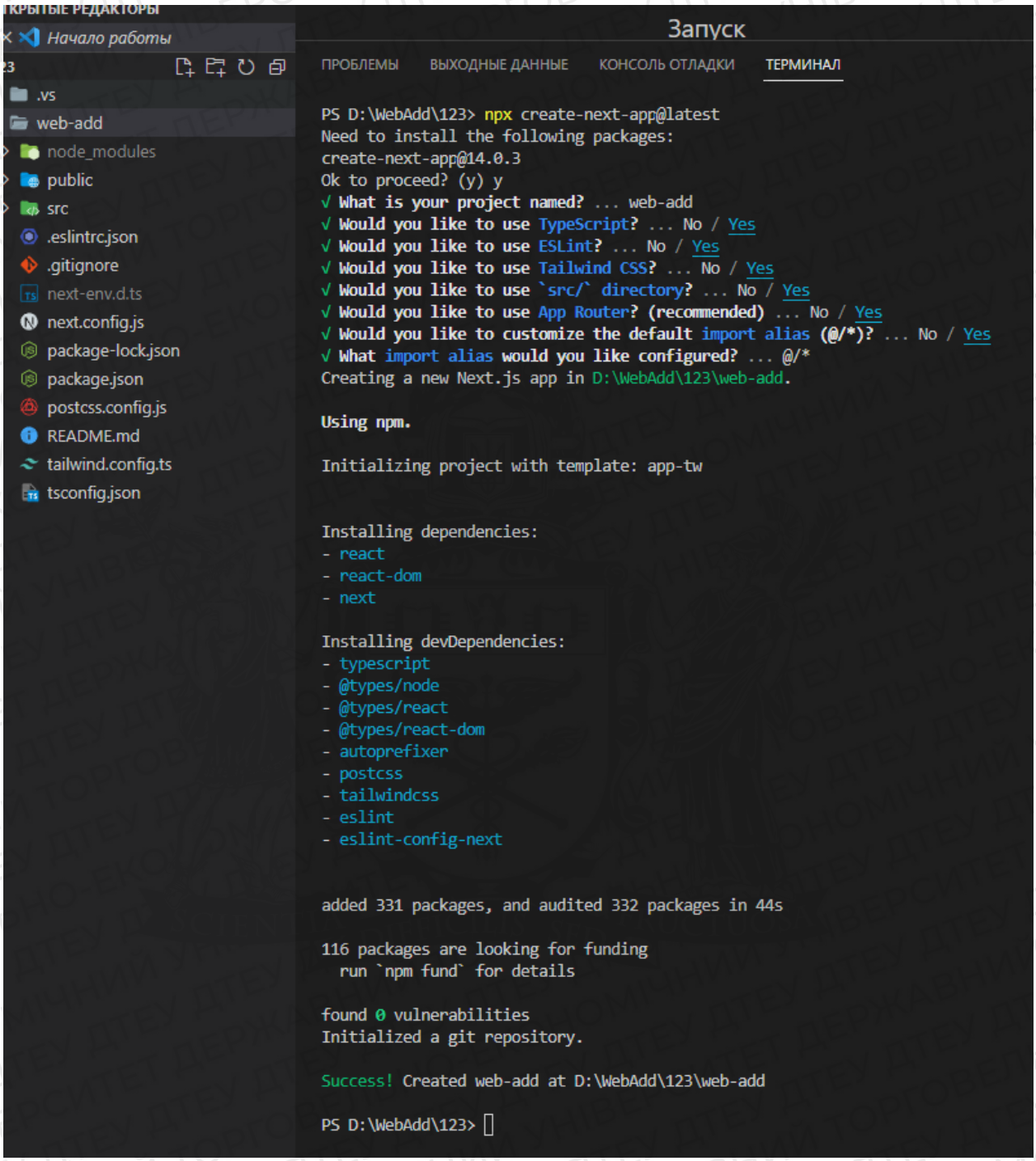

Рисунок 3.2 – Отримана конфігурація

Після чого, запускаємо проект за допомогою команди npm run dev, яка запустить локальний сервер localhost:3000. На якому потрібно видалити все з файлу page.tsx та все з файлу global.css, окрім директив Tailwind. Тепер, коли ми в середині цього файлу, додаємо кілька стилів.

```css
html,
body,
:root {
  height: 100%;
  background-color: black;
  color-scheme: dark;
}
```

Рисунок 3.3 – Зміна стилю сторінки

У файлі layout.tsx, можна змінити шифр, назву та опис нашого додатку, змінивши колір шрифту у page.tsx, ми перевіримо що Tailwind успішно налаштовано. Закінчивши налаштування навколишнього середовища, починаєм працювати з макетом, бічною панеллю та іншими елементами. Створимо компонент бічної панелі, який вміщує основний зміст. Цей компонент має бути динамічним, тому нам необхідно передавати серверні компоненти всередині клієнтських компонентів. Створюємо ще декілька компонентів, які будуть взаємодіяти SidebarItem.tsx і Box.tsx.

```tsx
        <Box>
          <div className="flex flex-col gap-y-4 px-5 py-4">
            {routes.map((item) => (
              <SidebarItem key={item.label} {...item} />
            ))}
          </div>
        </Box>
        <Box className="overflow-y-auto h-full">
          <Library songs={songs} />
        </Box>
      </div>
      <main className="h-full flex-1 overflow-y-auto py-2">
        {children}
      </main>
    </div>
  );
}
```

Рисунок 3.4 – Налаштовуємо Main content

Рисунок 3.5 – Вигляд додатку

Наступним етапом, буде створення власної бібліотеки пісень, додавання іконки та плейлисту. Налаштуємо розмір іконки та додаємо до неї назву Your Library. Далі приписання функціоналу, так як в додатку буде функція платної підписки, що дасть змогу додавати пісні в свою бібліотеку.

```
interface LibraryProps {
  songs: Song[];
}

const Library: React.FC<LibraryProps> = ({
  songs
}) => {
  const { user, subscription } = useUser();
  const uploadModal = useUploadModal();
  const authModal = useAuthModal();
  const subscribeModal = useSubscribeModal();

  const onPlay = useOnPlay(songs);

  const onClick = () => {
    if (!user) {
      return authModal.onOpen();
    }

    if (!subscription) {
      return subscribeModal.onOpen();
    }

    return uploadModal.onOpen();
  }

  return (
    <div className="flex flex-col">
      <div className="flex items-center justify-between px-5 pt-4">
        <div className="inline-flex items-center gap-x-2">
          <TbPlaylist className="text-neutral-400" size={26} />
          <p className="text-neutral-400 font-medium text-md">
            Your Library
          </p>
        </div>
```

Рисунок 3.6 – Надання кнопкам функціоналу

Налаштовуємо Header, стилізуємо його, додаємо компонент Button.tsx та імпортуємо його. Створюємо кнопки вперед, назад для переключання композицій, та в правій частині робимо кнопки профіля та входу.

```tsx
export interface ButtonProps
  extends React.ButtonHTMLAttributes<HTMLButtonElement> {}

const Button = forwardRef<HTMLButtonElement, ButtonProps>(({
  className,
  children,
  disabled,
  type = 'button',
  ...props
}, ref) => {
  return (
    <button
      type={type}
      className={twMerge(
        `
        w-full
        rounded-full
        bg-green-500
        border
        border-transparent
        px-3
        py-3
        disabled:cursor-not-allowed
        disabled:opacity-50
        text-black
        font-bold
        hover:opacity-75
        transition
      `,
        disabled && 'opacity-75 cursor-not-allowed',
        className
      )}
      disabled={disabled}
      ref={ref}
      {...props}
    >
      {children}
    </button>
```

Рисунок 3.7 – Button.tsx

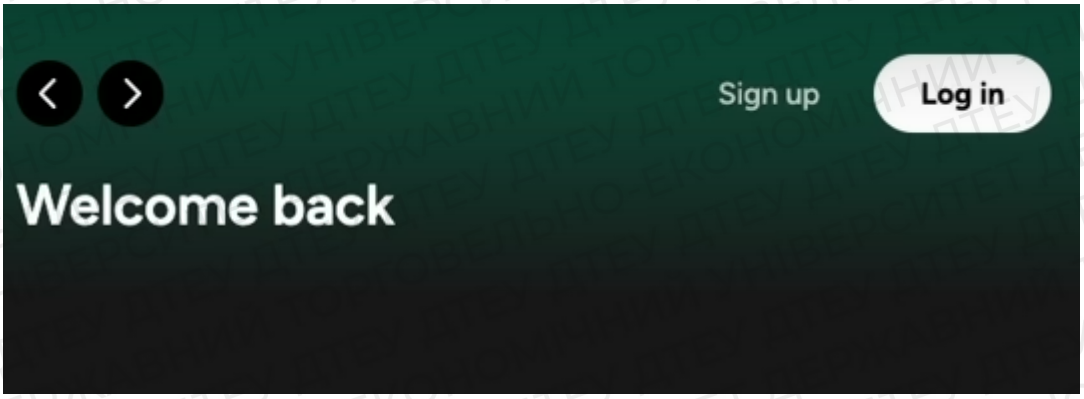

Рисунок 3.8 – Main content після доопрацювання

Розділимо компоненти на 2 групи, для зручності, в одній будуть знаходитись звичайні текстові правки, в іншій відповідати функціональним можливостям додатку. Створимо файл page.tsx, в якому будемо прописувати комунікацію, до прикладу Welcome back як на рисунку 3.8 згодом ця комунікація розшириться. В іншій папці створимо компонент Listitem.tsx і імпортуємо його з папки page.tsx та почнемо додавати інтерфейс, так як це інтерактивний компонент, маркуємо його як «use client». В середині нього буде знаходитись ImageElement. В папці public створимо нову папку images в яку завантажимо картинку, яка буде відповідати за вподобані пісні. В page.tsx ставимо шлях до цієї картинки, а в Listitem.tsx додаємо декілька атрибутів.

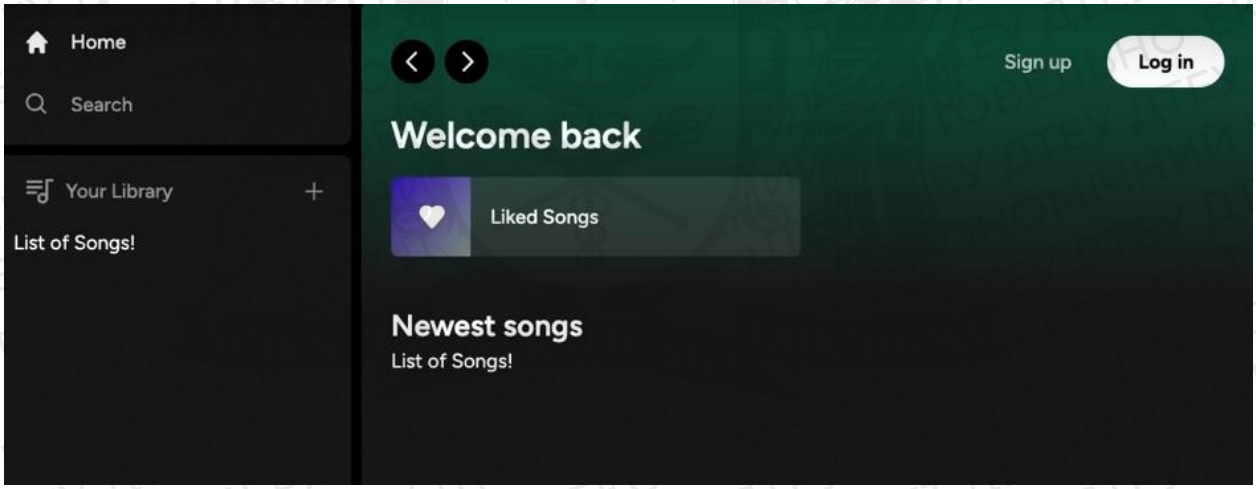

Рисунок 3.9 – Кінцевий результат

3.1.1 Створення та налаштування бази даних

Для початку необхідно зареєструватись на Supabase та створити новий проект і дати йому назву.

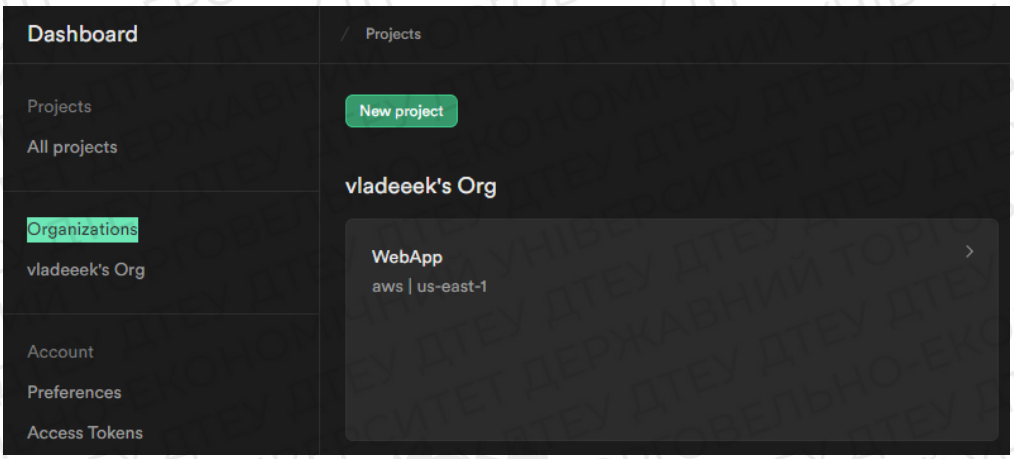

Рисунок 3.10 – Supabase Organizations

Створюємо новий файл .env.local це файл нашого середовища, який одразу буде в gitignore щоб не було змоги заштовхнути його до репозиторію або закоментувати. В середині цього файлу нам необхідно декілька API атрибутів.

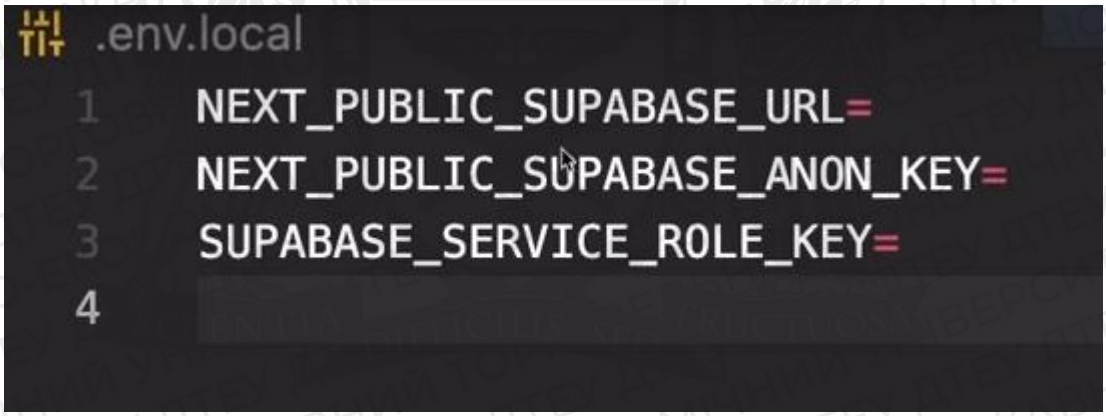

Рисунок 3.11 – Supabase Details from your project settings

Їх необхідно заповнити даними, які знаходяться в API Settings. Далі перейдемо в SQL Editor там є функція Quick Start, в якій нам необхідно обрати Stripe Subscriptions, що є стартовим шаблоном Next.js, в якому знаходиться скрипт, який ми використаємо в нашій базі даних, щоб створити таблиці, відносини, тригери і функції. Після запуску скрипту переходимо в Table editor, де будуть знаходитись створені таблиці.

Рисунок 3.12 – Готові таблиці

Але нам необхідно ще декілька таблиць для проекту, тому створюємо таблицю songs, заповнюємо її та вибираємо шаблон для надання контролю доступу.

