МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ
ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО
«___»_____2022 р.

# КВАЛІФІКАЦІЙНА РОБОТА

**(ПОЯСНЮВАЛЬНА ЗАПИСКА)**

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТР
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
«ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ПРОЕКТУВАННЯ»

**Тема: «Браузерний музичний плеєр на базі фреймворку Vue.js»**

Виконавець: Дарина ДУДАР

Керівник: к.т.н., доц. Олег ЗУДОВ

Нормоконтролер: к.т.н., доц. Олена ТОЛСТІКОВА

КИЇВ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *кібербезпеки, комп'ютерної та програмної інженерії*
Кафедра *Комп'ютерних інформаційних технологій*
Спеціальність *122 «Комп'ютерні науки»*
Освітньо-професійна програма *«Інформаційні технології проектування»*

ЗАТВЕРДЖУЮ:
завідувач кафедри КІТ
_____Аліна САВЧЕНКО
(підпис)
«____»_____2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи
Дудар Дарини Вікторівни
(ПІБ випускника )

1. Тема роботи: «Браузерний музичний плеєр на базі фреймворку Vue.js» затверджена наказом ректора № 1774/ст від 28.09.2022р.
2. Термін виконання роботи: з 26 вересня 2022 року по 27 листопада 2022 року.
3. Вихідні дані до роботи: Браузерний музичний плеєр на базі фреймворку Vue.js.
4. Зміст пояснювальної записки: 1. Аналіз предметної області. 2. Аналіз та порівняння обраних інструментів і технологій для розробки web-додатку. 3. Розробка інтерфейсу web-додатку. 4. Програмна реалізація web-додатку.
5. Перелік обов'язкового ілюстративного матеріалу: потрібно перелічити назву основних слайдів.

6. Календарний план-графік

| № з/п | Завдання | Термін виконання | Підпис керівника |
|---|---|---|---|
| 1. | Аналіз предметної області та огляд аналогів. Написання 1 розділу, представлення керівнику | 26.09.2022-09.10.2022 | |
| 2. | Вибір та опис використаних технологій. Написання 2 розділу, представлення керівнику | 10.10.2022-23.10.2022 | |
| 3. | Розробка інтерфейсу додатка за допомогою обраного графічного редактора. Написання 3 розділу, представлення керівнику | 24.10.2022-30.10.2022 | |
| 4. | Розробка програмної реалізації додатка за допомогою обраного стеку технологій. Написання 4 розділу, представлення керівнику | 31.10.2022-14.11.2022 | |
| 5. | Загальне редагування та друк пояснювальної записки | 15.11.2022-20.11.2022 | |
| 6. | Проходження нормоконтролю, перепліт пояснювальної записки. | 14.11.2022-20.10.2022 | |
| 7. | Розробка тексту доповіді. Оформлення графічного матеріалу для презентації | 20.11.2022-22.11.2022 | |

7. Дата видачі завдання 26.09.2022р.

Керівник кваліфікаційної роботи _____ Олег ЗУДОВ
(підпис керівника)

Завдання прийняв до виконання _____ Дарина ДУДАР
(підпис випускника)

# РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи на тему: «Браузерний музичний плеєр на базі фреймворку Vue.js» містить: 85 сторінки, 47 рисунків, 2 таблиці, 20 інформаційних джерел, 3 додатки.

**Об’єктом дослідження** є проектування та розробка веб-додатку музичного плеєру.

**Предметом дослідження** є процес розробки веб-додатків.

**Метою кваліфікаційної роботи** є отримання готової програмної реалізації браузерного музичного плеєру, з використанням сучасних технологій та інструментів розробки веб-додатків.

**Методи дослідження –** мова програмування JavaScript, HTML, CSS, фреймворк Vue.js, бібліотеки VueX та VueRouter, редактор коду Microsoft Visual Studio Code, графічний редактор Figma, система контролю версій Git.

**Отримані результати**: спроектовано, реалізовано і протестовано веб-додаток, який дозволяє прослуховувати музичні композиції та створювати користувацькі плейлисти.

Веб-додаток містить головну сторінку, на якій відображено останні новинки, рекомендації до прослуховуванн для користувача, сторінку з користувацькими плейлистами, окремі сторінки для певних виконавців і їх робіт.

Результати випускної кваліфікаційної роботи рекомендується використовувати для подальшого розвитку, додання та розширення функціоналу web-додатку.

WEB-ДОДАТОК, МОВА ПРОГРАМУВАННЯ JAVASCRIPT, РОЗРОБКА, ФРЕЙМВОРК, VUE.JS, МУЗИЧНИЙ ПЛЕЄР, МУЗИКА.

**ЗМІСТ**

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ ..... 8
ВСТУП ..... 9
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ..... 11
1.1. Розвиток цифрової музики ..... 11
1.1.1. Початок цифрового звукозапису ..... 11
1.1.2. Початок 2000-х. Музика підкорює Інтернет-простір ..... 13
1.1.3. Сучасна ситуація з музикою ..... 13
1.2. Поняття і принципи роботи web-додатків ..... 14
1.2.1. Відмінність понять «Web-додаток» та «Web-сайт» ..... 14
1.2.2. Принцип роботи web-програми ..... 15
1.3. Найпопулярніші web-сервіси для прослуховування музики та їх аналіз ..... 15
1.3.1. Глобальна статистика ..... 15
1.3.2. Огляд найпопулярніших сервісів ..... 16
1.3.3. Порівняльна таблиця сервісів для прослуховування музики ..... 21
1.3.4. Результати проведеного аналізу додатків-аналогів ..... 22
1.4. Постановка задачі розробки web-додатку ..... 23
ВИСНОВОК ДО РОЗДІЛУ 1 ..... 25
РОЗДІЛ 2. АНАЛІЗ ТА ПОРІВНЯННЯ ОБРАНИХ ІНСТРУМЕНІВ І ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ WEB-ДОДАТКУ ..... 26
2.1. Середовище розробки VS Code ..... 26
2.2. Дизайн ..... 28
2.2.1. Figma ..... 28
2.3. HTML, CSS, мова програмування JavaScript ..... 29
2.3.1. HTML ..... 29
2.3.2. CSS ..... 31
2.2.3. JavaScript ..... 33
2.4. Фреймворки у веб-розробці та їх порівняння ..... 34
2.4.1. Поняття фреймворку ..... 34

2.4.2. React ..... 35
2.4.3. Angular ..... 38
2.4.4. Vue ..... 39
2.4.5. Порівняння фреймворків ..... 40
2.5. Інструменти і технології для Vue.js ..... 42
2.5.1. Бібліотека VueX ..... 42
2.5.2. Vue Router ..... 42
2.6. Система контролю версій Git ..... 43
2.7. Технології для розробки бази даних ..... 43
ВИСНОВОК ДО РОЗДІЛУ 2 ..... 45
РОЗДІЛ 3. РОЗРОБКА ІНТЕРФЕЙСУ WEB-ДОДАТКУ ..... 46
3.1. UI/UX дизайн і його важливість ..... 46
3.2. Створення макету і налаштування сітки ..... 47
3.3. Розробка логотипу ..... 49
3.3.1. Логотип, його основні функції та види ..... 49
3.3.2. Концепції логотипу для РИТМ ..... 51
3.4. Створення дизайну сторінки «Головна» ..... 52
3.4.1. Навігація ..... 53
3.4.2. Банер ..... 54
3.4.3. Дизайн для компоненту композиції ..... 54
3.4.4. Рекомендації ..... 55
3.4.5. Дизайн головної сторінки ..... 55
3.5. Створення сторінки «Плейлисти» ..... 56
3.6. Створення дизайну сторінки плейлисту ..... 58
3.7. Створення дизайну сторінки «Нещодавні» ..... 59
3.8. Створення дизайну сторінки «Вибрані» ..... 60
ВИСНОВОК ДО РОЗДІЛУ 3 ..... 61
РОЗДІЛ 4. ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-ДОДАТКУ ..... 62
4.1. Підготовка до розробки додатку ..... 62
4.2. Файлова структура проекту ..... 62
4.3. Проектування та розробка БД ..... 65

4.4. Розробка проекту ..... 67
4.4.1. Створення проекту ..... 67
4.4.2. Базові файли ..... 67
4.4.3. Налаштування роутингу ..... 69
4.4.4. Створення сховища стану ..... 71
4.4.5. Створення навігації ..... 73
4.4.6. Створення компоненту MainComponent ..... 74
4.4.7. Створення компоненту Головної сторінки ..... 75
4.4.1. Створення компоненту Плеєру ..... 77
4.5. Тестування ..... 78
4.5.1. Основні види тестування сайтів ..... 78
4.5.2. Функціональне тестування ..... 79
4.5.3. Тестування додатку на сумісність ..... 79
4.5.4. Тестування веб-ресурсів: юзабіліті ..... 80
ВИСНОВОК ДО РОЗДІЛУ 4 ..... 81
ВИСНОВКИ ..... 82
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ..... 84
ДОДАТКИ ..... 86
Додаток А ..... 86
Додаток Б ..... 93
Додаток В ..... 94

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

| | | |
|---|---|---|
| VS Code *(Microsoft Visual Studio Code)* | – | Середовище програмування |
| IDE *(Integrated Drive Electronics)* | – | Інтегроване середовище розробки |
| HTML *(HyperText Markup Language)* | – | Мова розмітки гіпертексту |
| CSS *(Cascading Style Sheets)* | – | Каскадні таблиці стилів |
| SCSS | – | Скриптова метамова, яка інтерпретується в каскадні таблиці стилів (CSS) |
| XML *(Extensible Markup Language)* | – | Розширювана мова розмітки |
| JS *(JavaScript)* | – | Мова програмування |
| SPA *(Single Page Application)* | – | Односторінковий додаток |
| UI *(User Interface)* | – | Користувацький інтерфейс |
| UX *(User Experience)* | – | Досвід користувача |
| API *(Application Programming Interface)* | – | Опис способів взаємодії одного комп’ютера з іншим |
| БД | – | База даних |
| СУБД | – | Система управління базами даних |

# ВСТУП

В сучасному світі музика є невід'ємною частиною життя майже кожної людини. Люди слухають музику, щоб підняти настрій, замінити тишу, розслабитися, сконцентруватися на роботі, та щоб просто насолодитися нею.

Як і інші сфери діяльності людства, музика продовжує розвиватися і тісно сплітається із сучасними технологіями. Декілька сторічь назад музику було можливо послухати лише вживу. Століття назад домінуючим засобом прослуховування музики стало радіо. Епоха музичних носіїв, а можливо, і музичних альбомів відходить у минуле. На зміну їй прийшла музика як потік та як ефір. З переходом від аналогових пристроїв на цифрові, музика стала набагато доступнішою і тому, ще більш популярною. На сьогоднішній день існує вже досить багато різних розважально-музичних ресурсів у мережі, таких як *YouTube Music, Apple Music, Spotify*, але на території українських доменів, багатофункціональних та вартих уваги практично не існує.

Щодо музики, вона оточує нас усюди: можна натрапити у стрічці Facebook на альбом, викладений на *Apple Music, Zvooq* або *Spotify*. Можна увімкнути улюблену пісню на *YouTube* і потім півгодини слухати related tracks, підібрані алгоритмом. Можна використовувати чат-ботів в Телеграм. Або слухати все поспіль на *Soundcloud Boiler Room*. Але ще ж є онлайн-радіостанції, саундтрек до нещодавно переглянутого фільму, треки, «зашазамлені» вранці в кав'ярні... Характерно, що майже всі ці варіанти являють собою стрімінг. В епоху вільного контенту музика не померла, але, здається, розчинилася і заповнила все довкола, наче ефір.