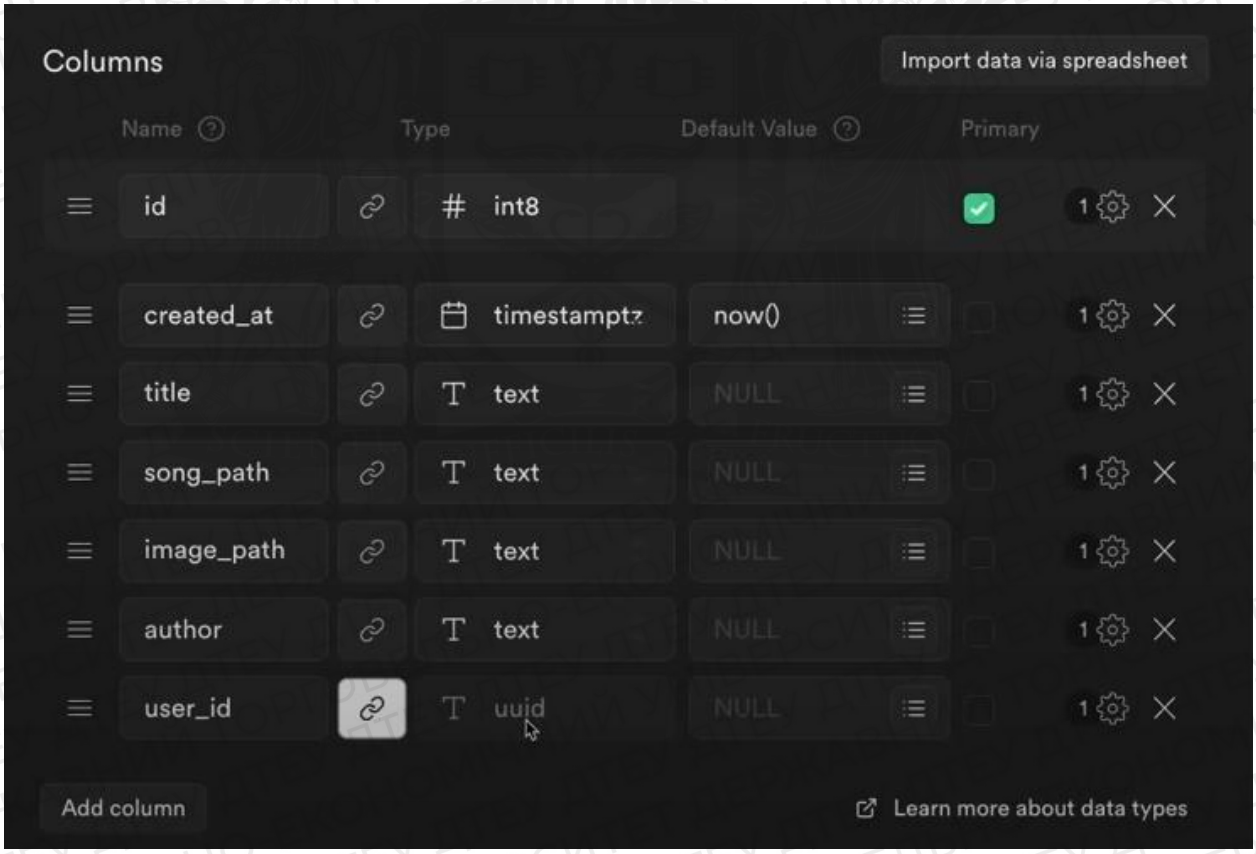

Рисунок 3.13 – Заповнення таблиці songs

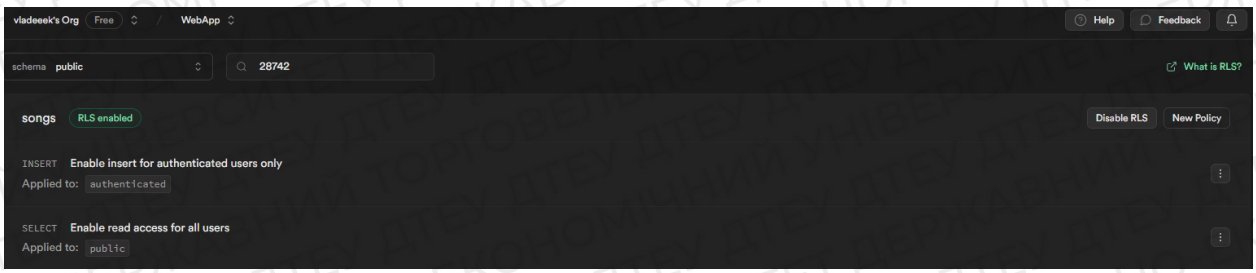

Рисунок 3.14 – Активні функції у системах управління базами даних

Створюємо ще одну таблицю, liked_songs та повторюємо всі кроки. Переходимо в Storage, де необхідно створити New bucket, songs та images, зробити їх public та обрати для них кілька шаблонів.

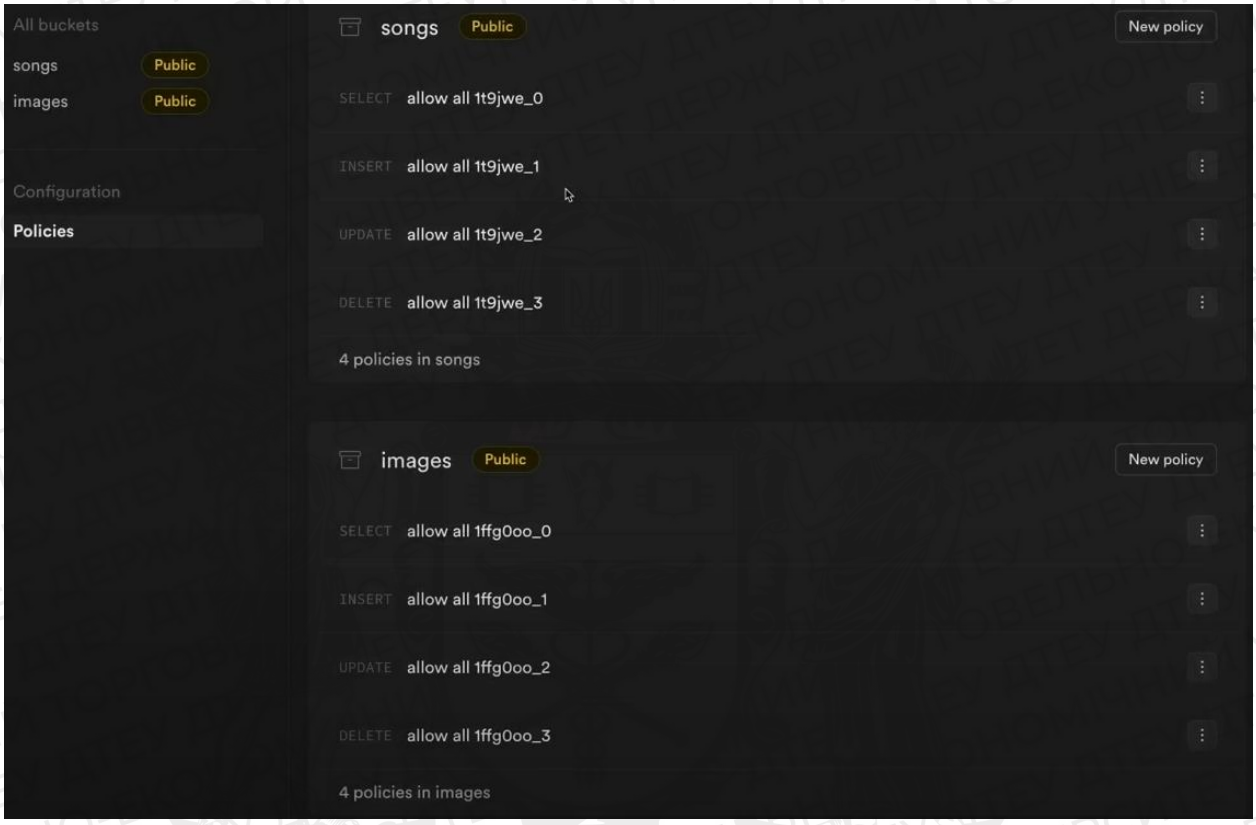

Рисунок 3.14 – Storage policies

Наступним етапом буде використання скрипту, який згенерує всі типи з бази даних і призначить в проект. Його можна знайти в гайді на Supabase documentation і необхідно ввести в терміналі.

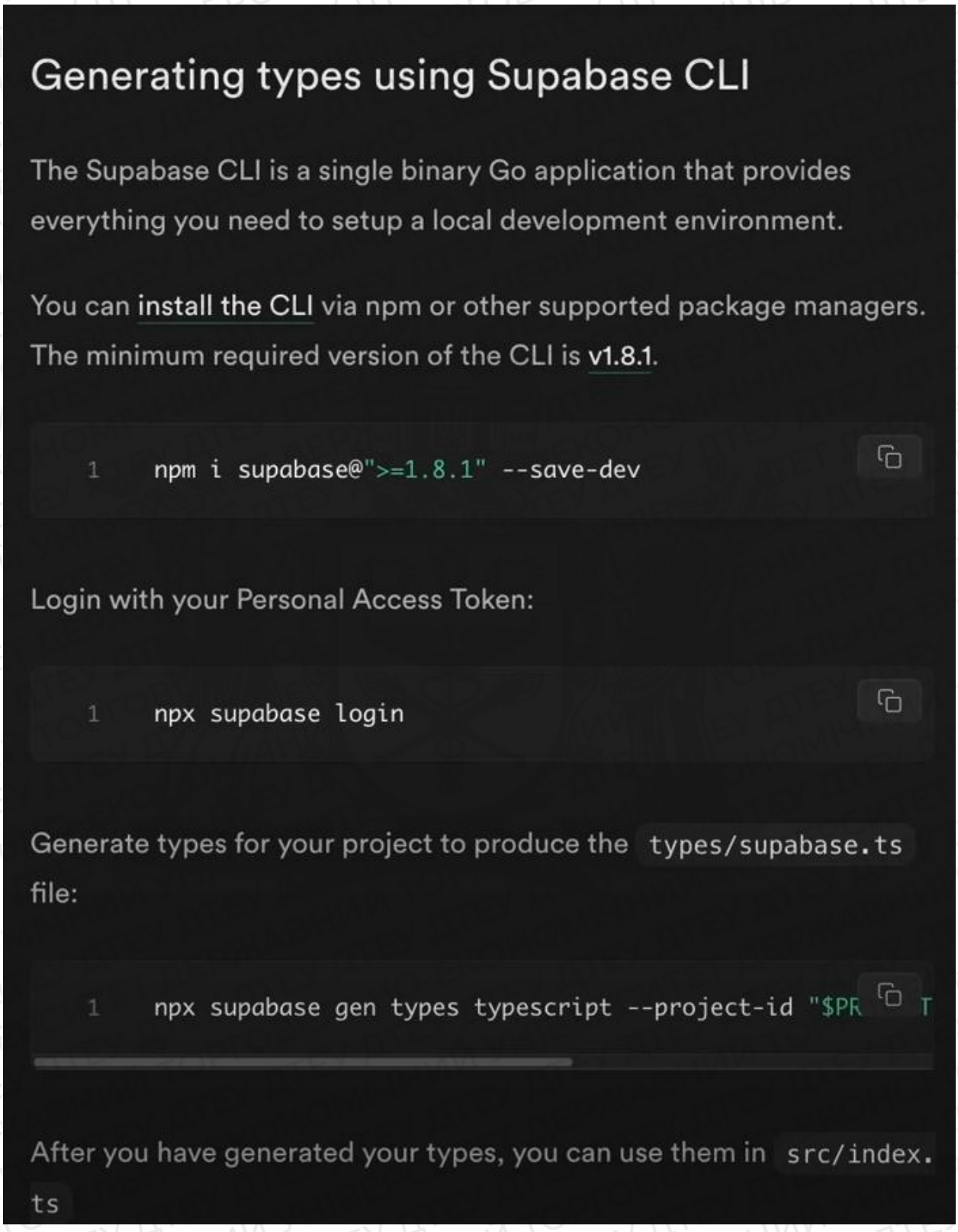

Рисунок 3.15 – Supabase CLI

Він згенерує адресу, в якій ми отримаємо токен доступу до логіну на Supabase, ввівши який ми отримаємо project-id, котрий вставляємо в наступну команду і отримуємо новий файл types_db.ts, який являється нашою базою

даних. Базу даних необхідно імпортувати в проект, тому створюємо папку providers, в якій робимо новий файл SupabaseProvider.tsx, де прописуємо властивості компоненту і встановлюємо відповідні пакети для react.

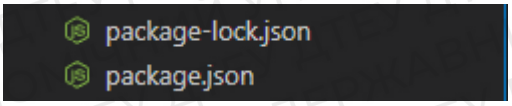

Рисунок 3.16 – Supabase helpers

Для зберігання статусу підписки на сервіс, необхідно створити нову папку hooks, в середині якої новий файл useUser.tsx, та кастомний файл з правами який ми створювали до того types_db, в ньому прописуємо команду npm install stripe, для змоги використання типів зі stripe, імпортуємо його та створюємо різноманітні деталі.

Рисунок 3.17 – Створення інтерфейсів

3.1.2 Розробка моделі автентифікації

В папці providers створюємо файл ModalProviders.tsx, маркеруємо його як “use client” та наповнюємо його функціями.

Рисунок 3.18 – Modal provider props

Повернувшись в папку components, необхідно створити файл Modal.tsx і встановити react dialog через необхідний функціонал, тож запускаємо термінал і прописуємо команду npm install @radix-ui/react dialog, та імпортуємо його.

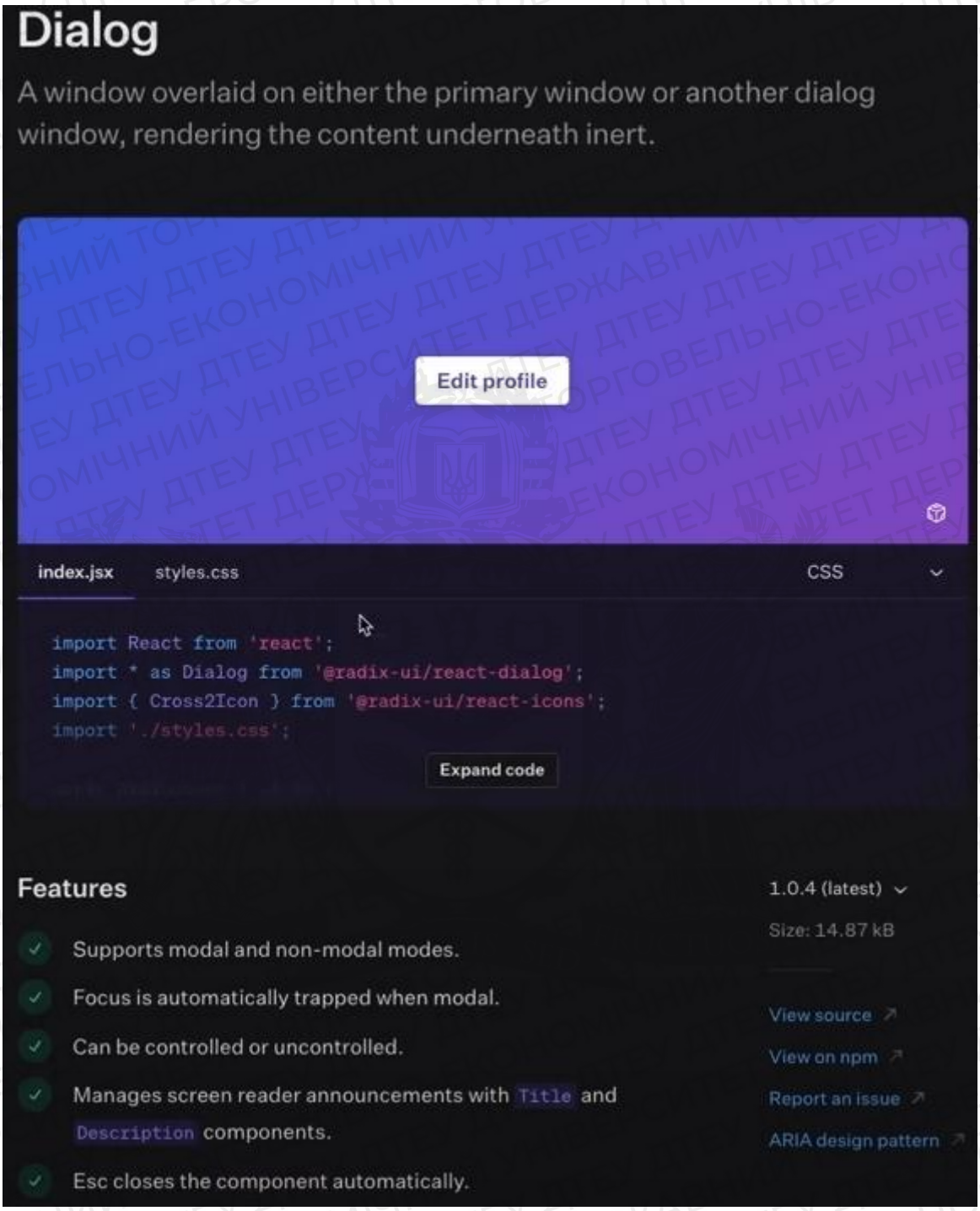

Рисунок 3.19 – Radix dialog features

Після налаштування, та доповнення компонент має ось такий вигляд:

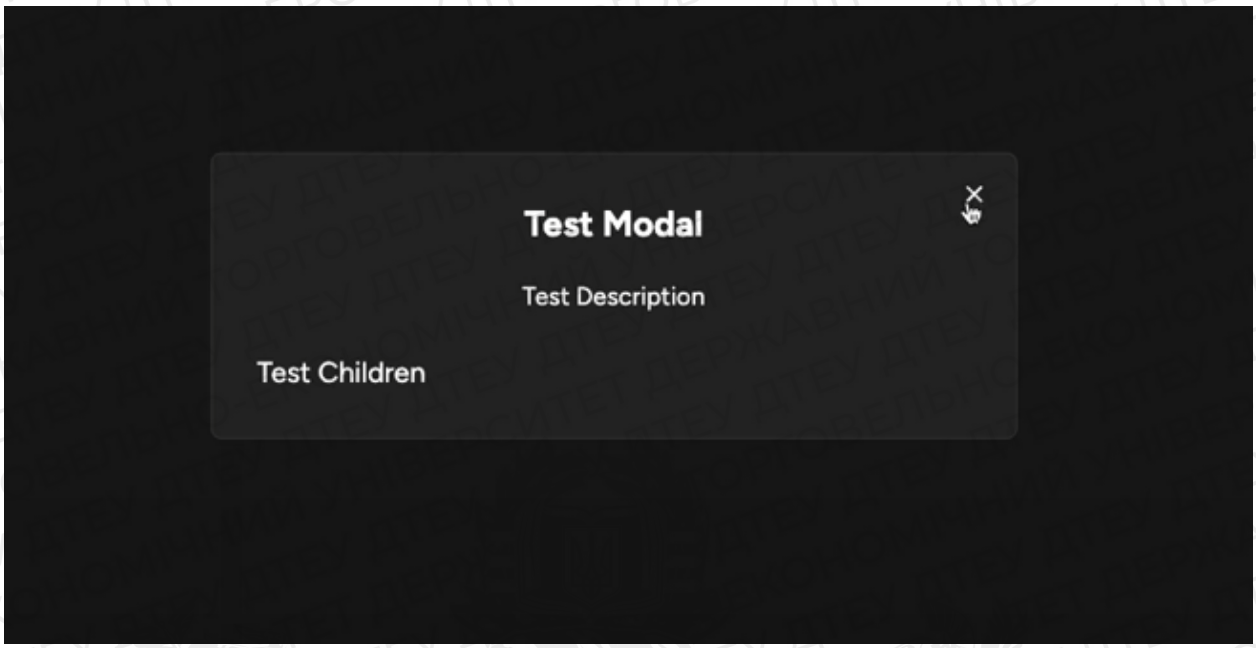

Рисунок 3.20 – Modal Component

Для того щоб викликати цю модель, необхідно в папці hooks, створити новий файл useAuthModal.ts і прописати команду npm install zustand з я кого будуть імпортовані інтерфейси, та створити в папці components файл AuthModal.tsx в якому буде використовуватись цей компонент. Наступним кроком буде встановлення необхідних пакетів npm install @supabase/auth-ui-react та @supabase/auth-ui-shared, що дасть нам змогу рендерити та надати нові характеристики моделі. Необхідно також прописати ефект, який буде не тільки відкривати вікно Log-in після натискання на кнопку, а також, який буде закривати його після успішного входу. Для початку необхідно зареєструватись, тому на пошту має приходити відповідне повідомлення, після чого, перейшовши за посиланням в базі даних вже буде міститись інформація про користувача в таблиці user, а самий додаток буде показувати, що користувач зайшов.