**Актуальність** теми кваліфікаційної роботи «Браузерний музичний плеєр на базі фреймворку *Vue.js*» обумовлена надзвичайною популярністю музичних сервісів серед людства та відсутністю українських веб-додатків у даній сфері. На сьогоднішній день музичні сервіси є найважливішим засобом релаксації, що забезпечує музичними композиціями багато людей і надають послуги, які важко уявити без застосування веб-додатків. Люди з усього світу

звикли до прослуховування музики в інтернеті, тому створення сервісу для прослуховування музики є актуальним.

В рамках програмної реалізації кваліфікаційної роботи було поставлено завдання розробити музичний плеєр для браузерів на основі фреймворку *Vue.js.*

**Метою кваліфікаційної роботи** є отримання готової програмної реалізації браузерного музичного плеєру, з використанням сучасних технологій та інструментів розробки веб-додатків.

Відповідно до поставленої мети роботи визначено основні **завдання виконання кваліфікаційної роботи**:

1. Провести аналіз предметної області.
2. Розглянути та проаналізувати існуючі аналогічні веб-додатки.
3. Розглянути особливості сучасних засобів по розробці web-додатків, виділити їх особливості, недоліки та переваги.
4. Розробити користувацький інтерфейс для web-додатку.
5. Розробити браузерний музичний плеєр.
6. Виконати перевірку працездатності web- додатку.

**Об’єктом дослідження** є проектування та розробка веб-додатку музичного плеєру.

**Предметом дослідження** є процес розробки веб-додатків.

Для досягнення поставленої мети й виконання завдань застосовано сучасні технології та засоби розробки веб-застосунків, зокрема стек *HTML/CSS/JS*, фреймворк *Vue.js,* бібліотеки *VueX* та *VueRouter*, редактор коду *Microsoft Visual Studio Code*, графічний редактор *Figma*, система контролю версій *Git*. Веб-додаток містить головну сторінку, на якій відображено основний функціонал додатку, сторінку з користувацькими плейлистами, окремі сторінки для певних виконавців і їх робіт.

Результати випускної кваліфікаційної роботи рекомендується використовувати для подальшого розвитку, додання та розширення функціоналу web-додатку.

# РОЗДІЛ 1
# АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1. Розвиток цифрової музики

Музика (від грец. μουσική - мистецтво муз) – мистецтво організації музичних звуків, насамперед у часовій (ритмічній), звуковисотній та тембровій шкалі. Музичним може бути практично будь-який звук з певними акустичними характеристиками, які відповідають естетиці тієї чи іншої епохи, та може бути відтвореним. Джерелами такого звуку можуть бути: людський голос, музичні інструменти, електричні генератори тощо. [3]

Як і будь-який вид мистецтва, музика розвивається і тісно сплітається з іншими областями людської діяльності (у тому числі і з інтернетом).

### 1.1.1. Початок цифрового звукозапису

Зародження цифрового звуку з якого пізніше і вийшла музика в цифровому форматі, почалося з того, що в 1928 Гаррі Найквіст визначив необхідну смугу лінії зв’язку для передачі імпульсного сигналу.

У 1963 році компанія *Philips* представила компакт касету, яка використовувала для запису магнітну стрічку, згодом цей носій аудіо став неймовірно популярним через свою простоту і дешевизну, проте згодом його витіснили приблизно на початку 2000-х років.

У 1967 році технічним інститутом досліджень *NHK* представлений перший цифровий котушковий стереорекордер на дюймовій відеострічці. Розроблений пристрій дав старт розвитку пристроїв для запису цифрового звуку.

Пізніше в 1969 році компанія Sony представила світові 13-бітний цифровий стереорекордер, запис в даному пристрої вівся вже на 2-дюймову стрічку, що дозволило збільшити тривалість і, що не м енш важливо, якість запису.

Через вісім років на аудіовиставці *Mitsubishi*, компанії *Sony* та *Hitachi* представляють прототип цифрових аудіодисків, а також грамплатівок.

У 1979 році компанією *Philips* був представлений прототип компакт-диска для аудіозапису, пізніше були невеликі розбіжності з компанією Sony через частоту запису.

У 1982 році в Японії та Європі було прийнято стандарт на систему компакт-дисків. У цей же рік компанією *Philips* був представлений перший програвач для компакт-дисків - пристрій був не схожим на попередні і був революційним (рис. 1.1.).

Рис. 1.1. Програвач Philips для дисків

В період з 1990-х диски лише набирали популярності, але компакт-касети все ще утримували лідерство за рахунок популярності та дешевизни. До СРСР компакт диски дісталися тільки в 1989 році.

У 1995 році був розроблений такий формат стиснення аудіоданих як *MPEG* 1, *Audio Layer* 3 або *MP3*. Формат дозволяв збільшити тривалість

аудіозапису який містився на диск, проте місце як на дисках так і на персональних комп'ютерах, як і раніше, не вистачало і обмінюватися музикою простим обивателям вдавалося важко.

У 1997 році було створено програмний плеєр *Winamp*, який дозволяв прослуховувати музичні файли на персональному комп'ютері. Можна сказати, що на цьому моменті епоха аналогової музики закінчилась.

### 1.1.2. Початок 2000-х. Музика підкорює Інтернет-простір

У 1999 році був запущений *Napster*. Він використовував протокол peer-to-peer (який згодом став основою для протоколу *bit*-торрент) для обміну файлами. Сервіс вплинув на людей, які використовували Інтернет і дійсно захоплювалися музикою. *Napster* дозволяв вільно і водночас безкоштовно обмінюватися музикою, і це створило певні проблеми для сервісу у вигляді звинувачень у порушенні авторських прав.

У 2003 році компанія *Apple* представила світові легальний спосіб прослуховувати музику через інтернет у вигляді сервісу *iTunes Store*, база композицій в онлайн-магазині на момент презентації складала понад 200 000 треків.

Наступним сервісом який дозволяв легально і водночас безкоштовно прослуховувати музику, аудіокниги та підкасти став шведський сервіс *Spotify*, який був запущений у 2006 році [4].

### 1.1.3. Сучасна ситуація з музикою

У наш час завдяки високошвидкісному Інтернету знайти будь-яку музику дуже легко та зручно. Для цього потрібно лише відвідати спеціальні сайти. Вони широко представлені численними музичними порталами – сховищами музичних композицій. На багатьох можна скачати безкоштовно музику, деякі пропонують аудіо продукцію за гроші. Разом з розвитком інтернету друге життя отримало радіо, зараз воно доступне майже на будь-якому інформаційному та розважальному ресурсі.

Багато сайтів та соціальних мереж дозволяють слухати музику онлайн, складати свої плей-листи та альбоми. Причому всі музичні напрямки будуть представлені, і всякий меломан знайде собі те, що до вподоби. Класична музика, джаз, рок, сучасна поп-музика та всі інші жанри представлені у глобальній мережі. Інтернет-користувачі можуть легко та швидко побачити розвиток музичного мистецтва у минулому та основні напрямки в сучасності. Музика стала доступнішою, вона перестала бути розвагою лише обраних.

## 1.2. Поняття і принципи роботи web-додатків

Web-додатки зазвичай розробляються за допомогою мов програмування, які підтримуються браузерами. Деякі сторінки web-застосунків є динамічними і потребують обробки на стороні сервера. Інші повністю статичні і не вимагають таких дій.

### 1.2.1. Відмінність понять «Web-додаток» та «Web-сайт»

Web-сайт – це сукупність вебсторінок та залежного вмісту, доступних у мережі Інтернет, які об’єднані як за змістом, так і за навігацією під’єдиним доменним ім’ям. Фізично сайт може розміщуватися як на одному, так і на кількох серверах.

Web-додаток – це будь-яка комп’ютерна програма, яка виконує певну функцію за допомогою веб-браузера як клієнта, незалежно від пристрою чи платформи, на якій відкрито браузер. Програма може бути простою, як дошка оголошень або контактна форма, текстовий процесор або програма для мобільних ігор для кількох гравців, яку ви завантажуєте на свій телефон.

Основною відмінністю даних понять є різні рівні взаємодії зі сторінкою. Хоча веб-сайт і містить текстовий і візуальний вміст, до якого користувач не може отримати доступ, веб-додатки дозволяють користувачеві не тільки читати, але й змінювати та додавати інформацію на сторінки.

### 1.2.2. Принцип роботи web-програми

Web-програма потребує, щоб web-сервер керував запитами клієнта, сервер додатків виконував функціональну логіку відповідно до запиту користувача, включаючи обробку та збереження інформації в базі даних.

Типовий принцип роботи web-програми виглядає наступним чином:

1. Користувач генерує запит до web-сервера за допомогою web-браузеру або через інтерфейс користувача програми.
2. Web-сервер пересилає цей запит на відповідний сервер web-додатку.
3. Сервер web-додатку виконує необхідну логіку відповідно до отриманого запиту, наприклад, виконує запит до бази даних або обробку даних, а потім генерує результат роботи.
4. Сервер web-додатку надсилає результати на web-сервер із необхідною інформацією або обробленими даними.
5. Web-сервер надає відповідь клієнту із необхідною інформацією, яка потім з’являється в інтерфейсі користувача.

## 1.3. Найпопулярніші web-сервіси для прослуховування музики та їх аналіз

### 1.3.1. Глобальна статистика

Згідно зі звітом дослідницької компанії *MIDiA*, станом на 2021 рік 523,9 мільйона людей у всьому світі підписалися на стрімінгові потоки музики (рис. 1.2.). Це на 109,5 мільйона (26,4%) більше, ніж у попередньому році.

Хоча *Spotify* залишається стрімінговим сервісом із найбільшою кількістю підписників, володіючи 31% ринку, його домінування дуже повільно зменшується — у 2020 році він займав 33% ринку та 34% у 2019. Тим часом *YouTube Music* від *Google* зріс більш ніж на 50% за рік до другого кварталу 2021 року, що зробило його єдиним західним стрімером, який

збільшив свою загальну частку на світовому ринку. Китайський ігровий гігант *Tencent*, який керує *Tencent Music*, має таку ж частку ринку, як і *Amazon Music*.