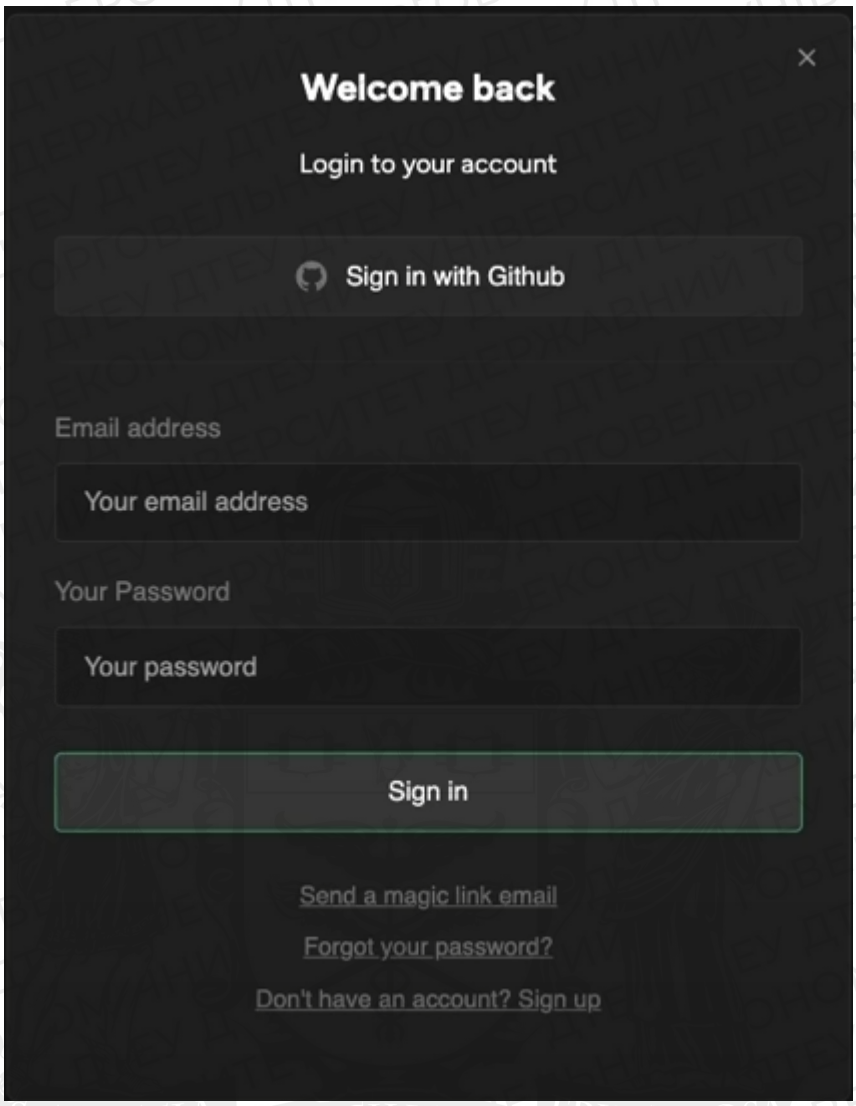

Рисунок 3.21 – Вікно після налаштування

3.1.3 Функція додавання пісень до бібліотеки

Необхідно створити контролер для того, щоб контролювати модель завантаження. В папці hooks копіюємо useAuthModal.ts, вставляємо і змінюємо назву на useUploadModal.ts, в середині файлу змінюємо екземпляри інтерфейсу на UploadModal, а також додаємо цей гачок в файл Library.tsx папки components. Надаємо цей функціонал тільки користувачам платної підписки, що перевірятиметься пізніше з інтеграцією Stripe. Для того щоб функція виконувалась , необхідно створити цей компонент, тому в папці components створюємо файл UploadModal.tsx та імпортуємо його в

ModalProvider.tsx, змінюємо заголовок та прописуємо функціонал та функцію реєстрації. Прописуємо в терміналі npm install react-hook-form та npm install unickid.

```tsx
import { FieldValues, SubmitHandler, useForm } from "react-hook-form";
import { useState } from "react";

import useUploadModal from "@/hooks/useUploadModal";

import Modal from "./Modal";

const UploadModal = () => {
  const [isLoading, setIsLoading] = useState();
  const uploadModal = useUploadModal();

  const {
    register,
    handleSubmit,
    reset
  } = useForm<FieldValues>({
    defaultValues: {
      author: '',
      title: '',
      song: null,
      image: null,
    }
  })
```

Рисунок 3.22 – ModalProvider.tsx

Створюємо новий файл в папці components під назвою Input.tsx та виправляємо помилки, які в нас будуть через імпортування.

Рисунок 3.22 – Виправлення помилок

В кінці налаштування всіх функцій та вказання всіх шляхів, майбутні пісні, які будуть додаватись, знаходитимуться в базі даних в папці songs, а кнопка викликатиме наступне вікно і матиме такий функціонал :

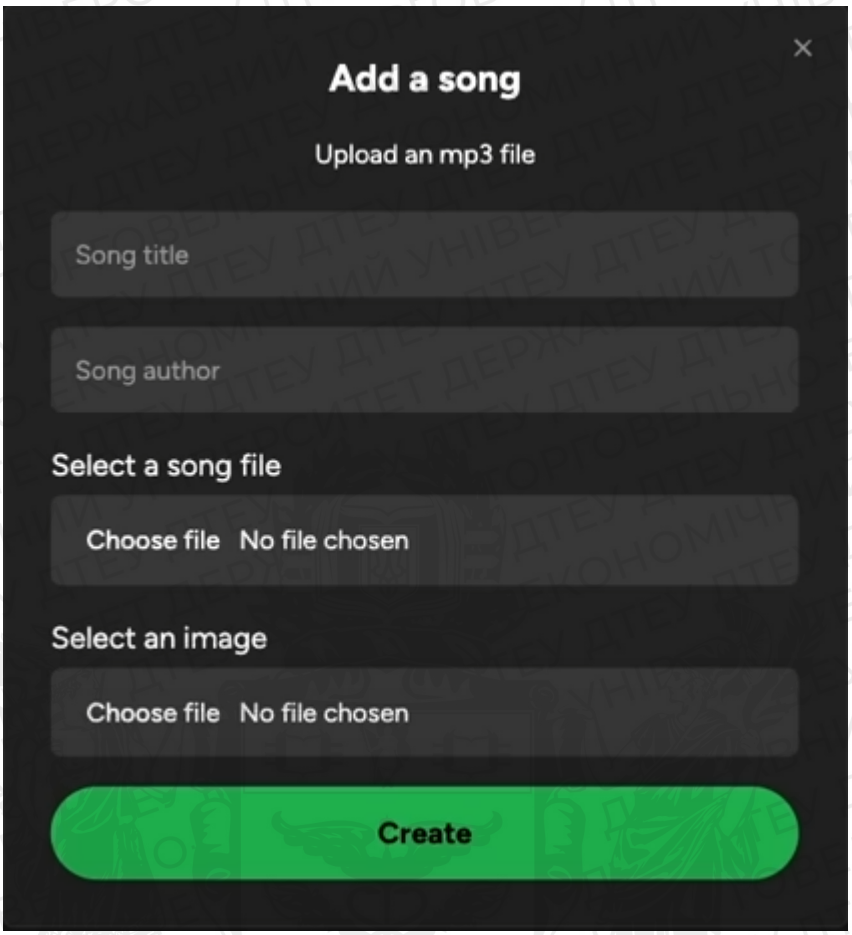

Рисунок 3.23 – Додавання пісень

3.1.4 Отримання пісень та відображення списку

Для цього створюємо папку actions в корені додатку в середині якої створюємо файл getSongs.ts де прописуємо функцію, в яку будемо імпортувати файли, тому в types.ts створимо інтерфейс Song з якого це буде відбуватись.

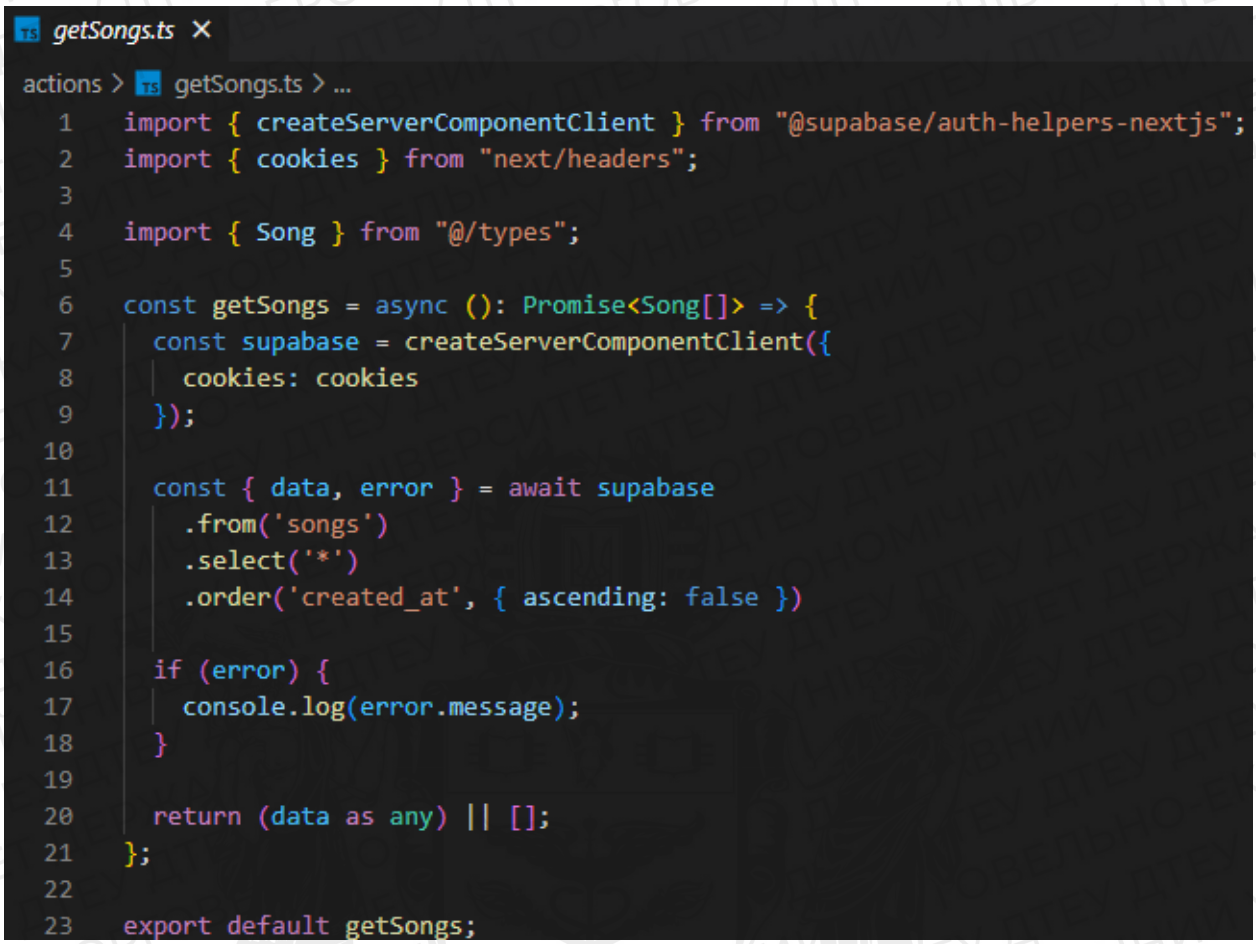

Рисунок 3.24 – Створення функції

В папці app переходимо до папки (site) і в розділі components створюємо файл PageContent.tsx виправляємо компоненти і імпортуємо їх в page.tsx. Тепер ми можемо безпечно розробляти інтерфейс який буде включати songs. У components створюємо новий файл SongItem.tsx для того, щоб ми могли додавати декілька пісень та загружати картинки. Тому в папці hooks створюємо файл useLoadImage.ts та імпортуємо туди supabaseClient. Після чого продовжуємо розробляти шлях в SongItem.tsx та налаштовувати його. Для запуску пісні нам необхідно створити новий файл в папці components під назвою PlayButton.tsx, який ми одразу імпортуємо в SongItem.tsx та починаємо його стилізувати.

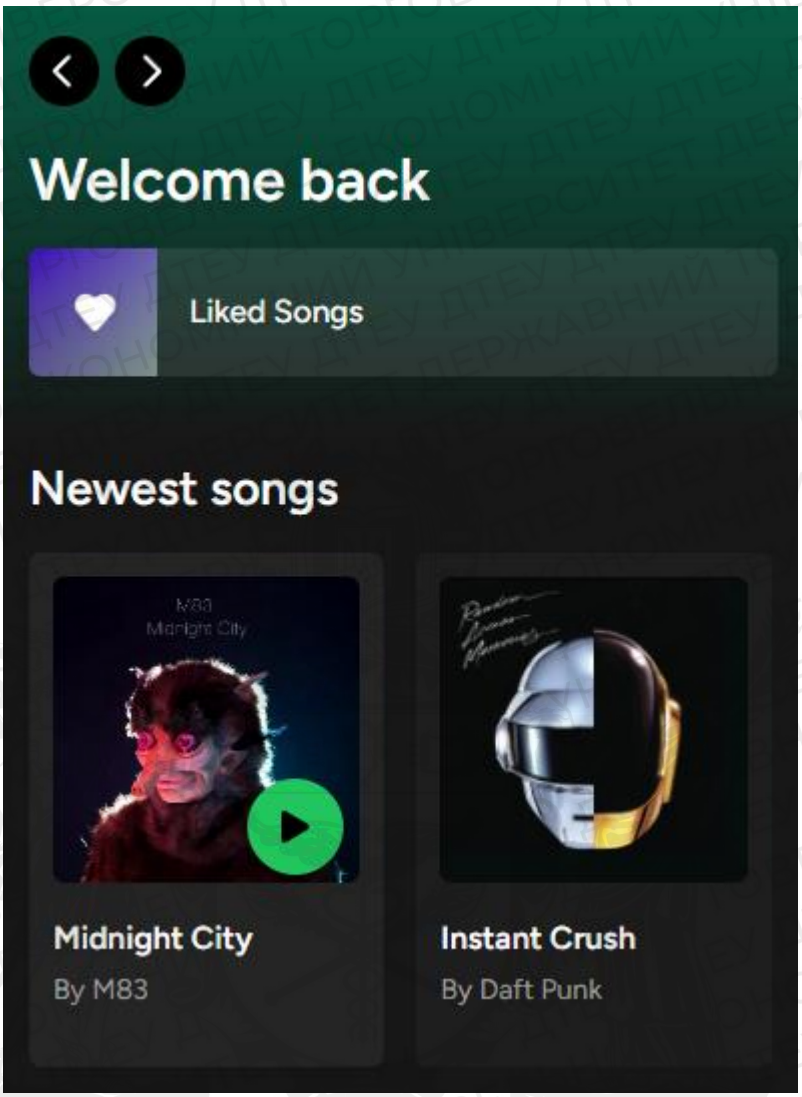

Рисунок 3.25 – Ефект PlayButton

Наступним кроком буде створення сторінки пошуку. В папці app створюємо папку search і в ній файл page.tsx, для якого одразу прописуємо інтерфейс експортуємо його та починаємо стилізувати.

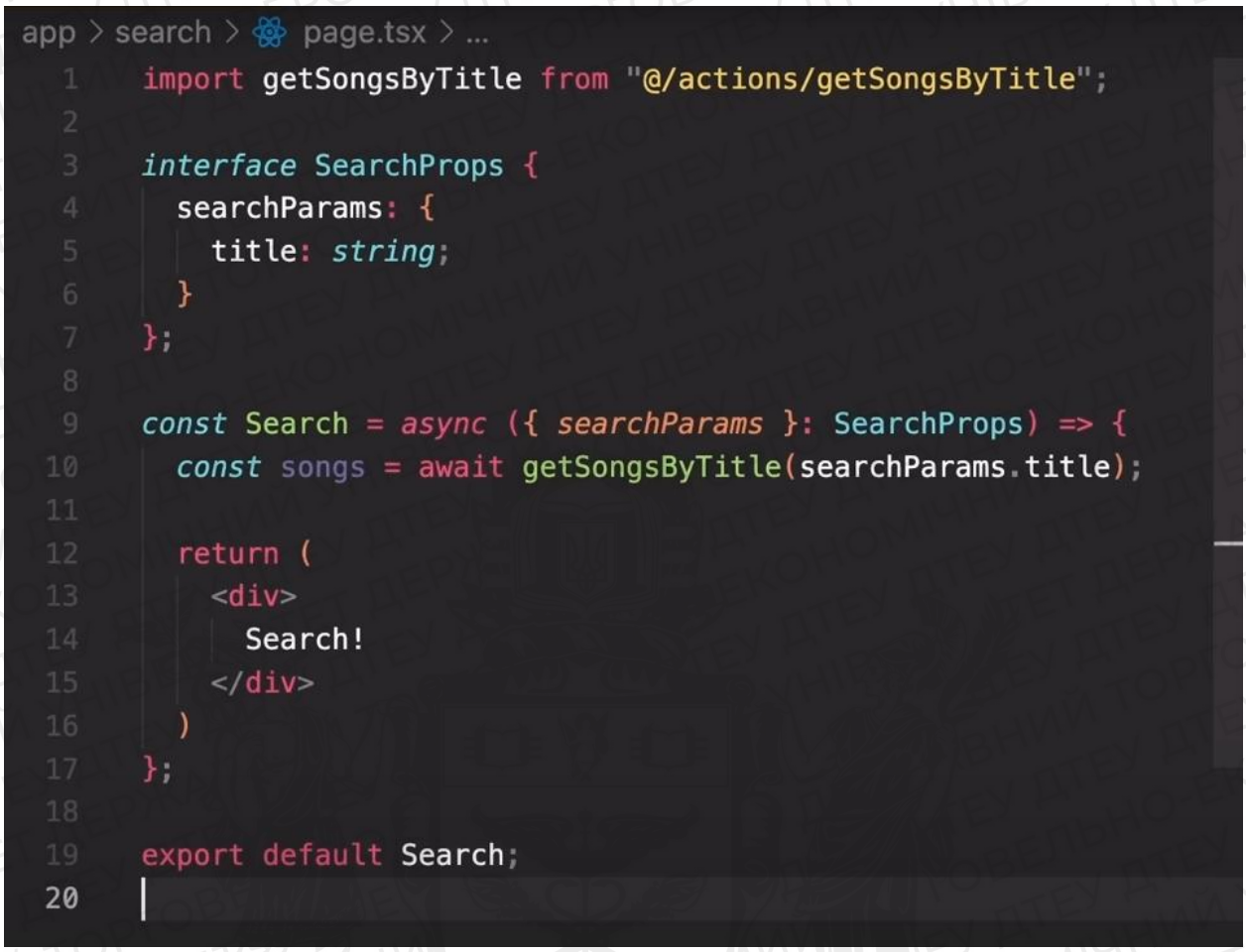

Рисунок 3.26 – Search props

Створимо новий файл SearchInput.tsx та додаємо зв’язки та ефекти, в папці hooks створюємо новий файл useDebounce.ts, в якому напишемо функцію з затримкою, яка буде вибивати наші значення коли ми будемо друкувати в графі пошуку. Надаємо функціонал SearchInput.tsx інсталюємо query-string імпортуємо його і створюємо ефект запиту. Тепер для того щоб бачити результат в пошуку повертаємось в page.tsx і додаємо SearchContent songs в якому будемо отримувати songs з getSongsByTitle, щоб уникнути помилки створюємо папку components в папці search та новий файл SearchContent.tsx який імпортуємо з page.tsx.

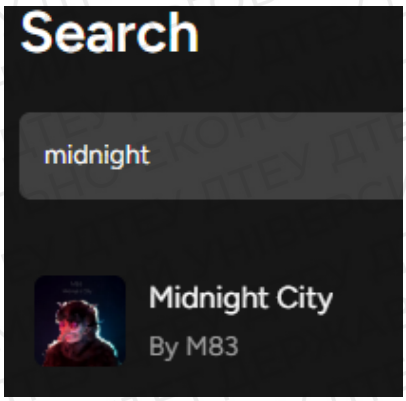

Рисунок 3.27 – Кінцевий результат

3.1.5 Функціонал «Favorites»

Необхідно створити функціонал вподобаних пісень, для якого вже була створена кнопка Liked Songs, що дасть змогу додавати пісні в окремий плейлист. Для того щоб зробити цей функціонал, необхідна кнопка LikeButton, тому створимо новий файл в папці components під назвою LikeButton.tsx та імпортуємо її з SearchContent.tsx, щоб можна було робити це з пошуку. Надаємо їй інтерфейс та імпортуємо всі дані.

Рисунок 3.28 – Процес створення LikeButton

В базі даних є таблиця liked_songs, де ми будемо бачити конкретного користувача та id конкретної вподобаної пісні, яку ми шукаємо. Тому створюємо функцію яка буде шукати пісню в цій таблиці через id. Далі необхідно створити дію, яка буде зберігати вибірку з вподобаних пісень в одному місці. В папці actions створюємо файл getLikedSongs.ts та робимо аналогічну функцію як у випадку із getSongs.ts, але обираємо зв’язки по id, так як в нашій базі даних songs має зв'язок до public.songs.id.

```
} = await supabase.auth.getSession();

const { data, error } = await supabase
  .from('liked_songs')
  .select('*, songs(*)')
  .eq('user_id', session?.user?.id)
  .order('created_at', { ascending: false });
```

Рисунок 3.29 – Зв'язок з базою даних

Створимо сторінку з вподобаними піснями, в папці app створюємо нову папку liked, де робимо файл page.tsx стилізуємо і додаємо навігацію.

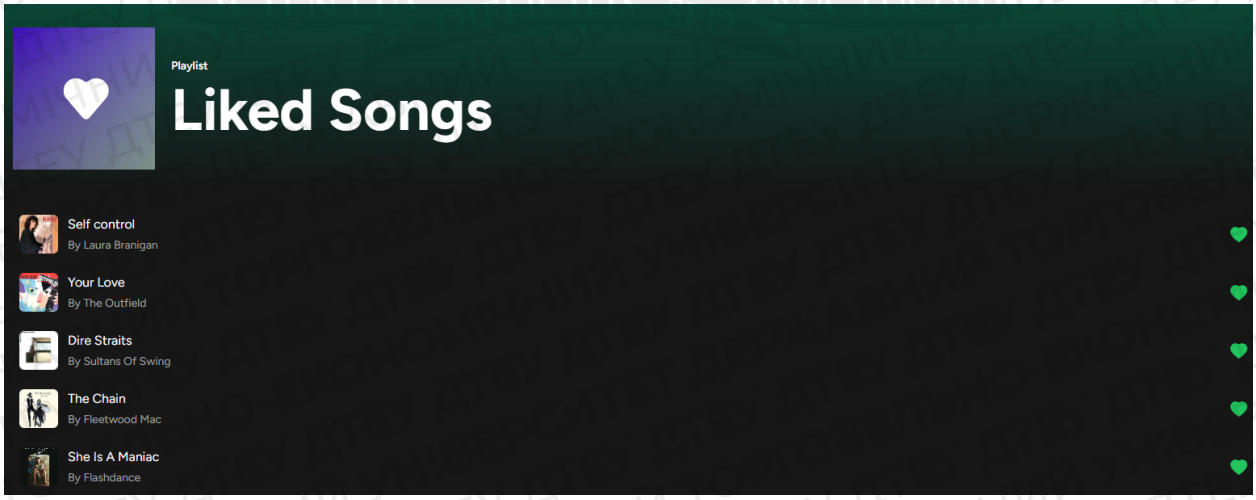

Рисунок 3.30 – Вигляд Liked Songs після доопрацювання

3.1.6 Функціональність плеєра

Один з ключових етапів створення плеєра, який буде програвати ці пісні. Він має бути динамічний і з’являтись знизу крану, кожного разу як грає пісня, та має можливість програвати музику з плейлисту під ряд. Це буде розумний плеєр який буде відслідковувати де саме ми натискаємо на програш пісні і що йому потрібно програвати. В папці hooks створюємо новий файл usePlayer.ts та створимо для нього інтерфейс. Далі в папці app файлу layout.tsx додаємо компонент Player під компонентом Sidebar, та створюємо такий компонент в папці components під назвою Player.tsx. Створимо новий файл в папці hooks під назвою useGetSongById.ts, та заповнюємо його з бази даних.

```
import { useEffect, useMemo, useState } from "react";
import { toast } from "react-hot-toast";
import { useSessionContext } from "@supabase/auth-helpers-react";

import { Song } from "@/types";

const useSongById = (id?: string) => {
  const [isLoading, setIsLoading] = useState(false);
  const [song, setSong] = useState<Song | undefined>(undefined);
  const { supabaseClient } = useSessionContext();

  useEffect(() => {
    if (!id) {
      return;
    }

    setIsLoading(true);

    const fetchSong = async () => {
      const { data, error } = await supabaseClient
        .from('songs')
        .select('*')
        .eq('id', id)
        .single();

      if (error) {
        setIsLoading(false);
        return toast.error(error.message);
      }

      setSong(data as Song);
      setIsLoading(false);
    }

    fetchSong();
  }, [id, supabaseClient]);

  return useMemo(() => ({
    isLoading,
    song
  }), [isLoading, song]);
};
```

Рисунок 3.31 – Use song by id

Після надання всього функціоналу, стилізації та додаткових функцій муту, зробимо слайдер гучності. Створимо файл Slider.tsx, нам необхідно встановити нові пакети npm install @radix-ui/react-slider та написати для нього інтерфейс.

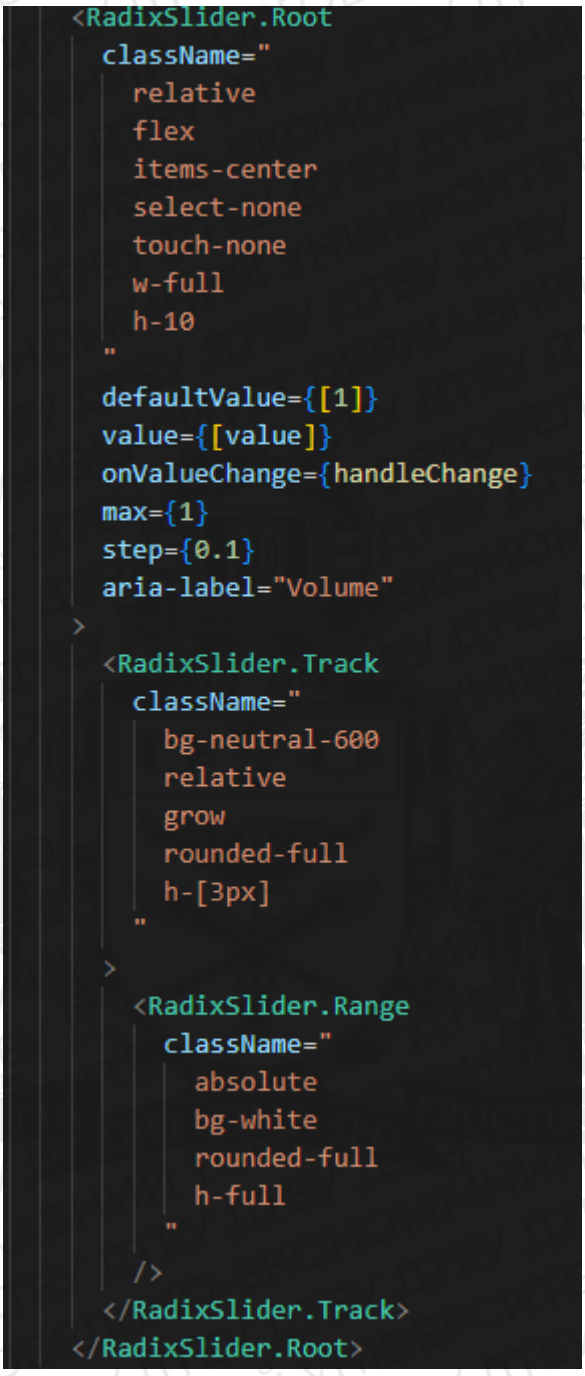

Рисунок 3.32 – Інтерфейс Slider

Після всіх налаштувань, нам необхідно встановити hook який буде використовувати sound, тому встановлюємо npm install use-sound, який необхідно налаштувати для всі файлів.

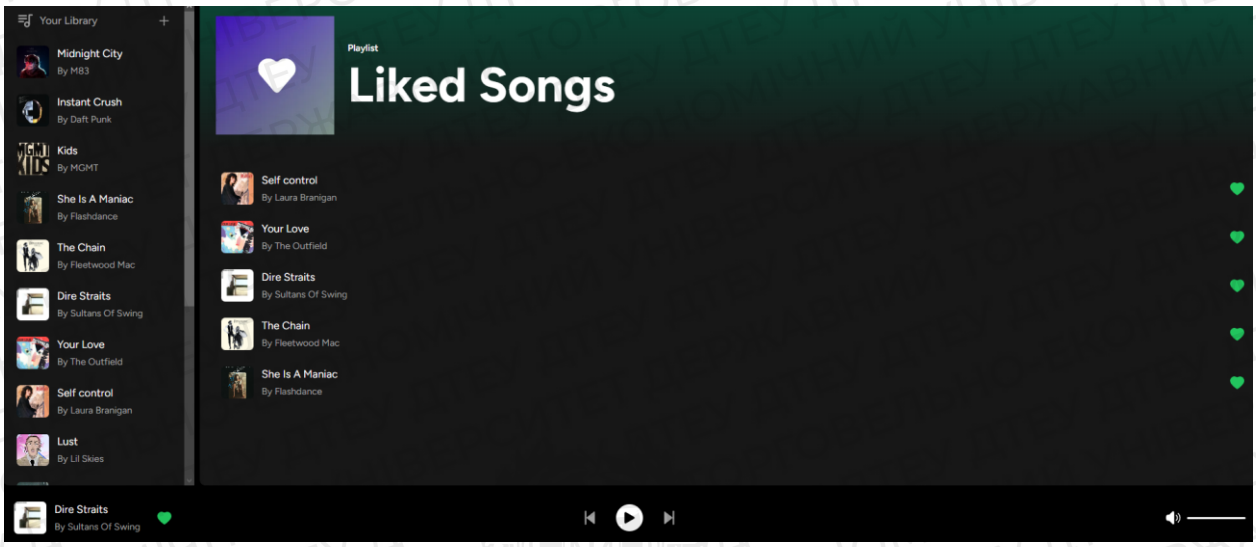

Рисунок 3.33 – Робота плеєра

3.1.7 Інтеграція Stripe

Переходимо у файл .env.local, де необхідно доповнити ще 3 API ключа. Для цього реєструємось на Stripe, внизу натискаємо на кнопку API keys for developers, копіюємо їх та вставляємо в наш файл.

Рисунок 3.34 – API keys

Далі необхідно створити кілька бібліотек для Stripe, створюємо папку libs та файл stripe.ts, в який імпортуємо Stripe. Наступний файл stripeClient.ts в який необхідно встановити нові пакети npm install @stripe/stripe-js та

імпортуємо їх в новий файл і підв’язуємо до ключів. Далі необхідно створити файл helper.ts, який допоможе потім почати створювати webhooks.

```
  let url =
    process?.env?.NEXT_PUBLIC_SITE_URL ?? // Set this to your site URL in production env.
    process?.env?.NEXT_PUBLIC_VERCEL_URL ?? // Automatically set by Vercel.
    'http://localhost:3000/';
  // Make sure to include `https://` when not localhost.
  url = url.includes('http') ? url : `https://${url}`;
  // Make sure to including trailing `/`.
  url = url.charAt(url.length - 1) === '/' ? url : `${url}/`;
  return url;
};

export const postData = async ({
  url,
  data
}: {
  url: string;
  data?: { price: Price };
}) => {
  console.log('posting,', url, data);

  const res: Response = await fetch(url, {
    method: 'POST',
    headers: new Headers({ 'Content-Type': 'application/json' }),
    credentials: 'same-origin',
    body: JSON.stringify(data)
  });

  if (!res.ok) {
    console.log('Error in postData', { url, data, res });

    throw Error(res.statusText);
  }

  return res.json();
};

export const toDateTime = (secs: number) => {
  var t = new Date('1970-01-01T00:30:00Z'); // Unix epoch start.
  t.setSeconds(secs);
  return t;
};
```

Рисунок 3.35 – helpers.ts

Після заповнення, необхідно створити ще один файл supabaseAdmin.ts, та імпортувати Stripe, createClient, Database, Price, Product.

```
export const supabaseAdmin = createClient<Database>(
  process.env.NEXT_PUBLIC_SUPABASE_URL || '',
  process.env.SUPABASE_SERVICE_ROLE_KEY || ''
);

const upsertProductRecord = async (product: Stripe.Product) => {
  const productData: Product = {
    id: product.id,
    active: product.active,
    name: product.name,
    description: product.description ?? undefined,
    image: product.images?.[0] ?? null,
    metadata: product.metadata
  };

  const { error } = await supabaseAdmin.from('products').upsert([productData]);
  if (error) throw error;
  console.log(`Product inserted/updated: ${product.id}`);
};

const upsertPriceRecord = async (price: Stripe.Price) => {
  const priceData: Price = {
    id: price.id,
    product_id: typeof price.product === 'string' ? price.product : '',
    active: price.active,
    currency: price.currency,
    description: price.nickname ?? undefined,
    type: price.type,
    unit_amount: price.unit_amount ?? undefined,
    interval: price.recurring?.interval,
    interval_count: price.recurring?.interval_count,
    trial_period_days: price.recurring?.trial_period_days,
    metadata: price.metadata
  };

  const { error } = await supabaseAdmin.from('prices').upsert([priceData]);
  if (error) throw error;
  console.log(`Price inserted/updated: ${price.id}`);
};
```

Рисунок 3.36 – Заповнення supabaseAdmin.ts

Кінцевим результатом буде створення нової API папки в якій створюємо папку webhooks в середині якої файл route.ts.

Рисунок 3.37 – route.ts

Тепер необхідно підключити Stripe до webhooks. Для цього необхідно заповнити останній ключ в папці .env.local, але щоб зробити це та протестувати, необхідно встановити CLI та увійти в Stripe акаунт.