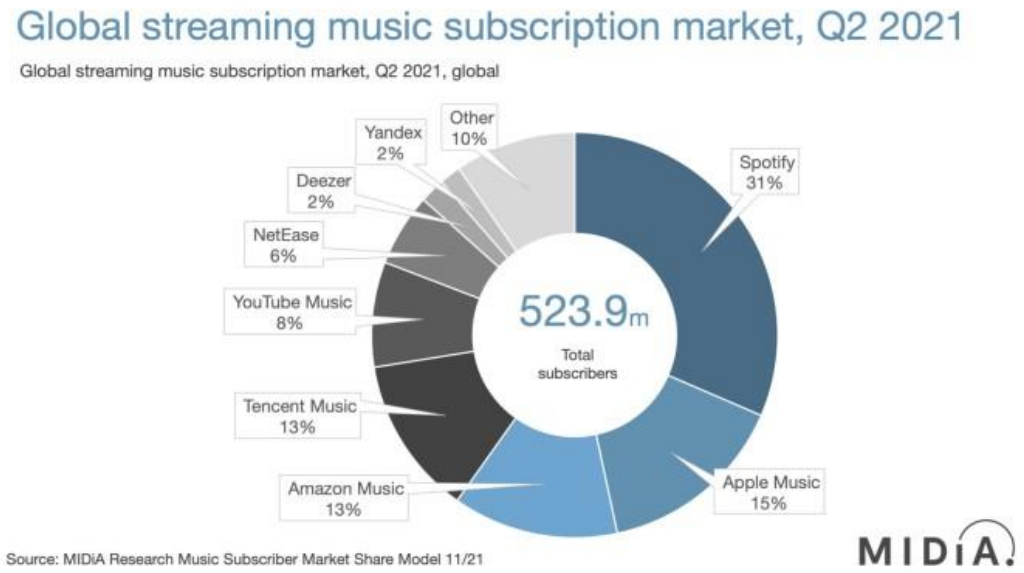



Рис.1.2. Статистика світових стрімінгових сервісів

### 1.3.2. Огляд найпопулярніших сервісів

1. Spotify

Шведський сервіс, який давно займає місце найпопулярнішого у світі музичного стрімінгового сервісу, що значно обганяє конкурентів за багатьма показниками. Цей сервіс – справжня музична соцмережа: тут можна додавати друзів, обмінюватися повідомленнями та коментувати музичні новини. На цій платформі вам доступний величезний каталог пісень, зручний інтерфейс та система пошуку, а також рекомендації, побудовані на уподобаннях користувача. Підтримує майже всі пристрої для відтворення треків, на яких можна налаштувати синхронізацію своєї медіатеки та операційні системи.

Переваги:

– велика медіатека – до 50 млн пісень;

– точні алгоритми для вибору треків;

– можливість користуватися сервісом безкоштовно за певних обмежень;

– тривалий пробний період;

– є режим для бігунів;

– є автомобільний режим;

– високий бітрейт;

– синхронізація відтворення на всіх пристроях.

Недоліки:

– тексти є лише окремих композицій;

– можливе збереження трохи більше 10 тисяч треків;

– відсутність підкастів.

Інтерфейс головної сторінки сервісу Spotify представлено на рис. 1.3.

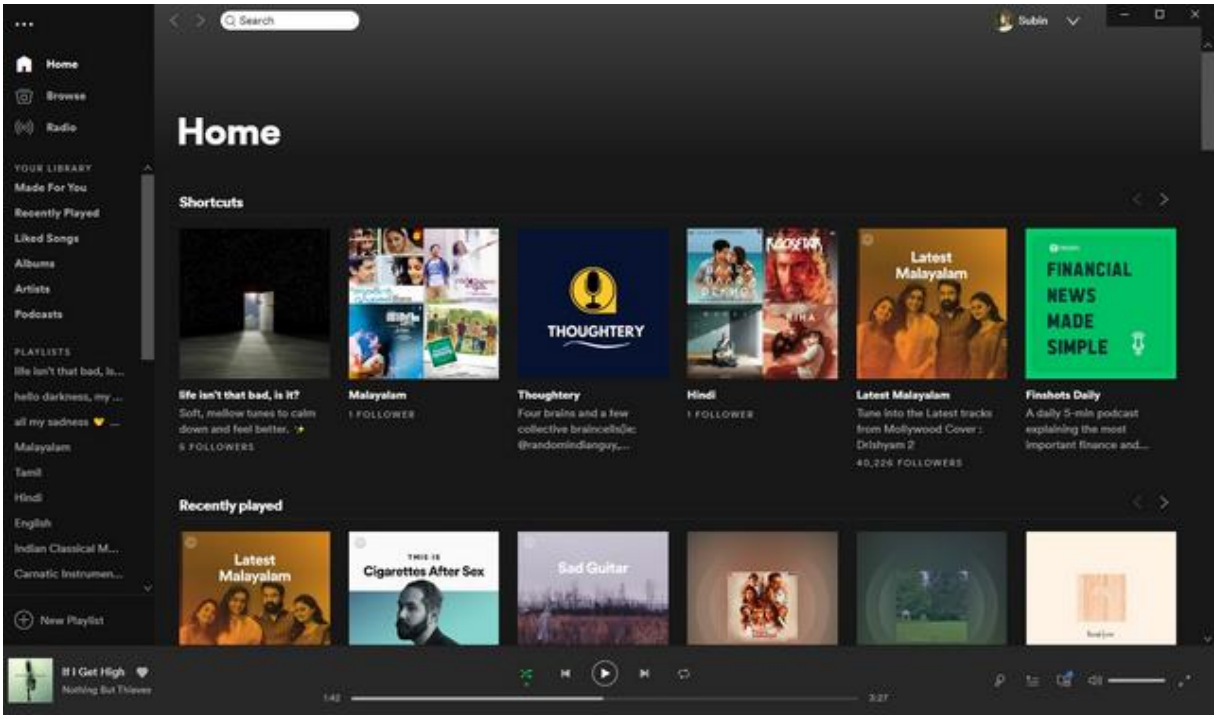



Рис. 1.3. Інтерфейс Spotify

2. Apple Music

Другий за популярністю у світі музичний стрімінговий сервіс після *Spotify*. Має велику базу в 60 мільйонів треків. *Apple Music* підтримує функцію складання власних плейлистів, якими можна ділитися з друзями за посиланням. Також сервіс має функцію підбору плейлистів на треків на основі того, що слухає та лайкає користувач.

Переваги:

– синхронізовані тексти пісень;

– інтеграція з екосистемою *Apple*;

– високий бітрейт (256 Кбіт/с, AAC);

– багато дитячих пісень іноземними мовами.

Недоліки:

– мало налаштувань;

– персональні рекомендації погано працюють у разі рідкісних жанрів;

– веб-версія недостатньо привітна.

Користувацький інтерфейс *Apple Music* на різних девайсах надано на рис. 1.4.

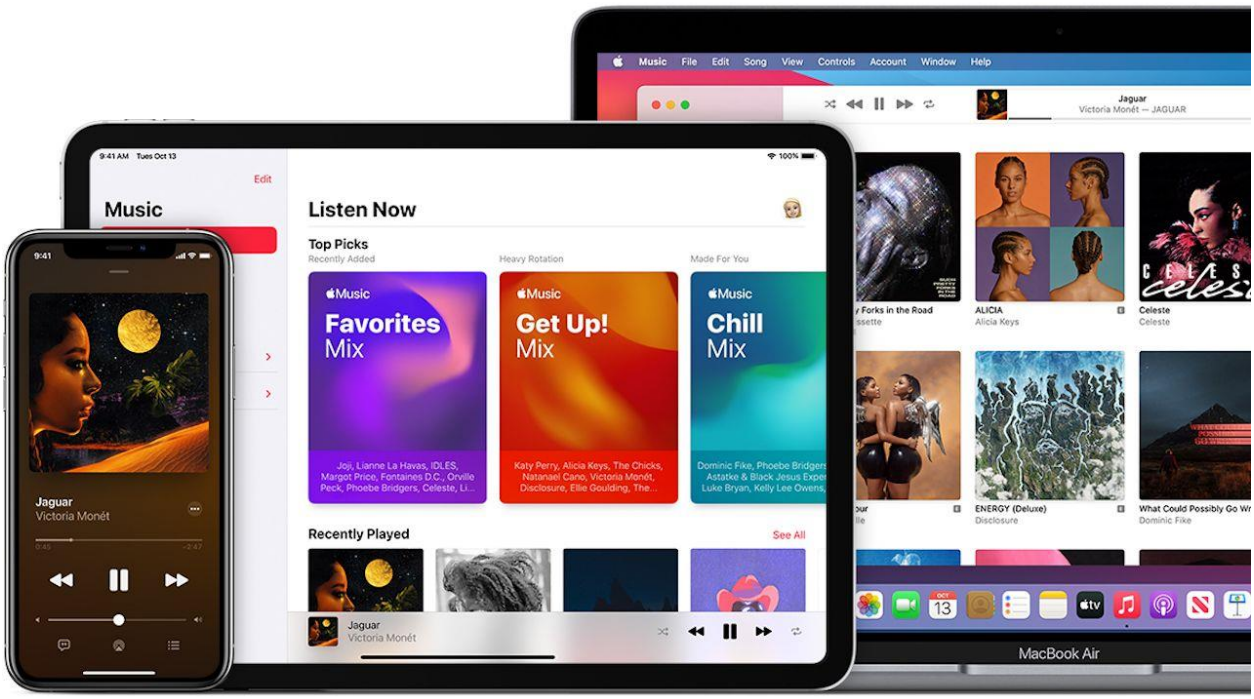



Рис. 1.4. Користувацький інтерфейс Apple Music на різних пристроях

3. YouTube Music

Стрімінговий сервіс, який раніше називався *Google Music*, але тепер об’єднався з платформою *YouTube*, тому в ньому можна не тільки слухати музику, а й дивитися відеокліпи з бази хостинг.

Переваги:

– є доступ не лише до великої бібліотеки музики, а й до відеокліпів;

– у підписку *YouTube Music* входить і *YouTube Premium*;

– можливість завантажувати відео та аудіо;

– супермікси;

– зручна система пошуку.

Недоліки:

– будь-який трек або відеокліп може зникнути з бази у разі порушення авторських прав;

– алгоритм добірки пісень не вдосконалено;

– різна якість звуку від треку до треку.

Вигляд додатку *YouTube Music* представлений на рис. 1.5.

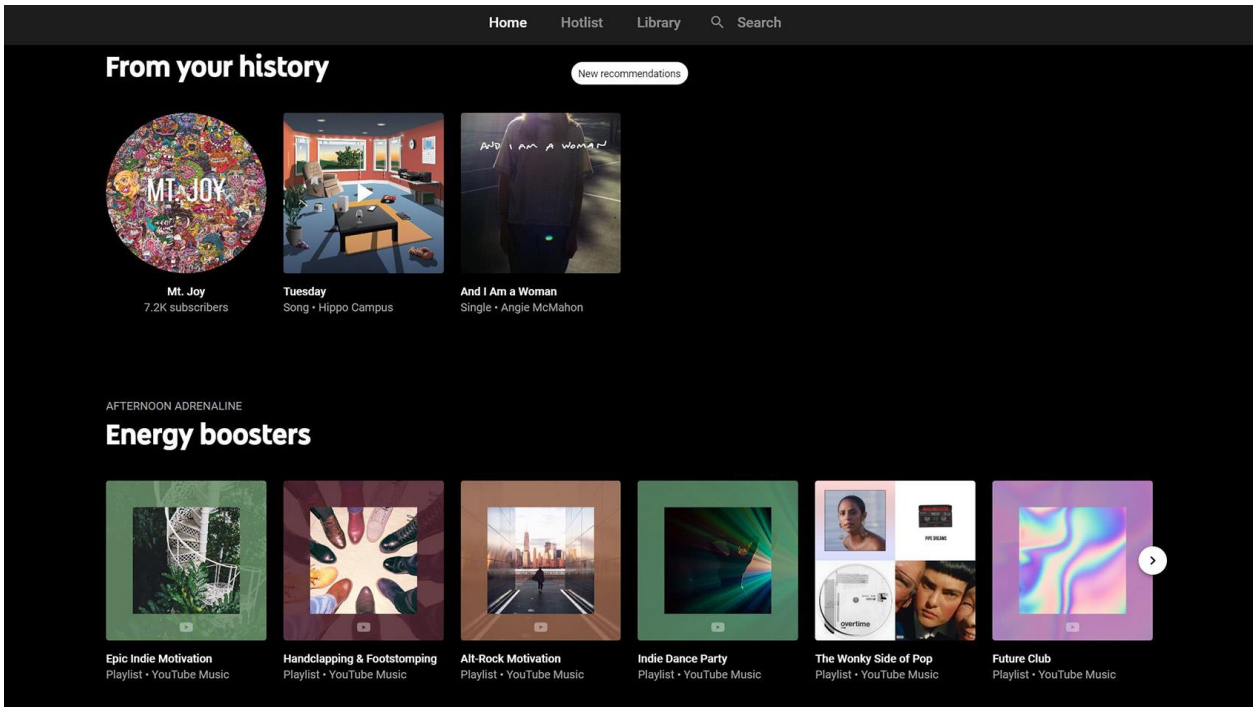



Рис. 1.5. Інтерфейс YouTube Music

4. Deezer

Музичний сервіс *Deezer* має бібліотеку не більше, ніж на *Apple Music*, але все одно досить значну — понад 56 млн. треків. Ви знайдете тут вибірки на будь-який смак: від тематичних до плейлистів, підібраних під ваш настрій

та на основі того, що ви слухали до цього (функція *Flow*). Також вам рекомендують нові композиції.

Переваги:

– програвання пісень у lossless-форматі;
– функція "караоке";
– авторські подкасти;
– інтеграція з соцмережами *Facebook, Twitter* та *Last.fm*;
– низьке споживання трафіку;
– кураторські та користувацькі плейлисти;
– робота на всіх пристроях та ОС.

Недоліки:

– можна завантажити не більше 700 пісень.

Користувацький інтерфейс сервісу *Deezer* представлений нижче на рис. 1.5.

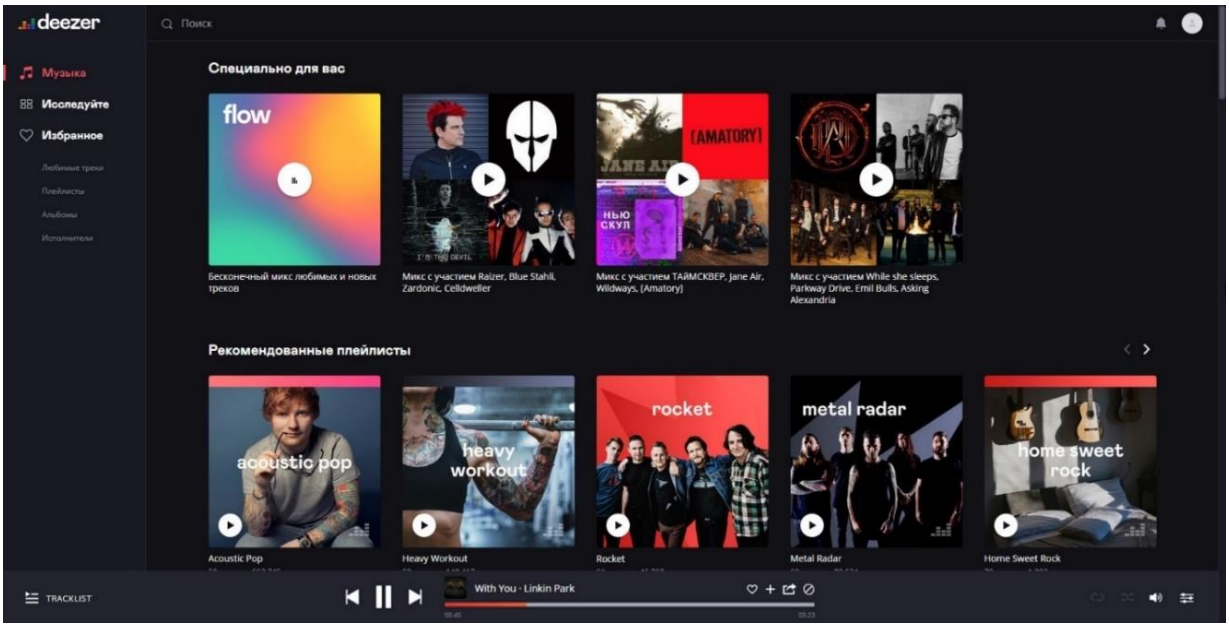



Рис. 1.6. Інтерфейс Deezer

### 1.3.3. Порівняльна таблиця сервісів для прослуховування музики

Для порівняльного аналізу було зведено дані про найбільш популярні стрімінгові сервіси в одну єдину таблицю (табл. 1.1).

Таблиця 1.1

Порівняльна таблиця стрімінгових сервісів

| | Spotify | Apple Music | Deezer | YouTube Music |
|---|---|---|---|---|
| Збереження та прослуховування музики в офлайн-режимі | З підпискою | Є | З підпискою | З підпискою |
| Безкоштовна версія (коротко які обмеження) | Є:<br>–реклама;<br>–відсутність офлайн-режиму;<br>–немає максимальної якості музики;<br>пропуск не більше 6 треків на годину. | Немає | Є:<br>- реклама;<br>- відсутність офлайн-режиму;<br>- немає аудіо у високій якості;<br>- обмеження пропуску треків;<br>можна відтворити не всі пісні. | Є:<br>–реклама;<br>–немає відтворення при закритому додатку;<br>–відсутність офлайн-режиму;<br>не можна перемішувати треки. |
| Підтримка текстів пісень | Лише окремих композицій | Є | Є | Є |
| Можливість завантажувати свою музику | З підпискою | Є | Є | З підпискою |

Продовження таблиці 1.1

| | Spotify | Apple Music | Deezer | YouTube Music |
|---|---|---|---|---|
| Формування плейлистів | Є | Є | Є | Є |
| Сервіс рекомендованих треків | Є | Є | Є | Є |
| Радіо | Є | Є | Ні | Є |
| Бітрейт | 320 Кбіт/с (Ogg Vorbis) при платній передплаті; до 160 Кбіт/с за безкоштовної | 256 Кбіт/с (AAC) | 320 Кбіт/с, Hi-Fi: 1411 Кбіт/с, (44.1 кГц/16 біт) | 256 Кбіт/с (AAC) |

### 1.3.4. Результати проведеного аналізу додатків-аналогів

В результаті пошуку додатків-аналогів, а також проведення їх аналізу, вдалося виявити основну проблему – неможливість знайти аналогічні веб-додатки від українських розробників, та й взагалі аналогічні веб-додатки меншого формату. Усі розглянуті вище аналоги – це стрімінгові сервіси від компанії-гігантів, що неможливо назвати повноцінними аналогами для додатку, що розробляється в ході виконання кваліфікаційної роботи.

## 1.4. Постановка задачі розробки web-додатку

Метою кваліфікаційної роботи є отримання готової програмної реалізації браузерного музичного плеєру, використовуючи сучасні технології та інструменти веб-розробки.

*Назва роботи*

Браузерний музичний плеєр на базі фреймворку *Vue.js*.

*Призначення та цілі системи*

Система, що розробляється, призначена для задоволення інформаційних потреб користувачів Інтернету.

Для досягнення мети проекту необхідно розробити web-додаток, що відповідає наступним вимогам.

*Вимоги до дизайну*

Користувацький інтерфейс web-додатку повинен відповідати таким вимогам [5]:

– дизайн має слідувати основним тенденціям сучасного *UI/UX* дизайну [6];

– основна тема web-додатку – темна, але повинна бути можливість перемикати її;

– кросбраузерність – сайт повинен коректно відображатися усіх найбільш популярних браузерах, таких як *Opera, MozillaFirefox, Google Chrome, Safari*, а також у мобільних браузерах

– інтуїтивне меню – навігація має бути максимально зручною для користувача;

– responsive дизайн – можливість відображення на пристроях з будь-якою роздільною здатністю;

– простота – дизайн має бути максимально простим та мінімалістичним, без зайвих елементів та великого набору кольорів;

– основні функції застосунку мають бути доступні з головної сторінки.

*Вимоги до системи загалом*

– web-сервіс повинен містити зручну систему навігації у вигляді меню з пунктами та підпунктами;

– навігаційне меню повинно відображатися на всіх сторінках;

– не допускається наявність багів та неточностей роботи web-застосунку.

Розділи сайту:

– головна – сторінка, на якій розміщені банер з новинками, а також розділ з рекомендаціями;

– плейлисти – сторінка, на якій відображається список усіх плейлистів користувача;

– альбоми;

– нещодавні – сторінка з останніми прослуханими треками;

– вибрані – сторінка з улюбленими композиціями;

– локальні – сторінка, на якій відображається локальний плейлист.

*Етапи розробки web-додатку*

Створення будь-якого web-додатку складається з наступних етапів розробки:

– формування цілей та завдань web-застосунку;

– проведення досліджень, аналіз додатків-аналогів у мережі Інтернет;

– формування технічного завдання на розробку сайту;

– збір необхідних теоретичних матеріалів;

– розробка дизайну

– розробка функціональності сторінок, а також програмних рішень;

– робота з наповненням сайту;

– тестування;

– виправлення багів та запуск продукту.

# ВИСНОВОК ДО РОЗДІЛУ 1

Перший розділ випускної кваліфікаційної роботи демонструє аналітичну частину та аналіз предметної області.

В ході виконання роботи було розглянуто історію розвитку музичного мистецтва та культури, розглянуто поняття web-сайту та web-додатку, проведено теоретичний аналіз предметної області кваліфікаційної роботи, а також зроблено огляд існуючих додатків-аналогів, зокрема виділено їх особливості, недоліки та переваги і проведено порівняльний аналіз.

Огляд аналогів показав, що перш за все необхідно звернути увагу на зручність і зрозумілість користувацького інтерфейсу, адаптивність дизайну та простоту функціоналу.

Результатом аналізу стали визначені основні вимоги до web-додатку, що розробляється, проведена постановка задачі розробки, виділені вимоги докористувацького дизайну і до системи в загальному випадку, а також виділені подальші етапи розробки.

Були виділені майбутні розділи сайту і їх наповнення: головна сторінка, плейлисти, альбоми, нещодавн, вибрані, локальні.

Усі дані, отримані на основі аналізу, дозволять якісно та коректно спроектувати і реалізувати музичний web-додаток.

# РОЗДІЛ 2
# АНАЛІЗ ТА ПОРІВНЯННЯ ОБРАНИХ ІНСТРУМЕНІВ І ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ WEB-ДОДАТКУ

При виконанні проектування роботи web-додатку були враховані загальноприйняті методи та стандарти дослідження.

Теоретичний метод дослідження (метод аналізу) складається з вивчення та аналізу предметної області та був використаний задля правильної постановки задач проектування та подальшої реалізації. Метод аналізу дозволив дослідити проблемні місця та мінуси web-сайтів та web-додатків – аналогів, що будуть уникненні в подальшій реалізації.