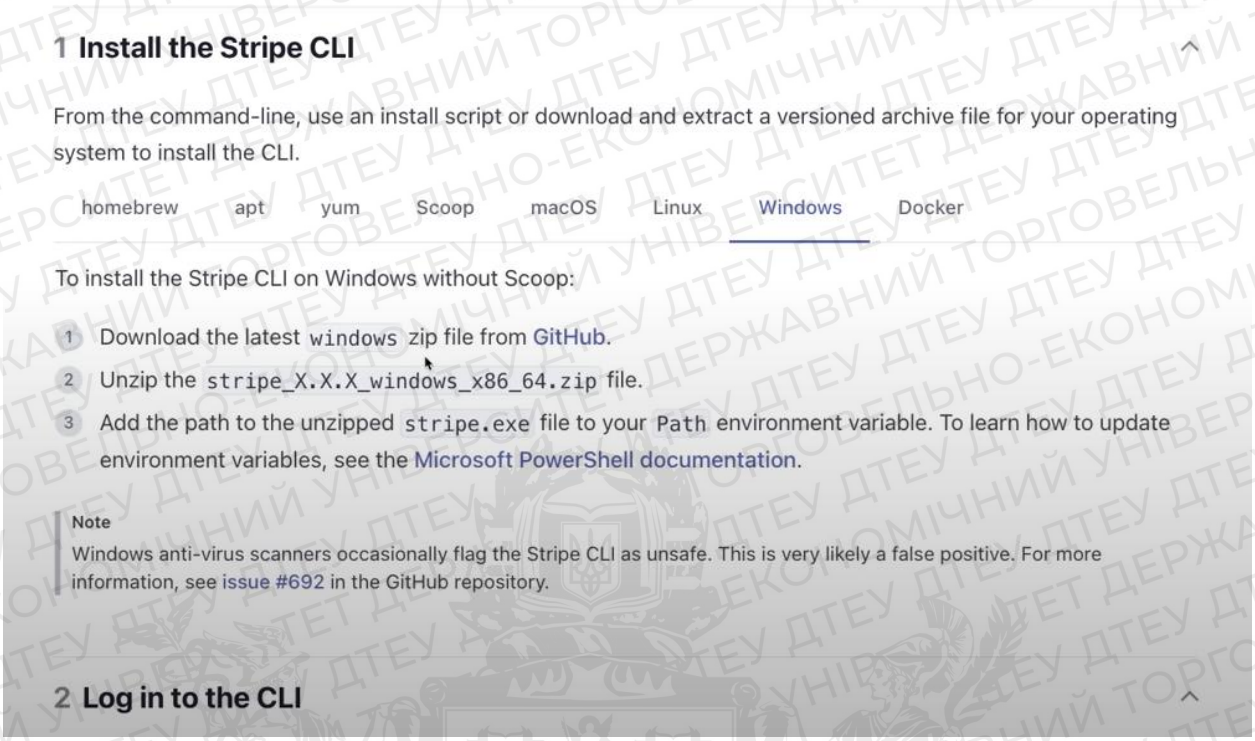

Рисунок 3.38 – Stripe CLI

Відкривши 2 термінали, в одному запускаємо проект, а в іншому вводимо команду stripe listen—forward-to localhost:3000/api/webhooks після чого ми отримаємо STRIPE_WEBHOOK_SECRET. Тепер ми можемо зробити платну підписку для нашого сервісу.

Рисунок 3.39 – Преміум підписка

Залишилось зробити сторінку для підписки за тим же принципом який вже розглядали та задеплоїти наш проект, щоб зробити його доступним для всіх.

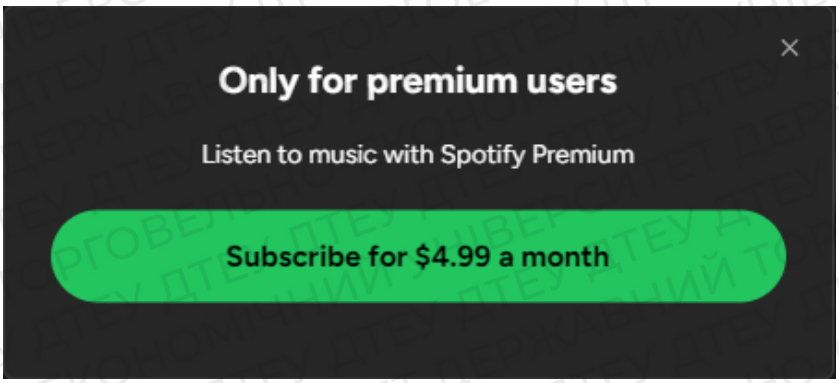

Рисунок 3.40 – Підписка

3.2. Інструкція користувача

Для початку необхідно створити новий репозиторій на GitHub. Після чого ввести наступні команди :

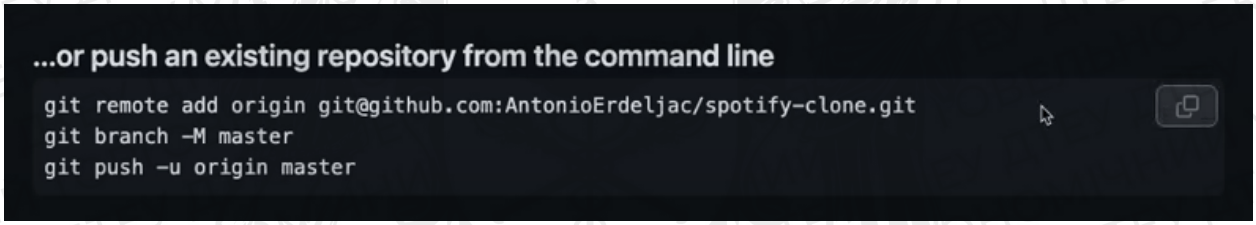

Рисунок 3.41 – Імпорт проекту

Як наслідок весь код з’явиться в створеному репозиторію, тепер треба перейти на Vercel, зайти через GitHub і натиснути “Add New Project”, з’явиться можливість імпортувати створений репозиторій, де необхідно буде замінити параметр Environment Variables, на ключі з файлу проекту .env.local і натиснути Deploy. В результаті чого, якщо не виникло помилок то в нас з’явиться готовий проект і посилання за яким ми можемо зайти на наш веб додаток.

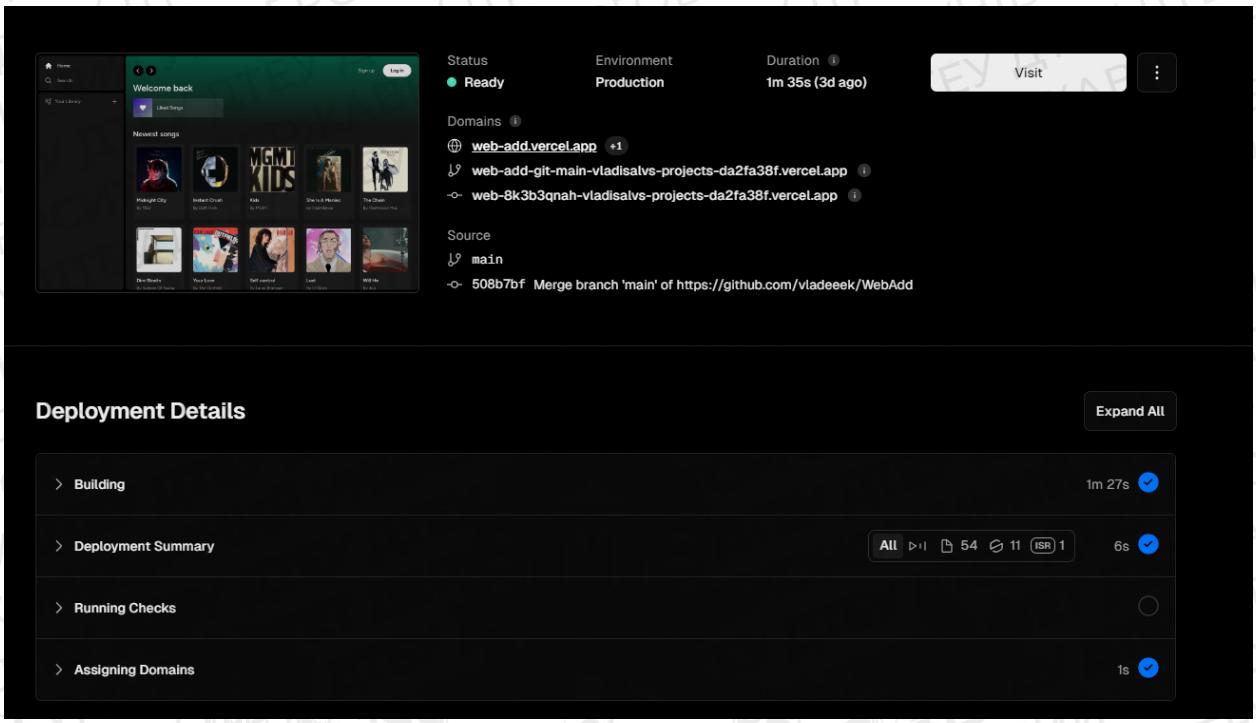

Рисунок 3.42 – Deployment project

Перейшовши за посиланням, нас зустрічає готовий проект, де залишилось перевірити його функції.

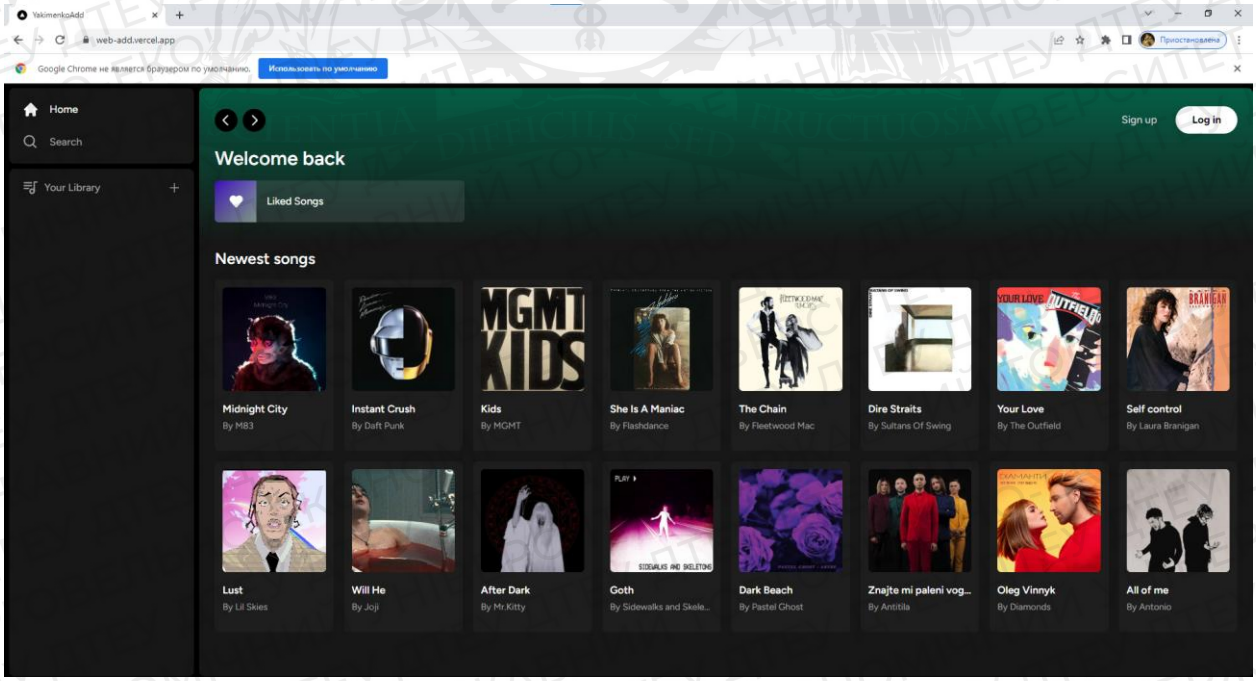

Рисунок 3.43 – Головна сторінка

Першим ділом необхідно перевірити реєстрацію, тож натискаємо відповідну кнопку і вводимо дані. Після цього перевіряємо підписку.