Після проведення аналізу предметної області, постановки завдань та вимог, було прийнято рішення про використання наступних технології:

- *VS Code;*
- *Figma;*
- *HTML ma CSS (SCSS);*
- *JavaScript;*
- *Vue.js;*
- *VueX;*
- *Vue Router;*
- *Git.*

## 2.1. Середовище розробки VS Code

Вибір зручного і функціонального середовища розробки є важливою частиною роботи будь-якого розробника і впливає на комфортність розробки в цілому.

Для досягнення мети кваліфікаційної роботи було вирішено використовувати *Microsoft Visual Studio Code*.

Інтерфейс редактору коду *Microsoft Visual Studio Code* представлений на рис. 2.1.

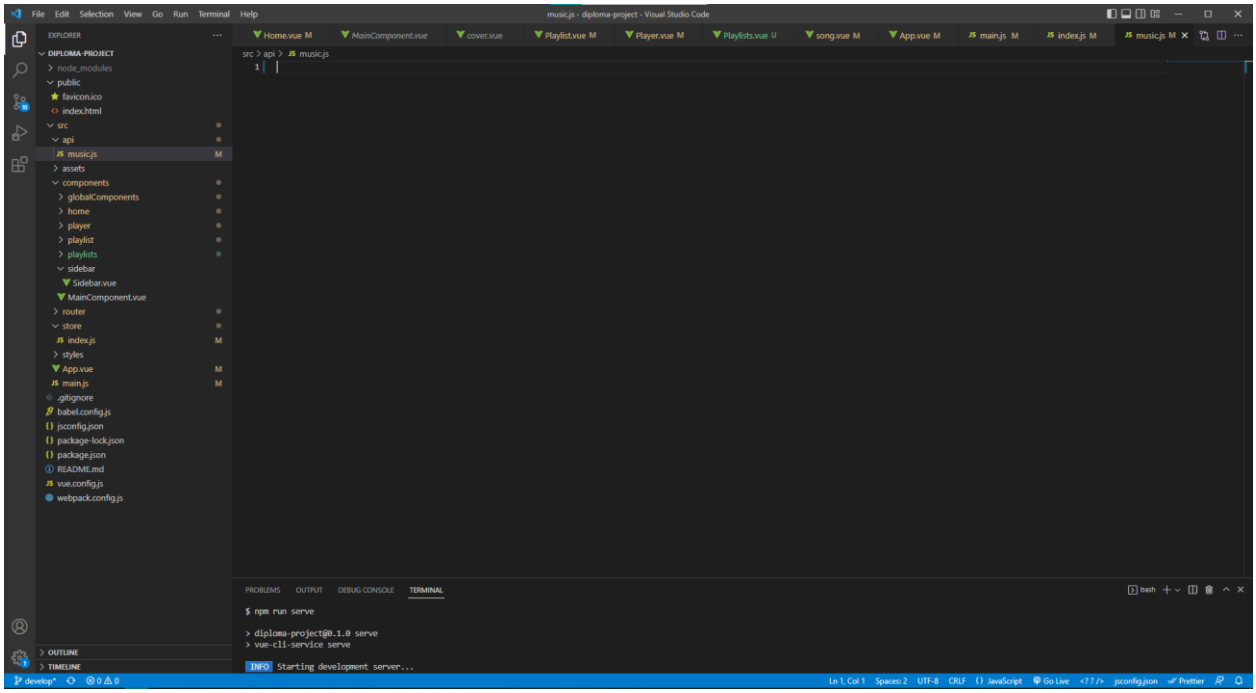



Рис. 2.1. Інтерфейс VS Code

*Microsoft Visual Studio Code (VS Code)* – це редактор вихідного коду, розроблений *Microsoft* для *Windows*, *Linux* та *MacOS*. Позиціонується як «легкий» редактор коду для кросплатформної розробки веб- та хмарних програм. Включає в себе відладчик, інструменти для роботи з *Git*, підсвічування синтаксису, *IntelliSense* та засоби для рефакторингу коду. Має широкі можливості для кастомізації: теми користувача, поєднання клавіш і файли конфігурації. Розповсюджується безкоштовно, розробляється як програмне забезпечення з відкритим вихідним кодом, але готові зборки розповсюджуються під ліцензією проприетарної.

*Visual Studio Code* заснований на *Electron* та реалізується через веб-редактор *Monaco*, розроблений для *Visual Studio Online*. Дуже зручною частиною редактора є інтерактивне підсвічування коду, що дозволяє

підтримувати зорову увагу на високому рівні і не викликає візуального стомлення. Також це дозволяє зручно бачити межі коду для його легкого написання та рефакторингу і масштабування.

Варто зазначити, що це не інтегроване середовище розробки, а звичайний редактор коду. Різниця полягає в тому, що в редакторі коду менше функціоналу порівняно з *IDE*, але це не значить, що немає потрібних функцій для розробки, тому що, за допомогою так званих плагінів і розширень, кількість і функціонал яких дуже розширений, тож можна налаштувати редактор коду, так як заманеться будь-якому розробнику.

Середовище розробки Visual Studio Code підтримує основні мови програмування і технології, одні з яких *JavaScript, C#, C++, TypeScript, PHP, Python, HTML, CSS, SASS, React, Vue.js* та інші. З вищевказаних обраних технологій для розробки додатку, можна зробити висновок, шо *Visual Studio Code* покриває усі вимоги.

## 2.2. Дизайн

### 2.2.1. Figma

*Figma* – це векторний онлайн-сервіс розробки інтерфейсів та прототипування з можливістю організації спільної роботи в режимі реального часу. Розробники позиціонують даний продукт як основний конкурент програмним продуктам компанії *Adobe. Figma*, має інтуїтивно зрозумілий і простий у використанні інтерфейс. При цьому можна працювати як в браузері, так і в додатку.

Особливістю програми є можливість спільної роботи декількох людей одночасно, причому всі користувачі можуть відслідковувати всі зміни в реальному часі. На екрані робочої області можна побачити курсори інших людей, які наразі працюють з цим документом, а в історії операцій позначено, які зміни додав той чи інший користувач. Також є можливість створення залежних копій оригіналу елемента. Всі зміни в елементі-оригіналі відображаються в елементах-копіях, при цьому зміни в копіях ніяк не

впливають на оригінал або інші копії. Це значно спрощує роботу над веб-інтерфейсом і економить час, так як при внесенні правок не потрібно міняти кожен елемент, досить змінити оригінал.

Застосунок є умовно безкоштовним. Платний тариф дає можливість працювати над будь-якою кількістю проектів, а також надає можливість використовувати додаткові інструменти для командної роботи, системну аналітику, доступ до спеціальних плагінів і поліпшену безпеку.

## 2.3. HTML, CSS, мова програмування JavaScript

### 2.3.1. HTML

*HTML* – це мова опису структури сторінок документів, яка дозволяє звичайний текст форматувати в абзаци, заголовки, списки та інші структури, створювати посилання на інші сторінки. Це текстова мова, в якій інструкції з форматування, що називаються тегами, вбудовані в розділи документа, які містять конкретну інформацію. Теги повідомляють браузерам, як форматувати і представляти інформацію на екрані.

Мова гіпертекстової розмітки *HTML* була запропонована Тімом Бернерсом-Лі у 1989 як один з компонентів технології розробки розподіленої гіпертекстової системи *World Wide Web*. Ідея гіпертекстової інформаційної системи полягає у тому, що користувач має можливість переглядати веб-сторінки у найбільш зручному для себе порядку, а не послідовно, як це прийнято при читанні книг. Досягається це шляхом створення спеціального механізму пов’язування різних сторінок тексту за допомогою гіпертекстових посилань.

Мова *HTML* дозволяє визначити структуру електронного документа із поліграфічним рівнем оформлення. Результуючий документ може містити різні елементи: ілюстрації, аудіо та відео фрагменти. Мова *HTML* включає розвинені засоби для визначення кількох рівнів заголовків, шрифтових виділень, різних груп об’єктів та багатьох інших можливостей.

Загалом, можна визначити такі функції в *HTML*:

– створення документу, який буде структурований за допомогою тегів: зображення, заголовки, таблиці, абзаци тощо;

– отримання інформації через гіперпосилання;

– створення різноманітних інтерактивних форм;

– можливість додавання зображення, відео до тексту.

За основу моделі розмітки документів у *HTML* прийнята тегова модель. Тегова модель описує документ як сукупність контейнерів, кожен з яких починається і закінчується тегами. Тобто документ *HTML* є не чим іншим, як звичайним *ASCII*-файлом з доданими до нього керуючими *HTML*-кодами (тегами).

Теги *HTML*-документів в основному є простими і зрозумілими для використання, оскільки вони створені за допомогою загальновживаних слів англійської мови, зрозумілих скорочень і позначень.

Створення *HTML*-документа відбувається розташуванням тегів (*tags*) мови *HTML* всередині звичайного неформатованого тексту. Теги *HTML* — це послідовності символів, які починаються знаком «менше» (<) і закінчуються знаком «більше» (>). Теги можуть мати атрибути, які, в свою чергу, можуть приймати певні значення. [10]

Приклад *HTML*-коду наведено на рис 2.2.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="UTF-8">
5      <title>Title goes here</title>
6    </head>
7    <body>
8
9    </body>
10 </html>
```

Рис.2.2. Приклад HTML-коду

### 2.3.2. CSS

*CSS* – це мова каскадних таблиць стилів, яка використовується для опису стилів багаторазового використання для подання документів, написаних мовою розмітки *HTML* або *XML*.

*CSS* надає можливість створювати правила, які легко змінювати, редагувати и застосовувати до усіх визначених нами елементів. Кожне правило, складається з двох частин. У лівій частині міститься селектор (*selector*), а у правій блок визначення (*declaration block*). Блок визначення складається з набору властивостей (*property*) та їх значень (*value*).

Приклад написаних стилів за допомогою *CSS* зображений на рис 2.3.