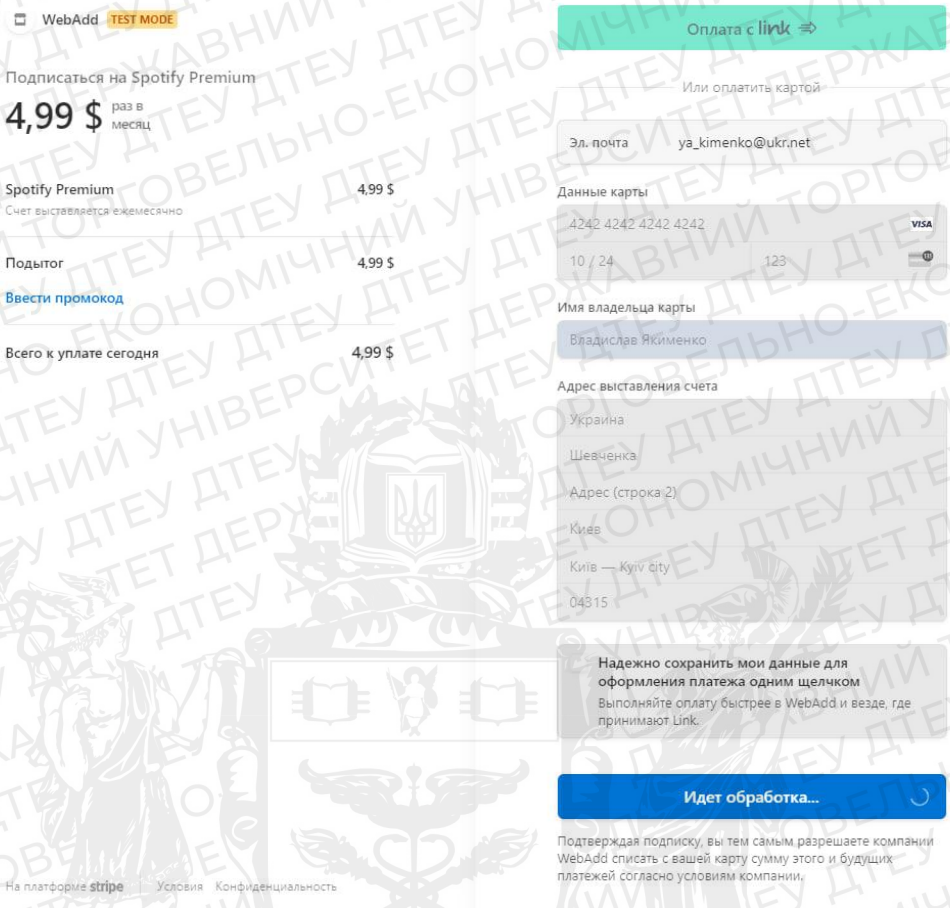

Рисунок 3.44 – Підписка на сервіс

Рисунок 3.45 – Програш пісні та робота слайдера гучності

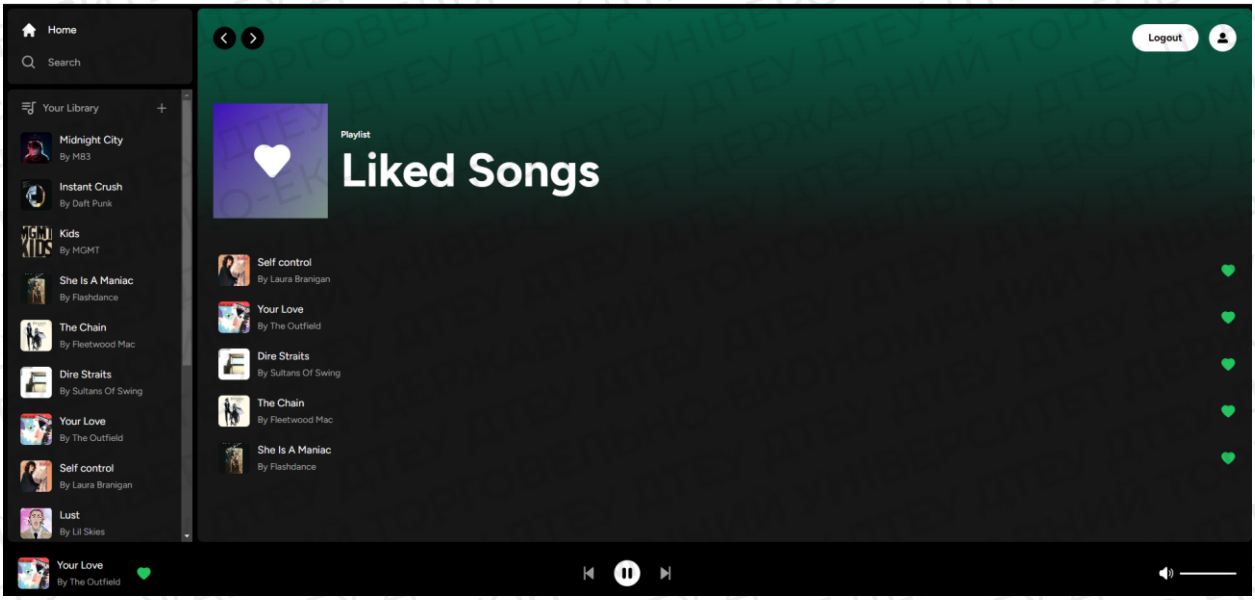

Рисунок 3.46 – Програш пісні з плейлисту вподобаних та робота перемикачів

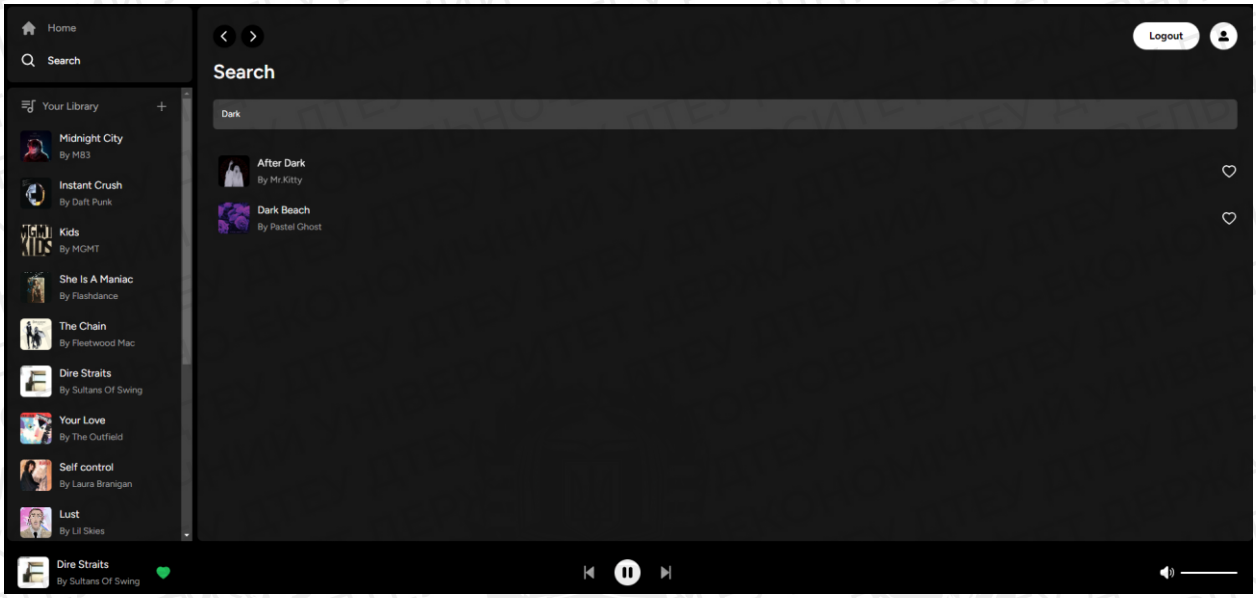

Рисунок 3.47 – Робота функції пошуку

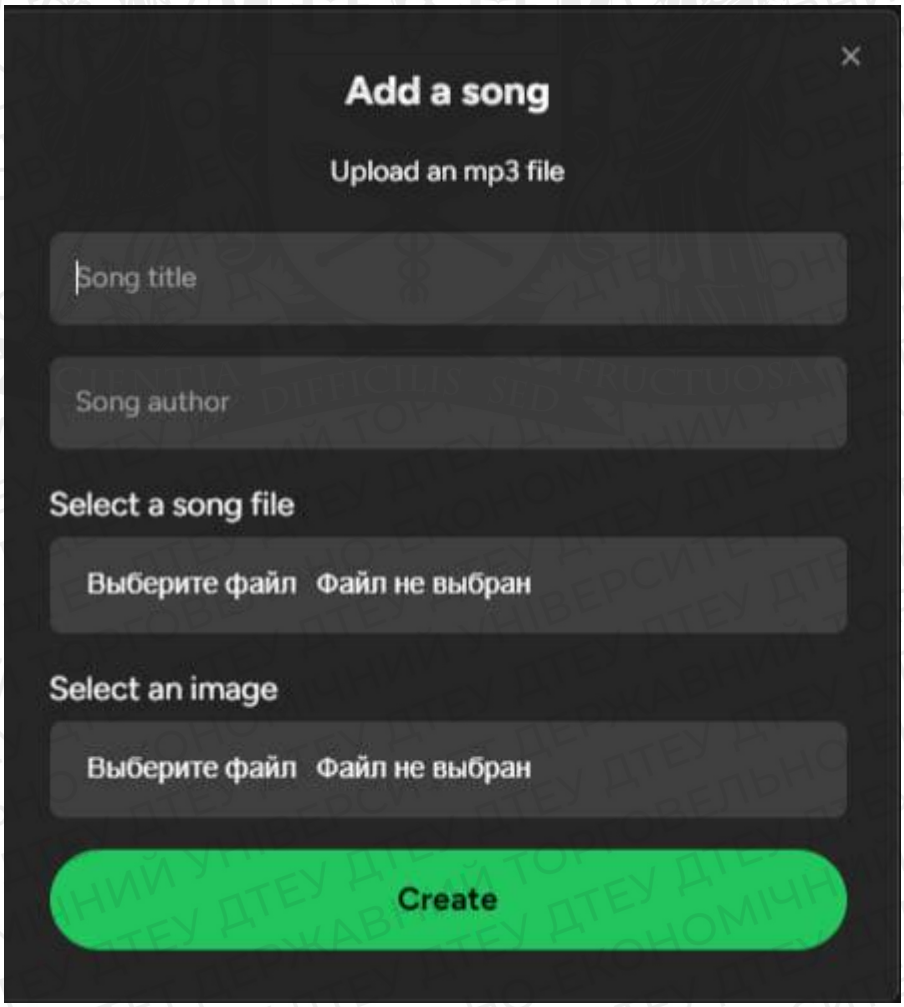

Рисунок 3.48 – Робота функції додавання пісні

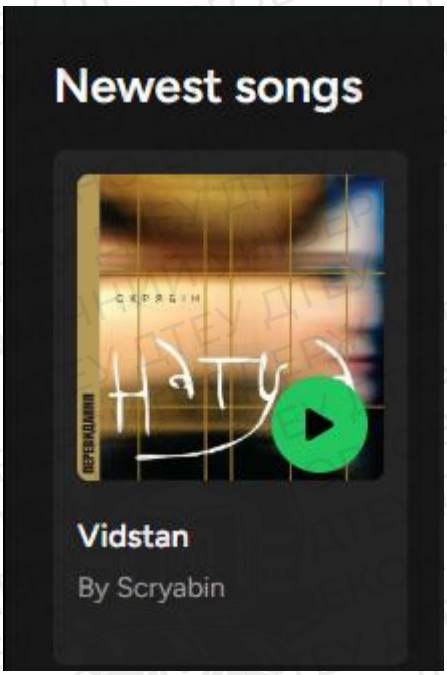

Рисунок 3.49 – Додана пісня

ВИСНОВКИ

Узагальнення результатів теоретичної та дослідно-експериментальної роботи дає підстави для таких висновків:

Аналіз теоретичних та практичних підходів розробки web-додатку для прослуховування музики дозволили визначити, що успіх такого додатку значною мірою залежить від забезпечення високоякісного користувацького досвіду, ефективного управління потоками даних та інтеграції з різноманітними музичними джерелами та сервісами.

Модель роботи web-додатку для прослуховування музики складається з декількох ключових компонентів: фронтенд-інтерфейсу, який забезпечує користувацький доступ та взаємодію; бекенд-системи, яка обробляє бізнес-логіку, управління даними та інтеграцію з музичними бібліотеками та API; системи управління базами даних для зберігання інформації про користувачів, їх налаштування та плейлисти.

Використання програмно-апаратних засобів сприяло розробці web-додатку для прослуховування музики, а саме: забезпечення швидкої та ефективної обробки аудіоданих завдяки потужним серверам та оптимізованому серверному програмному забезпеченню; використання сучасних фреймворків і бібліотек для фронтенд-розробки, що дозволяють створювати візуально привабливі та інтуїтивно зрозумілі інтерфейси; імплементування системи безпеки для захисту персональних даних користувачів та фінансових транзакцій.

Побудова моделі web-додатку для прослуховування музики та використання програмно-апаратних засобів допоможуть в подальшій

оптимізації процесу стрімінгу, забезпечуючи високу продуктивність та стабільність сервісу навіть при великому навантаженні.

Перспективами подальших досліджень є розробка покращених алгоритмів для персоналізації музичних рекомендацій, що враховують не лише історію прослуховувань користувача, але й контекстуальні фактори, такі як настрій, час доби та місцеположення. Крім того, важливим аспектом є розробка ефективніших способів захисту авторських прав у цифровому середовищі та боротьба з порушеннями у сфері стрімінгу музики.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Шип С. В. Музична форма від звуку до стилю [Текст]: навч. посіб. / С. В. Шип. — К. : Заповіт, 1998. — 368 с. — ISBN 966-7272-21-4 : Б. Ц.

2) Samson, Jim. "Genre". In Grove Music Online. Oxford Music Online. Accessed March 4, 2012.

3) Dannenberg, Roger (2010). Style in Music (PDF) (published 2009). p. 2. Bibcode:2010tsos.book...45D.

4) Green, Douglass M. (1965). Form in Tonal Music. Holt, Rinehart, and Winston, Inc. p. 1. ISBN 978-0-03-020286-5.

5) Fabbri, Franco (1982), A Theory of Musical Genres: Two Applications (PDF), p. 1

6) Schwartz, Kelly; Fouts; Gregory (2003). "Music preferences, personality style, and developmental issues of adolescents". Journal of Youth and Adolescence. 32 (3): 205–213. doi:10.1023/a:1022547520656. S2CID 41849910.

7) Bonneville-Roussy, Arielle; Rentfrow, Peter J.; Xu, Man K.; Potter, Jeff (2013). "Music through the ages: Trends in musical engagement and preferences from adolescence through middle adulthood". Journal of Personality and Social Psychology. 105 (4): 703–717. doi:10.1037/a0033770. PMID 23895269.

8) Vincenzo Caporaletti (2005). I processi improvvisativi nella musica. Lucca. LIM. ISBN 88-7096-420-5.

9) "EarMaster – Music Theory & Ear Training on PC, Mac and iPad". www.earmaster.com. Retrieved March 29, 2019.

10) "Home | International Council for Traditional Music". ictmusic.org. Retrieved March 29, 2019.

11) Майбурова К. Музичні жанри. — К., 1959;

12) Юрій Юцевич. Музика: словник-довідник. — Тернопіль, 2003. — 404 с. — ISBN 966-7924-10-6. (html-пошук по словнику, djvu)

13) Е. В. Назайкинский — Стиль і жанр в музиці — М., 2003

14) М. К. Михайлов — Стиль в музиці — М., 1981

15) Музика двадцятого століття: Навч. посіб. для вищ. навч. закл. культури і мистец. I—IV рівнів акредитації / С. Павлишин; Львів. держ. муз. акад. ім. М.Лисенка. — Л. : БаК, 2005. — 232 с. — Бібліогр.: 84 назви.

16) iTunes – Apple. Apple Inc. Архів оригіналу за 7 квітня 2018. Процитовано 3 червня 2019. «iTunes is the best way to organize and enjoy the music, movies, and TV shows you already have – and shop for the ones you want.»

17) How to use Google Play Music. Google Play Help. Google. Архів оригіналу за 2 лютого 2017. Процитовано 18 січня 2017.

18) The Music Industry. The Economist. 15 жовтня 2008. Архів оригіналу за 29 червня 2011. Процитовано 13 грудня 2011.

19) Resnikoff, Paul (12 грудня 2018). Worldwide Independent Network CEO Alison Wenham Is Stepping Down. Digital Music News. Процитовано 16 серпня 2019.

20) "Workstation Blues", стаття Roger Nichols

21) https://www.billboard.com/lists/ways-ai-has-changed-music-industry-artificial-intelligence/revolutionizing-production/

22) https://comma.com.ua/article/vr-music/

23) Economist Staff (31 жовтня 2015). Blockchains: The great chain of being sure about things. The Economist. Архів оригіналу

24) Smart Contracts, Explained. Архів оригіналу за 21 січня 2018. Процитовано 23 жовтня 2021.

25) Apple - iPad - From the App Store - Play and record music with GarageBand (2011). Дата обращения: 2 февраля 2011. Архивировано из оригинала 22 июля 2012 года.

26) Clarke, Steven Measuring API Usability. Dr. Dobb's (2004). Дата обращения: 9 июля 2021. Архивировано 3 марта 2022 года.

27) Spotify Technology S.A. FORM F-1 REGISTRATION STATEMENT UNDER THE SECURITIES ACT OF 1933 (амер.). Архів оригіналу за 29 квітня 2019. Процитовано 7 березня 2019.

28) Кількість користувачів Facebook в Україні досягла 13 мільйонів — дослідження. ms.detector.media. 14 лютого 2019. Архів оригіналу за 10 серпня 2020.

29) https://w.wiki/8CqU

30) «Apple W.W.D.C. 2015: iOS 9, Apple Pay and Other Announcements» [Архівовано 22 червня 2015 у Wayback Machine.].

31) McElhearn, Kirk (9 січня 2016). 15 years of iTunes: A look at Apple's media app and its influence on an industry. Macworld. International Data Group. Архів оригіналу за 17 грудня 2017. Процитовано 5 липня 2023.

32) Вийшла iOS 17.1. Змін дуже багато. // Автор: Скарбик Павло. 25.10.2023

33) https://www.sri.com/engage/ventures/siri

34) Morrison, Geoffrey Surround all around: Dolby Atmos explained. CNET. Дата обращения: 21 января 2020. Архивировано 21 января 2020 года.

35) "Amazon Music has more than 55 million customers worldwide". About Amazon. January 22, 2020. Retrieved February 26, 2021.

36) Christopher Levy (3 лютого 2003). Making Money with Streaming Media. streamingmedia.com. Архів оригіналу за 14 травня 2006. Процитовано 28 серпня 2006.

37) Amazon.com Help: Set Up Your Amazon Echo. Amazon.com. Дата обращения: 4 марта 2015. Архивировано 4 марта 2015 года.

38) Alexa Voice Service Overview. Дата обращения: 2 июня 2019. Архивировано 10 декабря 2019 года.

39) Історія Google. Google, Inc. Архів оригіналу за 25 травня 2013. Процитовано 17 серпня 2013.

40) SoundCloud » You can now master your tracks directly on SoundCloud. blog.soundcloud.com. Архів оригіналу за 4 серпня 2020. Процитовано 9 серпня 2020.

41) Masinter L., Connolly D. The 'text/html' Media Type (англ.) — IETF, 2000. — 8 p. — doi:10.17487/RFC2854

42) Mozilla Development Network — HTML5 [Архівовано 26 квітня 2011 у Wayback Machine.]

43) CSS (вікіпідручник) [Архівовано 27 березня 2018 у Wayback Machine.]

44) Press release announcing JavaScript, «Netscape and Sun announce Javascript», PR Newswire, December 4, 1995

45) Design Patterns: Elements of Reusable Object-Oriented Software [Архівовано 9 листопада 2012 у Wayback Machine.]

46) Releases – Facebook/React. GitHub. Архів оригіналу за 29 серпня 2023. Процитовано 14 червня 2022.

47) Angular: Branding Guidelines for AngularJS. Архів оригіналу за 9 січня 2018. Процитовано 28 січня 2018.

48) About Node.js, and why you should add Node.js to your skill set?. Training.com. Training.com. Архів оригіналу за 1 квітня 2017. Процитовано 23 жовтня 2016.

49) Richardson, Leonard; Amundsen, Mike; Ruby, Sam (2013). RESTful Web APIs (вид. First edition). O'Reilly. ISBN 978-1-4493-5806-8. Процитовано 13 вересня 2017.

50) [История MySQL (англ.). Архів оригіналу за 31 серпня 2009. Процитовано 3 червня 2010. История MySQL (англ.)]

51) Next.js Brand Guidelines. 26 серпня 2022.

52) Superbase and SIMPOL Together Again". Superbase. 28 November 2014. Retrieved 12 May 2020.

53) https://stripe.com/about

ДОДАТОК

Програмний код реалізації Web-системи

Folder actions
getActiveProductsWithPrices.ts

```
import { createServerComponentClient } from "@supabase/auth-helpers-nextjs";
import { cookies } from "next/headers";

import { ProductWithPrice } from "@/types";

const getActiveProductsWithPrices = async (): Promise<ProductWithPrice[]> => {
  const supabase = createServerComponentClient({
    cookies: cookies
  });

  const { data, error } = await supabase
    .from('products')
    .select('*, prices(*)')
    .eq('active', true)
    .eq('prices.active', true)
    .order('metadata->index')
    .order('unit_amount', { foreignTable: 'prices' });

  if (error) {
    console.log(error.message);
  }

  return (data as any) || [];
}

export default getActiveProductsWithPrices;
```

getLikedSongs.ts

```
import { Song } from "@/types";
import { createServerComponentClient } from "@supabase/auth-helpers-nextjs";
import { cookies } from "next/headers";

const getLikedSongs = async (): Promise<Song[]> => {
  const supabase = createServerComponentClient({
    cookies: cookies
  });

  const {
    data: { session },
  } = await supabase.auth.getSession();

  const { data } = await supabase
    .from('liked_songs')
    .select('*, songs(*)')
    .eq('user_id', session?.user?.id)
    .order('created_at', { ascending: false })

  if (!data) return [];

  return data.map((item) => ({
```

```
    ...item.songs
  }))
};

export default getLikedSongs;
```

getSongById.ts

```
import { createServerComponentClient } from "@supabase/auth-helpers-nextjs";
import { cookies } from "next/headers";

import { Song } from "@/types";

const getSongById = async (id: string): Promise<Song> => {
  const supabase = createServerComponentClient({
    cookies: cookies
  });

  const { data, error } = await supabase
    .from('songs')
    .select('*')
    .eq('id', id)
    .single();

  if (error) {
    console.log(error.message);
  }

  return (data as any) || [];
};

export default getSongById;
```

getSongs.ts

```
import { createServerComponentClient } from "@supabase/auth-helpers-nextjs";
import { cookies } from "next/headers";

import { Song } from "@/types";

const getSongs = async (): Promise<Song[]> => {
  const supabase = createServerComponentClient({
    cookies: cookies
  });

  const { data, error } = await supabase
    .from('songs')
    .select('*')
    .order('created_at', { ascending: false })

  if (error) {
    console.log(error.message);
  }

  return (data as any) || [];
};

export default getSongs;
```

getSongsByTitle.ts

```
import { createServerComponentClient } from "@supabase/auth-helpers-nextjs";
import { cookies, headers } from "next/headers";
```

```
import { Song } from "@/types";

import getSongs from "./getSongs";

const getSongsByTitle = async (title: string): Promise<Song[]> => {
  const supabase = createServerComponentClient({
    cookies: cookies
  });

  if (!title) {
    const allSongs = await getSongs();
    return allSongs;
  }

  const { data, error } = await supabase
    .from('songs')
    .select('*')
    .ilike('title', `%${title}%`)
    .order('created_at', { ascending: false })

  if (error) {
    console.log(error.message);
  }

  return (data as any) || [];
};

export default getSongsByTitle;
```