```
.nav {
  background-color: #4F4D53;
  height: 48px;
  width: 100%
}

.logo {
  position: relative;
  left: 25%;
  padding-top: 10px
}
```

Рис.2.3. Приклад CSS

*CSS* також дозволяє адаптувати контент до різних умов відображення (на екрані монітора, мобільного пристрою, у роздрукованому вигляді, на екрані телевізора, пристроях з підтримкою шрифту Брайля або голосових браузерах та ін.).

Переваги використання *CSS* є наступними:

– розмежування коду і оформлення;
– різне оформлення для різних пристроїв;
– розширені в порівнянні з *HTML* способи оформлення елементів;
– прискорення завантаження сайту;

– єдине стильове оформлення безлічі документів;

– централізоване зберігання [8].

Однак в подальшому веб-додаток, що розробляється, не буде використовувати *CSS* в чистому вигляді, а стилі будуть написані за допомогою препроцесора *SCSS* файли розширення .scss буде інтерпретовано в файли з розширенням .css.

Відмінності синтаксису чистого *CSS* та препроцесору *SCSS* показано на рис. 2.4.


Рис.2.4. Відмінності синтаксису CSS і SCSS

Загалом, буде справедливим назвати препроцесор *SASS* «синтаксичним цукром» [9] по відношенню до *CSS*. Оскільки, *SASS* розширює функціонал *CSS*, додаючи до останнього кілька зручних та необхідних для швидкої розробки, механізмів. Однією з головних переваг препроцесору *SASS* є можливість визначати змінні. Це зручно тоді коли слід змінити колір відразу в декількох місцях під час розробки, і не стає проблемою змінити в одному місці. Також важливою особливістю *SASS* є вкладені селектори. Це значно покращує та пришвидшує розробку, а також зменшує загальну кількість рядків коду в файлах. Також препроцесор *SASS* має можливість створювати міксини, перебор змінних в циклі і успадкування для визначених селекторів.

### 2.2.3. JavaScript

*JavaScript (JS)* – динамічна, об'єктно-орієнтована мова програмування, що найчастіше використовується при створенні сценаріїв поведінки браузера, що вбудовуються у веб-сторінки.

*JavaScript* був створений, щоб «оживити» веб-сторінки. Програми цією мовою називаються скриптами. Їх можна записати прямо в *HTML* веб-сторінки та запустити автоматично під час завантаження сторінки. Сценарії надаються та виконуються як звичайний текст. Для запуску їм не потрібна спеціальна підготовка чи компіляція, що вирізняє цю мову програмування з-поміж інших.

*JavaScript* дозволяє реалізовувати наступні складні функції на веб-сторінках:

– відображення своєчасного оновлення вмісту;

– підключення інтерактивних карт;

– підключення анімації 2D/3D-графіки.

*JS* не забезпечує низькорівневий доступ до пам'яті чи процесора, оскільки будь-який браузер підтримує синтаксис даної мови програмування.

Можливості мови *JavaScript* значною мірою залежать від середовища, в якому він працює. *JavaScript* у браузері виконує наступні дії:

– реагує на дії користувача: рух курсору, натискання клавіш чи кнопок миші;

– може додавати новий *HTML* на сторінку, замінити наявний вміст чи стилі;

– надсилає запити по мережі на віддалені сервери, а також завантажує і вивантажує файли;

– отримає та встановлює файли *cookie*;

– запам'ятовує дані зі сторони клієнта («*local storage*»).

*JavaScript* – це унікальна мова програмування. Є як мінімум три особливості в *JavaScript*:

– цілковита інтеграція з *HTML*/*CSS*;

– прості речі робляться просто;

– підтримується всіма сучасними браузерами за замовчуванням.

*JavaScript* – єдина браузерна технологія, яка суміщає ці три речі. В результаті існування перерахованих вище факторів, JavaScript є найбільш поширеним засобом створення браузерних інтерфейсів.

До слова, *JavaScript* також дозволяє створювати сервери, мобільні застосунки, тощо [11].

## 2.4. Фреймворки у веб-розробці та їх порівняння

Перш ніж перейти безпосередньо до обґрунтованого вибору фреймворку не буде зайвим торкнутися поняття фреймворку, описати найпоширеніші та найпопулярніші фреймворки для web-розробки, а також виділити їх особливості та переваги.

Немає сенсу говорити, що існує якийсь унікальний фреймворк для розробки web-додатків, який найкраще підходить для вирішення будь-яких завдань та проектів. Звісно, потрібно розуміти, що деякі фреймворки працюють краще, ніж інші, і вони є найкращі для конкретного випадку використання.

З огляду на інформацію та статистику було виділено найпоширеніші та найпопулярніші веб-фреймворки: *React, Vue* та *Angular*.

### 2.4.1. Поняття фреймворку

Фреймворк — це платформа, яка визначає структуру веб-програми. Його функціонал дозволяє здійснити взаємодію веб-ресурсу із сервером.

На веб-сайтах, які написані без використання фреймворку, початковий контент зберігається на сервер. Тому при завантаженні нового матеріалу на сайт потрібне перезавантаження сторінки. Перевагою фреймворку є постійні блоки, які зберігаються від однієї конфігурації до іншої. Це дозволяє вимагати миттєвого зворотного зв’язку з користувачем при додаванні нового контенту.

Даний принцип можна спостерігати при створенні односторінкових веб-застосунків, електронної комерції, хмарних сервісів та багатьох соціальних

мереж. Так, перехід користувача з одного розділу до іншого здійснюється миттєво. Перевантаження сторінки не відбувається, оскільки каркас залишається незмінним.

Переваги використання фреймворку:

– фреймворки є повністю безкоштовними та мають відкритий вихідний код;

– використання вбудованих шаблонів допомагає створювати проекти вищої якості, причому задіються менше рядків коду;

– висока швидкість розробки, яка досягається за рахунок відкритого доступу до документації та безлічі форумів.

За реалізацією проектів на класичному *JS* та *HTML* ховається багато труднощів, що призводить до появі великої кількості фреймворків над ринком. Але найпопулярнішими серед веб-розробників залишаються *Angular*, *React* та *Vue*.

### 2.4.2. React

React – це найпопулярніший *JavaScript* фреймворк з відкритим кодом для створення динамічних інтерфейсів користувача, створена та розроблена компанією *Facebook*. Застосовується для створення веб-застосунків з динамічними компонентами. В його основі лежить *JavaScript* та *JSX*. До *React* також входить *React Native*, спеціалізована міжплатформна система для розробки мобільних додатків.

React характеризується простотою в освоєнні, лаконічністю синтаксису, можливістю створення і використання *VirtualDOM*, за допомогою якого розвантажуються високонавантажені застосунки.

*Плюси використання React*

– Віртуальна об’єктна модель документа

Об’єктна модель документа (*DOM*) визначає деревоподібну структуру *HTML*-документа, що надсилається клієнту сервером після відповідного

запиту. *DOM* представляє веб-сторінку в об'єктно-орієнтованому форматі, щоб мови програмування могли взаємодіяти з нею.

Браузер регулярно перевіряє будь-які зміни в *DOM*, оновлюючи її належним чином. Браузер оновлює *DOM* щоразу, коли сторінка змінюється. Оскільки *DOM* сучасних сайтів є величезними, оновлення займає багато часу, уповільнюючи загальну продуктивність веб-додатку.

Замість повільних та незручних взаємодій безпосередньо з реальною об'єктною моделлю документа, *React* взаємодіє з її полегшеною копією — віртуальної *DOM*, тому реальна *DOM* оновлюється лише після взаємодії з віртуальною *DOM*.

Ефективність. *React* зберігає в пам'яті дві версії віртуального *DOM* — оновлений віртуальний *DOM* та його резервну копію, створену до оновлення. Після оновлення *React* порівнює обидві версії між собою, щоб знайти змінені елементи, а потім — оновлює частину реального *DOM*, що виключно змінилася. Подібний процес займає набагато менше часу, ніж оновлення реальної об'єктної моделі документа цілком, отже, він оптимізує роботу з *DOM*.

Висока продуктивність. Віртуальна *DOM* дозволяє сторінці негайно отримувати відповіді від сервера та відображати оновлення. Наприклад, *Facebook* застосовує технологію віртуального *DOM* для оновлення чатів та стрічок користувачів без перезавантаження сторінки.

– Велике community

*Facebook* доклала багато зусиль, щоб зробити *React* потужним інструментом для поліпшення інтерфейсу користувача будь-яких веб-додатків. Більше того, корпорація активно продовжує робити React все більш приємною та ефективною технологією.

Спільнота *React* надає тонни цінних бібліотек, що полегшують процес розробки, для вас відкриті нові способи та можливості покращити якість коду програм. Оскільки бібліотеками з готовими компонентами прийнято ділитися через інтернет, завжди можна впровадити в проект деякі з них, тим самим

суттєво скоротивши час, що витрачається на розробку інтерфейсів користувача.

– Браузерні інструменти

React-розробника *React Developer Tools* - це безкоштовне розширення для Chrome і Firefox, що надає цілий набір віджетів перевірки. Розширення спрощує налагодження, дозволяючи розробникам не лише шукати за списком усіх компонентів, але й переглядати глибоко вкладені компоненти прямо у браузері.

*Мінуси використання React*

– Погана документація

React - відносно нова технологія, вона продовжує розвиватися завдяки розробникам, які роблять її внесок, розширюючи її можливості.

Але документації щодо *React* та пов'язаних з ним бібліотек часто не вистачає. Більше того, розробники схильні випускати оновлення бібліотек без поновлення документації, що ускладнює інтеграцію бібліотек у існуючі проекти та знайомство з новими можливостями.

– Складність пошукової оптимізації

Пошукова оптимізація (*SEO*) є вкрай важливою, адже веб-додатку потрібно отримувати трафік і залучати нових клієнтів. Для кращого розуміння складності пошукової оптимізації програми на *React* потрібно зрозуміти, як *Google* індексує веб-сторінки.

Веб-сторінки індексуються спеціальними пошуковими роботами *Google*, ці роботи сканують вміст веб-сторінки і зберігають інформацію в індексі *Google*. Коли користувач запитує певну інформацію, *Google* перевіряє дані, що зберігаються в індексі, щоб надати найбільш релевантні джерела інформації.

Боти *Google* можуть легко індексувати *HTML*-сторінки. Однак з *JavaScript*-сторінками все вже не так гладко, адже динамічні веб-програми, у тому числі створені на *React*, проходять ускладнену процедуру індексації, на відміну від статичних веб-сторінок.

Таким чином, сторінка програми *React* може бути проіндексована неправильно або індексування може зайняти занадто багато часу. Що один випадок, що інший — обидва змусять ботів *Google* залишити сторінку сайту.

### 2.4.3. Angular

У теперішній час загальна продуктивність користувацьких пристроїв збільшується, тому стало можливим виконання логіки додатків у браузері. Це призвело до підходу до створення односторонніх додатків або *SPA*, для розробки яких був призначений *Angular*. Потім *Angular* був повністю переписаний під *Angular*, відповідно змінилося безліч основних концепцій фреймворка. У *Angular* відсутні контроллери або моделі представлення, замість цього є компоненти, які складаються з шаблонів, класів і метаданих. У той час, як *Angular* був зосереджений на шаблоні *MVC, Angular2* повністю покладається на ієрархію компонентів [2].

*Переваги використання Angular*

– Спрощена двостороння прив’язка даних

*Angular* дозволяє прив’язувати дані до *HTML* за допомогою виразів, а директиви *Angular* дозволяють розробникам розширювати функціонал *HTML* та створювати нові конструкції. Маніпуляції з *DOM* та код прив’язки даних обернуті у прості елементи, які можна швидко та просто вставити у *HTML* шаблони.

– Можливість керувати FE та BE

*Angular* входить до пакету MEAN, який також включає *MongoDB*, *Express.js* і *Node.js*. Тому він дозволяє керувати front end та back end проекту за допомогою JS.

– Модульна концепція

*Angular* побудований за технікою functionality-first, тому фреймворк найбільше підходить для розробки зверху донизу. Модульна концепція *Angular* дозволяє спростити розподіл роботи на різні команди у великих

проектах. У пріоритеті мінімальна кількість коду, тому програми *Angular*, як правило, компактні та легкі у редагуванні.

*Недоліки фреймворка Angular*

– Високий рівень входження

Різноманітність різних структур (*Injectables, Components, Pipes, Modules* тощо) ускладнює вивчення фреймворку і підвищує рівень входження порівняно з *React* і *Vue*, які мають лише «*Component*».

– Надлишковість

*Angular* також не пристосований обробляти маніпуляції з *DOM* з великою кількістю даних, оскільки покладається на «брудні перевірки» для управління змінами *DOM* (будь-яка зміна змінних тягне за собою оновлення *DOM*). Для більшості сайтів це не буде проблемою, але *GUI* редактори та відеоігри можуть класти.

### 2.4.4. Vue

Vue – це прогресивний фреймворк для створення користувацьких інтерфейсів. На відміну від фреймворків-монолітів, Vue створено придатним для поступового впровадження. Його ядро в першу чергу вирішує завдання рівня представлення (view), що спрощує інтеграцію з іншими бібліотеками, та існуючими проектами. З іншого боку, Vue повністю підходить і для створення складних односторонніх додатків (*SPA, Single-Page Applications*), якщо використовувати його спільно з сучасними інструментами та додатковими бібліотеками [3].

*Плюси використання Vue*

– Детальна документація

Vue.js має дуже хорошу документацію, яка може збільшити швидкість навчання розробників та заощадити багато часу на розробку програми з використанням базових знань *HTML* та *JavaScript*.

– Адаптивність

Vue.js забезпечує швидкий період переходу від інших фреймворків до *Vue.js* через його подібність з *Angular* і *React* з точки зору дизайну та архітектури.

– Чудова інтеграція

*Vue.js* можна використовувати як для створення односторінкових програм, так і для більш складних веб-інтерфейсів. Найважливіше, що невеликі інтерактивні частини можна легко інтегрувати в існуючу інфраструктуру, не надаючи негативного впливу на всю систему.

– Маленький розмір

*Vue.js* може важити близько 20 КБ і зберігати свою швидкість і гнучкість, що дозволяє досягти набагато більш високої продуктивності, порівняно з іншими фреймворками.

– Низький поріг входження

Vue вважається одним з найбільш легким фреймворком для входження, а тому приваблює нових розробників.

*Недоліки Vue*

– Маленьке ком'юніті

*Vue.js*, як і раніше, має досить невелику частку ринку в порівнянні з *React* або *Angular*. Це означає, що обмін знаннями у рамках фреймворку все ще формується.

– Ризик надмірної гнучкості

Іноді у *Vue.js* можуть виникати проблеми при інтеграції у величезні проекти, а досвіду про можливі рішення досі немає. Але вони обов'язково з'являться найближчим часом.

### 2.4.5. Порівняння фреймворків

Після короткого огляду найпопулярніших фреймворків сучасного світу web-розробки і виділення їх ключових особливостей, а також переваг і недоліків використання, було зведено усі дані в таблицю з порівнянням даних засобів розробки (табл. 2.1.).

Таблиця 2.1

Порівняння фреймворків за ключовими факторами

| Фреймворк | Швидкість | Дотримання стандартів | Community | Розробник | Рівень входження | Комплексність проекту |
|---|---|---|---|---|---|---|
| Angular | середня | + | велике | Google | високий | висока |
| React | достатня | - | велике | Facebook | середній | середня |
| Vue | висока | + | маленьке | Еван Ю | низький | низька |

Усі розглянуті вище фреймворки є популярними серед frontend-розробників та продовжують розвиватись, впевнено займаючи провідні місця на ринку веб-розробки.

Порівняльний графік зміни популярності з найбільш відомими фреймворками та бібліотеками для фронтенд розробки за останні 6 років зображено на рис. 2.5. [4].

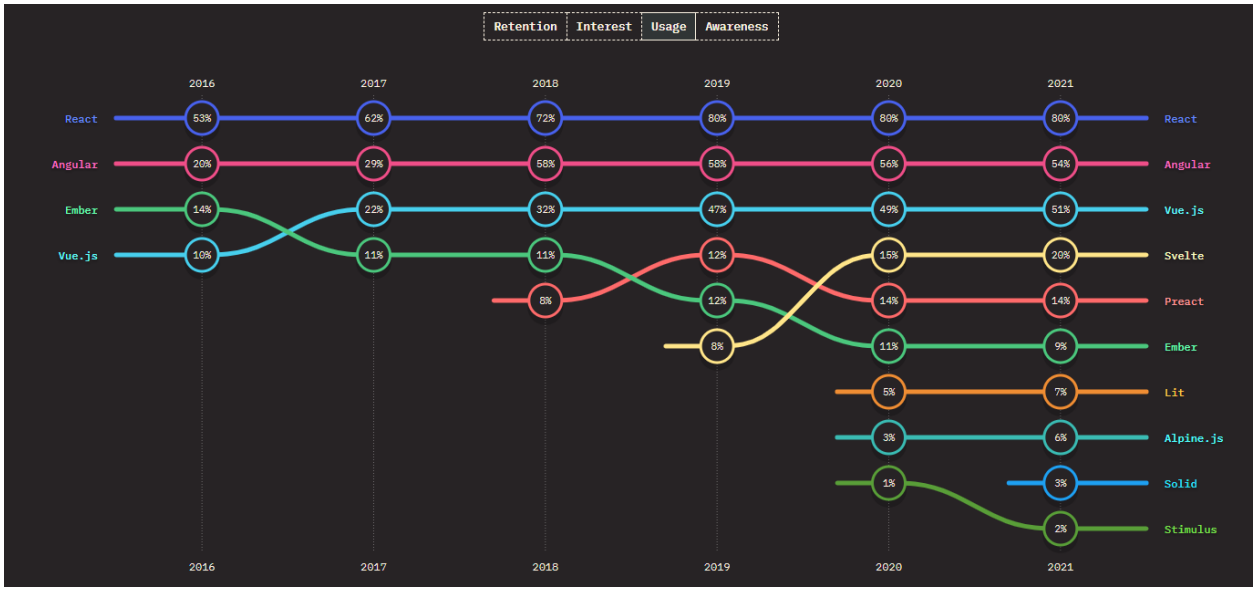



Рис.2.5. Популярність за використанням web-фреймворків станом на кінець 2021 року

З огляду на всі вищезазначені короткі характеристики кожного з фреймворків, а також за особистими вподобаннями було вирішено використовувати *Vue* в якості фреймворку для програмної реалізації

кваліфікаційної роботи. Ключовими факторами у виборі стали тривалий досвід роботи з даною технологією, а також простота використання.

## 2.5. Інструменти і технології для Vue.js

В ході розробки програмної реалізації кваліфікаційної роботи крім базових засобів web-розробки також були використані сучасні технології для покращення функціональності web-додатку.

### 2.5.1. Бібліотека VueX

*Vuex* – це патерн керування станом + бібліотека для web-додатків, що розроблені на *Vue.js*. Він служить централізованим сховищем даних всіх компонентів додатку з правилами, що гарантують, що стан може бути змінено лише очікуваним способом [14].

*Vuex* надає можливість розробникам впоратися зі спільним управлінням станом за рахунок додаткових концепцій і шаблонів. Це компроміс між короткостроковою та довгостроковою продуктивністю.

За допомого *Vuex* виконується кешування даних отриманих з серверу, що дозволяє забезпечити принцип оптимізації і не потрібно виконувати один і той же запит для отримання ідентичних даних.

### 2.5.2. Vue Router

Кожній платформі інтерфейсу *JavaScript*, яка дозволяє створювати односторінкові програми, потрібен спосіб переходу користувачів з однієї сторінки на іншу. Всім цим потрібно керувати на стороні клієнта, синхронізуючи уявлення, яке зараз відображається на сторінці, з *URL*-адресою в адресному рядку.

*VueRouter* – це пакет для фреймворку *Vue*, який дозволяє налаштовувати маршрутизацію для односторінкових програм (*SPA*).

## 2.6. Система контролю версій Git

Система контролю версій – це система, що записує зміни до файлу або набір файлів протягом часу і дозволяє повернутися пізніше до певної версії. Для контролю версій файлів у цій книзі як приклад використовуватиметься вихідний код програмного забезпечення, хоча насправді ви можете використовувати контроль версій практично для будь-яких типів файлів [15].

*Git* – це абсолютний лідер за популярністю серед сучасних систем управління версіями. Це розвинений проект з активною підтримкою та відкритим вихідним кодом. Система *Git* була спочатку розроблена в 2005 Лінусом Торвальдсом - творцем ядра операційної системи *Linux*. *Git* застосовується для управління версіями в рамках колосальної кількості проектів з розробки програмного забезпечення, як комерційних, так і з відкритим вихідним кодом. Система використовується безліччю професійних розробників програмного забезпечення. Вона чудово працює під управлінням різних операційних систем і може застосовуватися з безліччю вбудованих середовищ розробки (*IDE*).

*Git* – система управління версіями з розподіленою архітектурою. На відміну від колись популярних систем на кшталт *CVS* і *Subversion* (SVN), де повна історія версій проекту доступна лише одному місці, в Git кожна робоча копія коду як така є репозиторієм. Це дозволяє всім розробникам зберігати історію змін у повному обсязі.

Розробка в Git орієнтована на забезпечення високої продуктивності, безпеки та гнучкості розподіленої системи [15].

## 2.7. Технології для розробки бази даних

База даних (БД) — це організована структура, яка призначена для зберігання, зміни та обробки взаємозалежної інформації, переважно великих обсягів. У технічному розумінні включно й система управління БД.

Система управління базами даних (СУБД) - це комплекс програмних і мовних засобів, необхідних для створення баз даних, підтримання їх в актуальному стані та організації пошуку в них необхідної інформації.

Найпоширенішою технологією баз даних сьогодні є реляційна база даних. Реляційні бази даних зберігають дані нормалізованим способом — це означає, що дані розбиваються на різні таблиці, щоб уникнути надмірності. Хоча реляційні бази даних існують вже досить давно, вони пропонують універсальний інструмент як для зберігання даних, так і для управління ними.