getSongsByUserId.ts

```
import { createServerComponentClient } from "@supabase/auth-helpers-nextjs";
import { cookies } from "next/headers";

import { Song } from "@/types";

const getSongsByUserId = async (): Promise<Song[]> => {
  const supabase = createServerComponentClient({
    cookies: cookies
  });

  const { data: sessionData, error: sessionError } = await supabase.auth.getSession();

  if (sessionError) {
    console.log(sessionError.message);
    return [];
  }

  const { data, error } = await supabase
    .from('songs')
    .select('*')
    .eq('user_id', sessionData.session?.user.id)
    .order('created_at', { ascending: false })

  if (error) {
    console.log(error.message);
  }
```

```
  return (data as any) || [];
};

export default getSongsByUserId;
```

app/(site)/components/PageContent.tsx

```
"use client";

import { Song } from "@/types";
import useOnPlay from "@/hooks/useOnPlay";
import SongItem from "@/components/SongItem";

interface PageContentProps {
  songs: Song[];
}

const PageContent: React.FC<PageContentProps> = ({
  songs
}) => {
  const onPlay = useOnPlay(songs);

  if (songs.length === 0) {
    return (
      <div className="mt-4 text-neutral-400">
        No songs available.
      </div>
    )
  }

  return (
    <div
      className="
        grid
        grid-cols-2
        sm:grid-cols-3
        md:grid-cols-3
        lg:grid-cols-4
        xl:grid-cols-5
        2xl:grid-cols-8
        gap-4
        mt-4
      "
    >
      {songs.map((item) => (
        <SongItem
          onClick={(id: string) => onPlay(id)}
          key={item.id}
          data={item}
        />
      ))}
    </div>
  );
}

export default PageContent;
```

app/(site)/error.tsx

```
"use client";
```

```
import Box from "@/components/Box";

const Error = () => {
  return (
    <Box className="h-full flex items-center justify-center">
      <div className="text-neutral-400">
        Something went wrong.
      </div>
    </Box>
  );
}

export default Error;
```

app/(site)/loading.tsx

```
"use client";

import { BounceLoader } from "react-spinners";

import Box from "@/components/Box";

const Loading = () => {
  return (
    <Box className="h-full flex items-center justify-center">
      <BounceLoader color="#22c55e" size={40} />
    </Box>
  );
}

export default Loading;
```

app/(site)/page.tsx

```
import getSongs from "@/actions/getSongs";
import Header from "@/components/Header";
import ListItem from "@/components/ListItem";

import PageContent from "./components/PageContent";

export const revalidate = 0;

export default async function Home() {
  const songs = await getSongs();

  return (
    <div
      className="
        bg-neutral-900
        rounded-lg
        h-full
        w-full
        overflow-hidden
        overflow-y-auto
      "
    >
      <Header>
        <div className="mb-2">
          <h1
            className="
            text-white
```

```
            text-3xl
            font-semibold
          ">
            Welcome back
          </h1>
          <div
            className="
              grid
              grid-cols-1
              sm:grid-cols-2
              xl:grid-cols-3
              2xl:grid-cols-4
              gap-3
              mt-4
            "
          >
            <ListItem
              name="Liked Songs"
              image="/images/liked.png"
              href="liked"
            />
          </div>
        </div>
      </Header>
      <div className="mt-2 mb-7 px-6">
        <div className="flex justify-between items-center">
          <h1 className="text-white text-2xl font-semibold">
            Newest songs
          </h1>
        </div>
        <PageContent songs={songs} />
      </div>
    </div>
  )
}
```

app/account/components/AccountContent.tsx

```
"use client";

import { useEffect, useState } from "react";
import { useRouter } from "next/navigation";

import { useUser } from "@/hooks/useUser";
import Button from "@/components/Button";
import useSubscribeModal from "@/hooks/useSubscribeModal";
import { postData } from "@/libs/helpers";

const AccountContent = () => {
  const router = useRouter();
  const subscribeModal = useSubscribeModal();
  const { isLoading, subscription, user } = useUser();

  const [loading, setLoading] = useState(false);

  useEffect(() => {
    if (!isLoading && !user) {
      router.replace('/');
    }
```

```
  }, [isLoading, user, router]);

  const redirectToCustomerPortal = async () => {
   setLoading(true);
   try {
    const { url, error } = await postData({
     url: '/api/create-portal-link'
    });
    window.location.assign(url);
   } catch (error) {
    if (error) return alert((error as Error).message);
   }
   setLoading(false);
  };

  return (
   <div className="mb-7 px-6">
    {!subscription && (
     <div className="flex flex-col gap-y-4">
     <p>No active plan.</p>
     <Button
      onClick={subscribeModal.onOpen}
      className="w-[300px]"
     >
      Subscribe
     </Button>
    </div>
    )}
    {subscription && (
     <div className="flex flex-col gap-y-4">
      <p>You are currently on the
       <b> {subscription?.prices?.products?.name} </b>
       plan.
      </p>
      <Button
       disabled={loading || isLoading}
       onClick={redirectToCustomerPortal}
       className="w-[300px]"
      >
       Open customer portal
      </Button>
     </div>
    )}
   </div>
  );
}

export default AccountContent;
```

app/account/error.tsx

```
"use client";

import Box from "@/components/Box";

const Error = () => {
 return (
  <Box className="h-full flex items-center justify-center">
   <div className="text-neutral-400">
```

```
      Something went wrong.
    </div>
  </Box>
 );
}

export default Error;
```

app/account/loading.tsx

```
"use client";

import { BounceLoader } from "react-spinners";

import Box from "@/components/Box";

const Loading = () => {
 return (
  <Box className="h-full flex items-center justify-center">
   <BounceLoader color="#22c55e" size={40} />
  </Box>
 );
}

export default Loading;
```

app/account/page.tsx

```
import Header from "@/components/Header";

import AccountContent from "./components/AccountContent";

const Account = () => {
 return (
  <div
   className="
    bg-neutral-900
    rounded-lg
    h-full
    w-full
    overflow-hidden
    overflow-y-auto
   "
  >
   <Header className="from-bg-neutral-900">
    <div className="mb-2 flex flex-col gap-y-6">
     <h1 className="text-white text-3xl font-semibold">
      Account Settings
     </h1>
    </div>
   </Header>
   <AccountContent />
  </div>
 )
}

export default Account;
```

app/api/create-checkout-session/route.ts

```
import { createRouteHandlerClient } from '@supabase/auth-helpers-nextjs';
import { cookies } from "next/headers";
import { NextResponse } from 'next/server';
```

```
import { stripe } from '@/libs/stripe';
import { getURL } from '@/libs/helpers';
import { createOrRetrieveCustomer } from '@/libs/supabaseAdmin';

export async function POST(
  request: Request
) {
  const { price, quantity = 1, metadata = {} } = await request.json();

  try {
    const supabase = createRouteHandlerClient({
      cookies
     });      const {
     data: { user }
    } = await supabase.auth.getUser();

    const customer = await createOrRetrieveCustomer({
      uuid: user?.id || '',
      email: user?.email || ''
    });

    const session = await stripe.checkout.sessions.create({
      payment_method_types: ['card'],
      billing_address_collection: 'required',
      customer,
      line_items: [
        {
          price: price.id,
          quantity
        }
      ],
      mode: 'subscription',
      allow_promotion_codes: true,
      subscription_data: {
        trial_from_plan: true,
        metadata
      },
      success_url: `${getURL()}/account`,
      cancel_url: `${getURL()}/`
    });

    return NextResponse.json({ sessionId: session.id });
  } catch (err: any) {
    console.log(err);
    return new NextResponse('Internal Error', { status: 500 });
  }
}
```

app/api/create-portal-link/route.ts

```
import { createRouteHandlerClient } from '@supabase/auth-helpers-nextjs'
import { cookies } from "next/headers";
import { NextResponse } from 'next/server';

import { stripe } from '@/libs/stripe';
import { getURL } from '@/libs/helpers';
import { createOrRetrieveCustomer } from '@/libs/supabaseAdmin';
```

```
export async function POST() {
  try {
    const supabase = createRouteHandlerClient({
     cookies
    });

    const {
     data: { user }
    } = await supabase.auth.getUser();

    if (!user) throw Error('Could not get user');
    const customer = await createOrRetrieveCustomer({
     uuid: user.id || '',
     email: user.email || ''
    });

    if (!customer) throw Error('Could not get customer');
    const { url } = await stripe.billingPortal.sessions.create({
     customer,
     return_url: `${getURL()}/account`
    });

    return NextResponse.json({ url });
  } catch (err: any) {
    console.log(err);
    new NextResponse('Internal Error', { status: 500 })
  }
};
```

app/api/webhooks/route.ts

```
import Stripe from 'stripe';
import { NextResponse } from 'next/server';
import { headers } from 'next/headers';

import { stripe } from '@/libs/stripe';
import {
  upsertProductRecord,
  upsertPriceRecord,
  manageSubscriptionStatusChange
} from '@/libs/supabaseAdmin';

const relevantEvents = new Set([
  'product.created',
  'product.updated',
  'price.created',
  'price.updated',
  'checkout.session.completed',
  'customer.subscription.created',
  'customer.subscription.updated',
  'customer.subscription.deleted'
]);

export async function POST(
  request: Request
) {
    const body = await request.text()
    const sig = headers().get('Stripe-Signature');
```

```
    const webhookSecret =
      process.env.STRIPE_WEBHOOK_SECRET_LIVE ??
      process.env.STRIPE_WEBHOOK_SECRET;
    let event: Stripe.Event;

    try {
      if (!sig || !webhookSecret) return;
      event = stripe.webhooks.constructEvent(body, sig, webhookSecret);
    } catch (err: any) {
      console.log(`❌ Error message: ${err.message}`);
      return new NextResponse(`Webhook Error: ${err.message}`, { status: 400 });
    }

  if (relevantEvents.has(event.type)) {
    try {
      switch (event.type) {
        case 'product.created':
        case 'product.updated':
          await upsertProductRecord(event.data.object as Stripe.Product);
          break;
        case 'price.created':
        case 'price.updated':
          await upsertPriceRecord(event.data.object as Stripe.Price);
          break;
        case 'customer.subscription.created':
        case 'customer.subscription.updated':
        case 'customer.subscription.deleted':
          const subscription = event.data.object as Stripe.Subscription;
          await manageSubscriptionStatusChange(
            subscription.id,
            subscription.customer as string,
            event.type === 'customer.subscription.created'
          );
          break;
        case 'checkout.session.completed':
          const checkoutSession = event.data
            .object as Stripe.Checkout.Session;
          if (checkoutSession.mode === 'subscription') {
            const subscriptionId = checkoutSession.subscription;
            await manageSubscriptionStatusChange(
              subscriptionId as string,
              checkoutSession.customer as string,
              true
            );
          }
          break;
        default:
          throw new Error('Unhandled relevant event!');
      }
    } catch (error) {
      console.log(error);
      return new NextResponse('Webhook error: "Webhook handler failed. View logs."', { status:
400 });
    }
  }

  return NextResponse.json({ received: true }, { status: 200 });
```

```
};
```

components/AuthModal.tsx

```
"use client";

import React, { useEffect } from 'react';
import { Auth } from '@supabase/auth-ui-react';
import { ThemeSupa } from '@supabase/auth-ui-shared';
import {
 useSessionContext,
 useSupabaseClient
} from '@supabase/auth-helpers-react';
import { useRouter } from 'next/navigation';

import useAuthModal from "@/hooks/useAuthModal";

import Modal from './Modal';

const AuthModal = () => {
 const { session } = useSessionContext();
 const router = useRouter();
 const { onClose, isOpen } = useAuthModal();

 const supabaseClient = useSupabaseClient();

 useEffect(() => {
  if (session) {
   router.refresh();
   onClose();
  }
 }, [session, router, onClose]);

 const onChange = (open: boolean) => {
  if (!open) {
   onClose();
  }
 }

 return (
  <Modal
   title="Welcome back"
   description="Login to your account."
   isOpen={isOpen}
   onChange={onChange}
  >
   <Auth
    supabaseClient={supabaseClient}
    providers={['github']}
    magicLink={true}
    appearance={{
     theme: ThemeSupa,
     variables: {
      default: {
       colors: {
        brand: '#404040',
        brandAccent: '#22c55e'
       }
      }
```

```
        }
       }}
       theme="dark"
      />
    </Modal>
  );
}

export default AuthModal;
```

components/Box.tsx

```
import { twMerge } from "tailwind-merge";

interface BoxProps {
  children: React.ReactNode;
  className?: string;
}

const Box: React.FC<BoxProps> = ({
  children,
  className
 }) => {
  return (
    <div
      className={twMerge(
        `
        bg-neutral-900
        rounded-lg
        h-fit
        w-full
        `,
        className
      )}>
      {children}
    </div>
  );
}

export default Box;
```

components/Button.tsx

```
import { forwardRef } from "react";
import { twMerge } from "tailwind-merge";

export interface ButtonProps
  extends React.ButtonHTMLAttributes<HTMLButtonElement> {}

const Button = forwardRef<HTMLButtonElement, ButtonProps>(({
  className,
  children,
  disabled,
  type = 'button',
  ...props
}, ref) => {
  return (
    <button
      type={type}
      className={twMerge(
        `
```

```
        w-full
        rounded-full
        bg-green-500
        border
        border-transparent
        px-3
        py-3
        disabled:cursor-not-allowed
        disabled:opacity-50
        text-black
        font-bold
        hover:opacity-75
        transition
      `,
        disabled && 'opacity-75 cursor-not-allowed',
        className
      )}
      disabled={disabled}
      ref={ref}
      {...props}
    >
      {children}
    </button>
  );
});

Button.displayName = "Button";

export default Button;
```

components/Header.tsx

```
"use client";

import { twMerge } from "tailwind-merge";
import { RxCaretLeft, RxCaretRight } from "react-icons/rx";
import { useRouter } from "next/navigation";
import { FaUserAlt } from "react-icons/fa";
import { useSupabaseClient } from '@supabase/auth-helpers-react';
import { toast } from "react-hot-toast";
import { HiHome } from "react-icons/hi";
import { BiSearch } from "react-icons/bi";

import useAuthModal from "@/hooks/useAuthModal";
import { useUser } from "@/hooks/useUser";
import usePlayer from "@/hooks/usePlayer";

import Button from "./Button";

interface HeaderProps {
  children: React.ReactNode;
  className?: string;
}

const Header: React.FC<HeaderProps> = ({
  children,
  className,
}) => {
  const player = usePlayer();
```

```
const router = useRouter();
const authModal = useAuthModal();

const supabaseClient = useSupabaseClient();
const { user } = useUser();

const handleLogout = async () => {
 const { error } = await supabaseClient.auth.signOut();
 player.reset();
 router.refresh();

 if (error) {
  toast.error(error.message);
 }
}

return (
  <div
   className={twMerge(`
    h-fit
    bg-gradient-to-b
    from-emerald-800
    p-6
    `,
    className
   )}>
   <div className="w-full mb-4 flex items-center justify-between">
    <div className="hidden md:flex gap-x-2 items-center">
     <button
      onClick={() => router.back()}
      className="
       rounded-full
       bg-black
       flex
       items-center
       justify-center
       cursor-pointer
       hover:opacity-75
       transition
      "
     >
      <RxCaretLeft className="text-white" size={35} />
     </button>
     <button
      onClick={() => router.forward()}
      className="
       rounded-full
       bg-black
       flex
       items-center
       justify-center
       cursor-pointer
       hover:opacity-75
       transition
      "
     >
      <RxCaretRight className="text-white" size={35} />
```

```
  </button>
 </div>
 <div className="flex md:hidden gap-x-2 items-center">
  <button
   onClick={() => router.push('/')}
   className="
    rounded-full
    p-2
    bg-white
    flex
    items-center
    justify-center
    cursor-pointer
    hover:opacity-75
    transition
   "
  >
   <HiHome className="text-black" size={20} />
  </button>
  <button
   onClick={() => router.push('/search')}
   className="
    rounded-full
    p-2
    bg-white
    flex
    items-center
    justify-center
    cursor-pointer
    hover:opacity-75
    transition
   "
  >
   <BiSearch className="text-black" size={20} />
  </button>
 </div>
 <div className="flex justify-between items-center gap-x-4">
  {user ? (
   <div className="flex gap-x-4 items-center">
    <Button
     onClick={handleLogout}
     className="bg-white px-6 py-2"
    >
     Logout
    </Button>
    <Button
     onClick={() => router.push('/account')}
     className="bg-white"
    >
     <FaUserAlt />
    </Button>
   </div>
  ) : (
   <>
    <div>
     <Button
      onClick={authModal.onOpen}
```

```
              className="
                bg-transparent
                text-neutral-300
                font-medium
              "
            >
              Sign up
            </Button>
          </div>
          <div>
            <Button
              onClick={authModal.onOpen}
              className="bg-white px-6 py-2"
            >
              Log in
            </Button>
          </div>
        </>
      )}
    </div>
  </div>
  {children}
 </div>
);
}

export default Header;
```

components/Input.tsx

```
import { forwardRef } from "react";
import { twMerge } from "tailwind-merge"

export interface InputProps
 extends React.InputHTMLAttributes<HTMLInputElement> {}

const Input = forwardRef<HTMLInputElement, InputProps>(({
 className,
 type,
 disabled,
 ...props
}, ref) => {
 return (
  <input
   type={type}
   className={twMerge(
    `
     flex
     w-full
     rounded-md
     bg-neutral-700
     border
     border-transparent
     px-3
     py-3
     text-sm
     file:border-0
     file:bg-transparent
     file:text-sm
```

```
        file:font-medium
        placeholder:text-neutral-400
        disabled:cursor-not-allowed
        disabled:opacity-50
        focus:outline-none
      `,
        disabled && 'opacity-75',
        className
      )}
      disabled={disabled}
      ref={ref}
      {...props}
    />
  )
});

Input.displayName = "Input";

export default Input
```

components/Library.tsx

```
"use client";

import { TbPlaylist } from "react-icons/tb";
import { AiOutlinePlus } from "react-icons/ai";

import { Song } from "@/types";
import useUploadModal from "@/hooks/useUploadModal";
import { useUser } from "@/hooks/useUser";
import useAuthModal from "@/hooks/useAuthModal";
import useSubscribeModal from "@/hooks/useSubscribeModal";
import useOnPlay from "@/hooks/useOnPlay";

import MediaItem from "./MediaItem";

interface LibraryProps {
  songs: Song[];
}

const Library: React.FC<LibraryProps> = ({
  songs
}) => {
  const { user, subscription } = useUser();
  const uploadModal = useUploadModal();
  const authModal = useAuthModal();
  const subscribeModal = useSubscribeModal();

  const onPlay = useOnPlay(songs);

  const onClick = () => {
    if (!user) {
      return authModal.onOpen();
    }

    if (!subscription) {
      return subscribeModal.onOpen();
    }
```

```
    return uploadModal.onOpen();
  }

  return (
    <div className="flex flex-col">
      <div className="flex items-center justify-between px-5 pt-4">
        <div className="inline-flex items-center gap-x-2">
          <TbPlaylist className="text-neutral-400" size={26} />
          <p className="text-neutral-400 font-medium text-md">
            Your Library
          </p>
        </div>
        <AiOutlinePlus
          onClick={onClick}
          size={20}
          className="
            text-neutral-400
            cursor-pointer
            hover:text-white
            transition
          "
        />
      </div>
      <div className="flex flex-col gap-y-2 mt-4 px-3">
        {songs.map((item) => (
          <MediaItem
            onClick={(id: string) => onPlay(id)}
            key={item.id}
            data={item}
          />
        ))}
      </div>
    </div>
  );
}

export default Library;
```

components/LikeButton.tsx

```
"use client";

import { useEffect, useState } from "react";
import { AiOutlineHeart, AiFillHeart } from "react-icons/ai";
import { useRouter } from "next/navigation";
import { toast } from "react-hot-toast";
import { useSessionContext } from "@supabase/auth-helpers-react";

import { useUser } from "@/hooks/useUser";
import useAuthModal from "@/hooks/useAuthModal";

interface LikeButtonProps {
  songId: string;
};

const LikeButton: React.FC<LikeButtonProps> = ({
  songId
}) => {
  const router = useRouter();
```

```
const {
 supabaseClient
} = useSessionContext();
const authModal = useAuthModal();
const { user } = useUser();

const [isLiked, setIsLiked] = useState<boolean>(false);

useEffect(() => {
 if (!user?.id) {
  return;
 }

 const fetchData = async () => {
  const { data, error } = await supabaseClient
   .from('liked_songs')
   .select('*')
   .eq('user_id', user.id)
   .eq('song_id', songId)
   .single();

  if (!error && data) {
   setIsLiked(true);
  }
 }

 fetchData();
}, [songId, supabaseClient, user?.id]);

const Icon = isLiked ? AiFillHeart : AiOutlineHeart;

const handleLike = async () => {
 if (!user) {
  return authModal.onOpen();
 }

 if (isLiked) {
  const { error } = await supabaseClient
   .from('liked_songs')
   .delete()
   .eq('user_id', user.id)
   .eq('song_id', songId)

  if (error) {
   toast.error(error.message);
  } else {
   setIsLiked(false);
  }
 } else {
  const { error } = await supabaseClient
   .from('liked_songs')
   .insert({
    song_id: songId,
    user_id: user.id
   });

  if (error) {
```

```
        toast.error(error.message);
      } else {
        setIsLiked(true);
        toast.success('Success');
      }
    }

    router.refresh();
  }

  return (
    <button
      className="
        cursor-pointer
        hover:opacity-75
        transition
      "
      onClick={handleLike}
    >
      <Icon color={isLiked ? '#22c55e' : 'white'} size={25} />
    </button>
  );
}

export default LikeButton;
```

components/ListItem.tsx

```
"use client";

import Image from "next/image";
import { useRouter } from "next/navigation";
import { FaPlay } from "react-icons/fa";

import useAuthModal from "@/hooks/useAuthModal";
import { useUser } from "@/hooks/useUser";

interface ListItemProps {
  image: string;
  name: string;
  href: string;
}

const ListItem: React.FC<ListItemProps> = ({
  image,
  name,
  href,
}) => {
  const router = useRouter();
  const authModal = useAuthModal();
  const { user } = useUser();

  const onClick = () => {
    if (!user) {
      return authModal.onOpen();
    }

    router.push(href);
  };
```

```
  return (
    <button
      onClick={onClick}
      className="
        relative
        group
        flex
        items-center
        rounded-md
        overflow-hidden
        gap-x-4
        bg-neutral-100/10
        cursor-pointer
        hover:bg-neutral-100/20
        transition
        pr-4
      "
    >
      <div className="relative min-h-[64px] min-w-[64px]">
        <Image
          className="object-cover"
          src={image}
          fill
          alt="Image"
        />
      </div>
      <p className="font-medium truncate py-5">
        {name}
      </p>
      <div
        className="
          absolute
          transition
          opacity-0
          rounded-full
          flex
          items-center
          justify-center
          bg-green-500
          p-4
          drop-shadow-md
          right-5
          group-hover:opacity-100
          hover:scale-110
        "
      >
        <FaPlay className="text-black" />
      </div>
    </button>
  );
}

export default ListItem;
```

components/MediaItem.tsx

```
"use client";
```

```
import Image from "next/image";

import useLoadImage from "@/hooks/useLoadImage";
import { Song } from "@/types";
import usePlayer from "@/hooks/usePlayer";

interface MediaItemProps {
 data: Song;
 onClick?: (id: string) => void;
}

const MediaItem: React.FC<MediaItemProps> = ({
 data,
 onClick,
}) => {
 const player = usePlayer();
 const imageUrl = useLoadImage(data);

 const handleClick = () => {
  if (onClick) {
   return onClick(data.id);
  }

  return player.setId(data.id);
 };

 return (
  <div
   onClick={handleClick}
   className="
    flex
    items-center
    gap-x-3
    cursor-pointer
    hover:bg-neutral-800/50
    w-full
    p-2
    rounded-md
   "
  >
   <div
    className="
     relative
     rounded-md
     min-h-[48px]
     min-w-[48px]
     overflow-hidden
    "
   >
    <Image
     fill
     src={imageUrl || "/images/music-placeholder.png"}
     alt="MediaItem"
     className="object-cover"
    />
   </div>
   <div className="flex flex-col gap-y-1 overflow-hidden">
```

```
      <p className="text-white truncate">{data.title}</p>
      <p className="text-neutral-400 text-sm truncate">
       By {data.author}
      </p>
     </div>
    </div>
  );
}

export default MediaItem;
```

components/Modal.tsx

```
import * as Dialog from '@radix-ui/react-dialog';
import { IoMdClose } from 'react-icons/io';

interface ModalProps {
  isOpen: boolean;
  onChange: (open: boolean) => void;
  title: string;
  description: string;
  children: React.ReactNode;
}

const Modal: React.FC<ModalProps> = ({
  isOpen,
  onChange,
  title,
  description,
  children
}) => {
  return (
    <Dialog.Root open={isOpen} defaultOpen={isOpen} onOpenChange={onChange}>
      <Dialog.Portal>
        <Dialog.Overlay
          className="
            bg-neutral-900/90
            backdrop-blur-sm
            fixed
            inset-0
          "
        />
        <Dialog.Content
          className="
            fixed
            drop-shadow-md
            border
            border-neutral-700
            top-[50%]
            left-[50%]
            max-h-full
            h-full
            md:h-auto
            md:max-h-[85vh]
            w-full
            md:w-[90vw]
            md:max-w-[450px]
            translate-x-[-50%]
            translate-y-[-50%]
```

```
          rounded-md
          bg-neutral-800
          p-[25px]
          focus:outline-none
        ">
          <Dialog.Title
          className="
            text-xl
            text-center
            font-bold
            mb-4
          "
          >
            {title}
          </Dialog.Title>
          <Dialog.Description
          className="
            mb-5
            text-sm
            leading-normal
            text-center
          "
          >
            {description}
          </Dialog.Description>
          <div>
            {children}
          </div>
          <Dialog.Close asChild>
            <button
            className="
              text-neutral-400
              hover:text-white
              absolute
              top-[10px]
              right-[10px]
              inline-flex
              h-[25px]
              w-[25px]
              appearance-none
              items-center
              justify-center
              rounded-full
              focus:outline-none
            "
            aria-label="Close"
            >
              <IoMdClose />
            </button>
          </Dialog.Close>
        </Dialog.Content>
      </Dialog.Portal>
    </Dialog.Root>
  );
}

export default Modal;
```

components/PlayButton.tsx

```
import { FaPlay } from "react-icons/fa";

const PlayButton = () => {
 return (
  <button
   className="
    transition
    opacity-0
    rounded-full
    flex
    items-center
    justify-center
    bg-green-500
    p-4
    drop-shadow-md
    translate
    translate-y-1/4
    group-hover:opacity-100
    group-hover:translate-y-0
    hover:scale-110
   "
  >
   <FaPlay className="text-black" />
  </button>
 );
}

export default PlayButton;
```

components/Player.tsx

```
"use client";

import usePlayer from "@/hooks/usePlayer";
import useLoadSongUrl from "@/hooks/useLoadSongUrl";
import useGetSongById from "@/hooks/useGetSongById";

import PlayerContent from "./PlayerContent";

const Player = () => {
 const player = usePlayer();
 const { song } = useGetSongById(player.activeId);

 const songUrl = useLoadSongUrl(song!);

 if (!song || !songUrl || !player.activeId) {
  return null;
 }

 return (
  <div
   className="
    fixed
    bottom-0
    bg-black
    w-full
    py-2
    h-[80px]
```

```
      px-4
    "
  >
    <PlayerContent key={songUrl} song={song} songUrl={songUrl} />
  </div>
 );
}

export default Player;
```

components/PlayerContent.tsx

```
"use client";

import useSound from "use-sound";
import { useEffect, useState } from "react";
import { BsPauseFill, BsPlayFill } from "react-icons/bs";
import { HiSpeakerWave, HiSpeakerXMark } from "react-icons/hi2";
import { AiFillStepBackward, AiFillStepForward } from "react-icons/ai";

import { Song } from "@/types";
import usePlayer from "@/hooks/usePlayer";

import LikeButton from "./LikeButton";
import MediaItem from "./MediaItem";
import Slider from "./Slider";


interface PlayerContentProps {
 song: Song;
 songUrl: string;
}

const PlayerContent: React.FC<PlayerContentProps> = ({
 song,
 songUrl
}) => {
 const player = usePlayer();
 const [volume, setVolume] = useState(1);
 const [isPlaying, setIsPlaying] = useState(false);

 const Icon = isPlaying ? BsPauseFill : BsPlayFill;
 const VolumeIcon = volume === 0 ? HiSpeakerXMark : HiSpeakerWave;

 const onPlayNext = () => {
  if (player.ids.length === 0) {
   return;
  }

  const currentIndex = player.ids.findIndex((id) => id === player.activeId);
  const nextSong = player.ids[currentIndex + 1];

  if (!nextSong) {
   return player.setId(player.ids[0]);
  }

  player.setId(nextSong);
 }
```

```
const onPlayPrevious = () => {
  if (player.ids.length === 0) {
    return;
  }

  const currentIndex = player.ids.findIndex((id) => id === player.activeId);
  const previousSong = player.ids[currentIndex - 1];

  if (!previousSong) {
    return player.setId(player.ids[player.ids.length - 1]);
  }

  player.setId(previousSong);
}

const [play, { pause, sound }] = useSound(
  songUrl,
  {
    volume: volume,
    onplay: () => setIsPlaying(true),
    onend: () => {
      setIsPlaying(false);
      onPlayNext();
    },
    onpause: () => setIsPlaying(false),
    format: ['mp3']
  }
);

useEffect(() => {
  sound?.play();

  return () => {
    sound?.unload();
  }
}, [sound]);

const handlePlay = () => {
  if (!isPlaying) {
    play();
  } else {
    pause();
  }
}

const toggleMute = () => {
  if (volume === 0) {
    setVolume(1);
  } else {
    setVolume(0);
  }
}

return (
  <div className="grid grid-cols-2 md:grid-cols-3 h-full">
    <div className="flex w-full justify-start">
      <div className="flex items-center gap-x-4">
```

```
          <MediaItem data={song} />
          <LikeButton songId={song.id} />
        </div>
      </div>

      <div
        className="
          flex
          md:hidden
          col-auto
          w-full
          justify-end
          items-center
        "
      >
        <div
          onClick={handlePlay}
          className="
            h-10
            w-10
            flex
            items-center
            justify-center
            rounded-full
            bg-white
            p-1
            cursor-pointer
          "
        >
          <Icon size={30} className="text-black" />
        </div>
      </div>

      <div
        className="
          hidden
          h-full
          md:flex
          justify-center
          items-center
          w-full
          max-w-[722px]
          gap-x-6
        "
      >
        <AiFillStepBackward
          onClick={onPlayPrevious}
          size={30}
          className="
            text-neutral-400
            cursor-pointer
            hover:text-white
            transition
          "
        />
        <div
          onClick={handlePlay}
```

```
            className="
              flex
              items-center
              justify-center
              h-10
              w-10
              rounded-full
              bg-white
              p-1
              cursor-pointer
            "
          >
            <Icon size={30} className="text-black" />
          </div>
          <AiFillStepForward
            onClick={onPlayNext}
            size={30}
            className="
              text-neutral-400
              cursor-pointer
              hover:text-white
              transition
            "
          />
        </div>

        <div className="hidden md:flex w-full justify-end pr-2">
          <div className="flex items-center gap-x-2 w-[120px]">
            <VolumeIcon
              onClick={toggleMute}
              className="cursor-pointer"
              size={34}
            />
            <Slider
              value={volume}
              onChange={(value) => setVolume(value)}
            />
          </div>
        </div>

      </div>
    );
}

export default PlayerContent;
```

components/SearchInput.tsx

```
"use client";

import qs from "query-string";
import { useEffect, useState } from "react";
import { useRouter } from "next/navigation";

import useDebounce from "@/hooks/useDebounce";

import Input from "./Input";

const SearchInput = () => {
```

```
  const router = useRouter();
  const [value, setValue] = useState<string>('');
  const debouncedValue = useDebounce<string>(value, 500);

  useEffect(() => {
    const query = {
      title: debouncedValue,
    };

    const url = qs.stringifyUrl({
      url: '/search',
      query
    });

    router.push(url);
  }, [debouncedValue, router]);

  return (
    <Input
      placeholder="What do you want to listen to?"
      value={value}
      onChange={(e) => setValue(e.target.value)}
    />
  );
}

export default SearchInput;
```

components/Sidebar.tsx

```
"use client";

import { HiHome } from "react-icons/hi";
import { BiSearch } from "react-icons/bi";
import { twMerge } from "tailwind-merge";
import { usePathname } from "next/navigation";

import { Song } from "@/types";
import usePlayer from "@/hooks/usePlayer";

import SidebarItem from "./SidebarItem";
import Box from "./Box";
import Library from "./Library";
import { useMemo } from "react";

interface SidebarProps {
  children: React.ReactNode;
  songs: Song[];
}

const Sidebar = ({ children, songs }: SidebarProps) => {
  const pathname = usePathname();
  const player = usePlayer();

  const routes = useMemo(() => [
    {
      icon: HiHome,
      label: 'Home',
      active: pathname !== '/search',
```

```
      href: '/'
    },
    {
      icon: BiSearch,
      label: 'Search',
      href: '/search',
      active: pathname === '/search'
    },
  ], [pathname]);

  return (
    <div
      className={twMerge(`
        flex
        h-full
        `,
        player.activeId && 'h-[calc(100%-80px)]'
      )}
    >
      <div
        className="
          hidden
          md:flex
          flex-col
          gap-y-2
          bg-black
          h-full
          w-[300px]
          p-2
        "
      >
        <Box>
          <div className="flex flex-col gap-y-4 px-5 py-4">
            {routes.map((item) => (
              <SidebarItem key={item.label} {...item} />
            ))}
          </div>
        </Box>
        <Box className="overflow-y-auto h-full">
          <Library songs={songs} />
        </Box>
      </div>
      <main className="h-full flex-1 overflow-y-auto py-2">
        {children}
      </main>
    </div>
  );
}

export default Sidebar;
```

components/SidebarItem.tsx

```
import Link from 'next/link';
import { IconType } from 'react-icons';
import { twMerge } from 'tailwind-merge';

interface SidebarItemProps {
  icon: IconType;
```

```
  label: string;
  active?: boolean;
  href: string;
}

const SidebarItem: React.FC<SidebarItemProps> = ({
  icon: Icon,
  label,
  active,
  href
}) => {
  return (
    <Link
      href={href}
      className={twMerge(`
        flex
        flex-row
        h-auto
        items-center
        w-full
        gap-x-4
        text-md
        font-medium
        cursor-pointer
        hover:text-white
        transition
        text-neutral-400
        py-1`,
        active && "text-white"
        )
      }
    >
      <Icon size={26} />
      <p className="truncate w-100">{label}</p>
    </Link>
  );
}

export default SidebarItem;
```

components/Slider.tsx

```
"use client";

import * as RadixSlider from '@radix-ui/react-slider';

interface SlideProps {
  value?: number;
  onChange?: (value: number) => void;
}

const Slider: React.FC<SlideProps> = ({
  value = 1,
  onChange
}) => {
  const handleChange = (newValue: number[]) => {
    onChange?.(newValue[0]);
  };
```

```
  return (
    <RadixSlider.Root
      className="
        relative
        flex
        items-center
        select-none
        touch-none
        w-full
        h-10
      "
      defaultValue={[1]}
      value={[value]}
      onValueChange={handleChange}
      max={1}
      step={0.1}
      aria-label="Volume"
    >
      <RadixSlider.Track
        className="
          bg-neutral-600
          relative
          grow
          rounded-full
          h-[3px]
        "
      >
        <RadixSlider.Range
          className="
            absolute
            bg-white
            rounded-full
            h-full
          "
        />
      </RadixSlider.Track>
    </RadixSlider.Root>
  );
}

export default Slider;
```