В розробленому web-застосунку повинні відображатися користувацькі плейлисти. Сторінка буде відображатися користувачеві за допомогою *HTML*, *CSS* і *Vue*, тоді як серверна сторона обробляє з'єднання з базою даних, виконує запит і повертає дані для відображення.

Реляційні бази даних складаються з двох або більше таблиць із пов'язаною інформацією, кожна зі стовпцями та рядками. Ці зв'язані таблиці називаються об'єктами бази даних; для створення та керування ними потрібна система управління реляційною базою даних. Системи управління базами даних дозволяють розробникам створювати та підтримувати програму бази даних, включаючи інструменти для:

- створення даних для запитів;
- редагування існуючих даних;
- проектування загальної структури бази даних;
- збір інформації та генерація звітів;
- перевірки даних в таблицях на відповідність.

Вони часто містять вбудовану мову програмування для автоматизації деяких з цих функцій, наприклад SQL. Використовуючи мову запитів, розробники можуть отримувати дані на основі значення певного стовпця, об'єднувати пов'язані дані з результатом, виконувати розширені обчислення, форматувати дані у потрібний спосіб тощо.

## ВИСНОВОК ДО РОЗДІЛУ 2

Другий розділ демонструє обрані інструменти та технології, які використовувались при роботі над програмною реалізацією кваліфікаційної роботи.

Описано та обґрунтовано вибір середовища розробки, а також описані ключові відмінності редактору коду та інтегрованого середовища розробки.

Розглянуто графічний редактор *Figma* для розробки інтерфейсу веб-додатку.

Розглянуто технології web-розробки *HTML, CSS*, а також препроцесор *SCSS*. Було зроблено короткий огляд мови програмування *JavaScript*, а також описані основні фреймворки для розробки web-додатків і проведено їх порівняльний аналіз.

В розробленому браузерному музичному плеєрі основними технологіями та засобами для розробки було вирішено використовувати *HTML, CSS* та *Vue.js*, побудований на основі мови програмування JavaScript. *HTML* та *CSS* дозволяють створити загальну структуру користувацького інтерфейсу та задати стилі зовнішнього вигляду та розміщення елементів на веб-сторінках.

Web-додаток було розроблено з використанням сучасного стеку технологій. За допомогою розглянутих у другому розділі технологій та засобів розробки було створено стабільний та надійний web-додаток, що легко тестується та масштабується.

# РОЗДІЛ 3
# РОЗРОБКА ІНТЕРФЕЙСУ WEB-ДОДАТКУ

Розробка дизайну веб-застосунку була здійснена за допомогою онлайн-сервісу для розробки інтерфейсів *Figma*.

## 3.1. UI/UX дизайн і його важливість

Розробка продукту, який подобається користувачам, часто потребує якісного *UI* та *UX* дизайну. Наприклад, є банківська програма, яка виглядає чудово та має інтуїтивно зрозумілу навігацію (*UI*), але якщо програма завантажується повільно або змушує користувача робити багато непотрібних дій, щоб переказати кошти (*UX*), гарний інтерфейс перестає мати значення. Ймовірно, користувачі не будуть користуватися подібним додатком.

З іншого боку, веб-сайт може бути завантажений унікальним, корисним вмістом, організованим у логічний та інтуїтивно зрозумілий спосіб, але якщо він виглядає застарілим або юзер не можете легко та швидко зрозуміти, як переміщатися між сторінками чи прокручувати параметри, він, скоріш за все, не захоче користуватися подібним web-додатком.

Щоб зрозуміти роль *UI/UX* дизайну в розробці застосунків, спочатку потрібно визначити, що собою являє і які має цілі *UI* та *UX* дизайн окремо.

*UI* дизайн відповідає за вигляд сторінок, кнопок, перемикачів, іконок та інших візуальних елементів, з якими взаємодіє користувач під час використання веб-сайту, програми чи іншого електронного пристрою. *UX*, в свою чергу, відповідає за всю взаємодію юзера з продуктом, включаючи те, як він ставиться до цієї взаємодії [16].

*UX* дизайн відповідальний конкретно за представлення властивостей та функціональних можливостей користувачу в такому вигляді, щоб створений продукт був приємним у взаємодії та простим у використанні. З вищесказаного, можна виділити мету *UX* дизайну максимально просто привести користувача до його фінальної цілі та забезпечити вирішення проблеми.

*UI* дизайн відповідальний за візуальне представлення застосунку, тобто вибір кольорової палітри, читабельність тексту, зручність використання візуальних компонентів.

Отже, задачею *UI/UX* дизайну є вплив на правильний вибір користувача через створений інтерфейс. Створення всіх умов задля того, щоб кінцевий користувач дотримувався певного алгоритму дій при роботі з застосунком на інтуїтивному рівні.

Задля розробки зручного та дружнього інтерфейсу варто дотримуватись певних правил:

спрощення взаємодії користувача з застосунком. Чим швидше користувач досягає своєї цілі, тим кращим є користувацький досвід;

колір – це функціональна частиною застосунку. Вибір кольорової палітри впливає на взаємодії користувача з системою. За допомогою кольорових акцентів можна загострити увагу користувача, виділи відмінності;

використання знайомих користувачу стандартів. Наприклад, усі знають, що кнопки мають прямокутну чи круглу форму. Це вже стандарт, а тому кнопки інших форм можуть бути незрозумілі для користувачів;

діалоговий підхід. Використання діалогового підходу може бути набагато кращим методом отримання даних, аніж форми.

### 3.2. Створення макету і налаштування сітки

Розробка дизайну будь-якого веб-сайту починається зі створення макету. Перш за все необхідно зробити фрейм (робочий шар). Для цього треба вибрати інструмент *Frame* або натиснути кнопку F, і намалювати

прямокутник. Далі необхідно налаштувати розміри даного фрейму. Він є основним і повинен збігатися з розмірами майбутнього веб-застосунку. В даному випадку це буде 1280x720px. Для ширших екранів контент буде вирівнюватися по ширині.

Далі необхідно додати сітку, тобто *Layout Grid*. Сітка потрібна для того, щоб дизайнер міг вибудовувати всі елементи щодо фрейму і один одного, дотримуватися структури, композиції. Також сітка в макеті значно полегшить написання css-стилів для розробників.

Стандартною сіткою для дизайнів, що орієнтовані на розміри комп'ютерних дисплеїв – є 12-колоночна сітка. Отже, щоб додати таку сітку до основного фрейму необхідно на правій робочій панелі перейти до розділу Layout Grid та натиснути кнопку «+». Автоматично у Фігмі включається сітка у вигляді клітини 10x10px. Змінюємо параметри – установлюємо кількість колонок 12, марджини задаємо 50px, а відступи між колонками – 20px. На рис. 3.1. зображені вибрані параметри сітки.

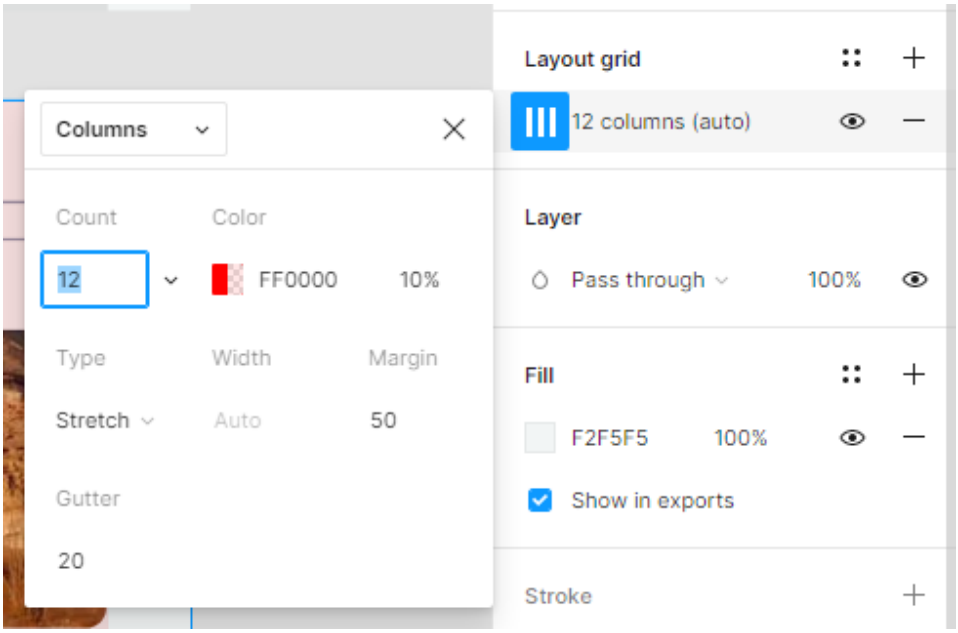



Рис. 3.1. Параметри сітки

Таким чином була створена сітка для майбутнього веб-застосунку (рис.3.2.).

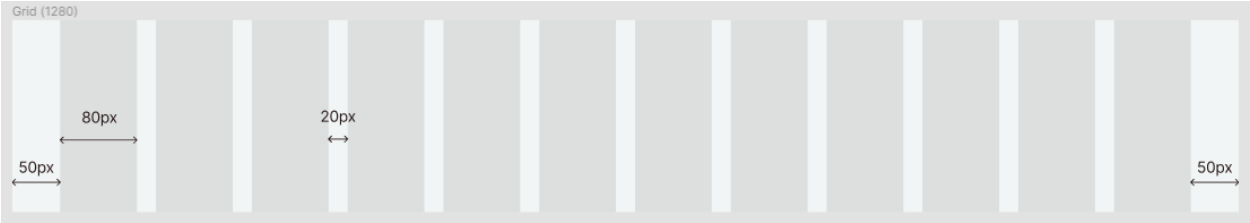



Рис. 3.2. Вигляд готової сітки

## 3.3. Розробка логотипу

### 3.3.1. Логотип, його основні функції та види

Логотип або фірмовий шрифтовий напис – це унікальне зображення назви бренду або конкретного товару, яке використовується для підвищення впізнаваності серед споживачів.

Основні функції логотипу:

– ідентифікація. Графічний або шрифтовий символ формує унікальний образ бренду та робить продукт таким, що впізнається;

– комунікація. За допомогою взаємодії зі споживачем, виробник може донести до нього інформацію про ключові особливості продукту або компанії;

– формування привабливого образу. Естетично привабливий логотип викликає позитивні емоції, які автоматично переносяться на продукт.

*Види логотипів*

Виділяють текстові, символьні (графічні), комбіновані логотипи і емблеми (назва та графічні елементи, укладені в певну форму).

1. Текстові логотипи

Особливістю текстових логотипів є використання виключно букв та цифр. Використання графічних елементів виключено. За допомогою різних шрифтів можна створити не менш оригінальне лого, яке буде незабутнім серед покупців усього світу. *Samsung, Canon, Netflix* створили нехитрі логотипи, які впізнаються практично будь-якою людиною (рис. 3.3.).


Рис. 3.3. Приклад текстових логотипів

2. Символьні логотипи

Відомі також як графічні знаки, символьні логотипи або символи дуже популярні серед усіх видів бізнесу. Яскравим прикладом таких логотипів є знак *Apple*. Логотип *Twitter* також є надихаючим прикладом (див. рис. 3.4.). Оскільки ці бренди тепер добре відомі та визнані, їхні логотипи мають значний статус серед типів фірмових знаків.

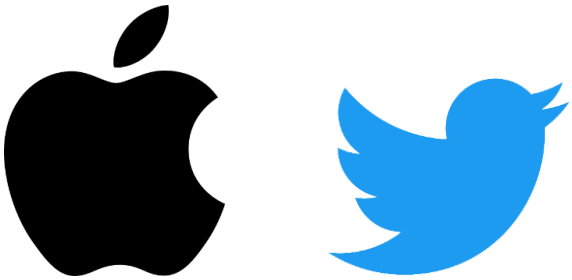


Рис. 3.4. Приклад символьних логотипів

Однак для нових компаній символьний лого може бути не найкращим варіантом, оскільки вони не мають великої впізнаваності бренду.

3. Комбіновані логотипи

У логотипі з комбінованим знаком використовуються елементи текстових та абстрактних логотипів. Існує комбіноване використання графічних знаків, абревіатур та елементів талісмана. У цих логотипах дизайнер використовує зображення і текст пліч-о-пліч, разом з дизайном. В основному зображення та текст накладаються один на одного. *BurgerKing* і *Lacoste* є кращими прикладами комбінованих логотипів (зображено на рис. 3.5).


Рис. 3.5. Приклад комбінованих логотипів

### 3.3.2. Концепції логотипу для РИТМ

Виходячи з того факту, що об'єктом дослідження кваліфікаційної роботи є браузерний музичний плеєр, було вирішено спробувати передати в назві та логотипі веб-застосунку тематику музики. Назва веб-застосунку – РИТМ (RHYTHM).

Початкова розробка ескізів логотипу мала 3 концепції – перша відтворювала текстове зображення назви веб-застосунку, друга базувалась на зображенні назви веб-застосунку з графічним символом хвилі над нею, третя ж концепція пропонувала використати додавання мінімалістичного зображення хвилі прямо в текстовому логотипі замість літери «М».

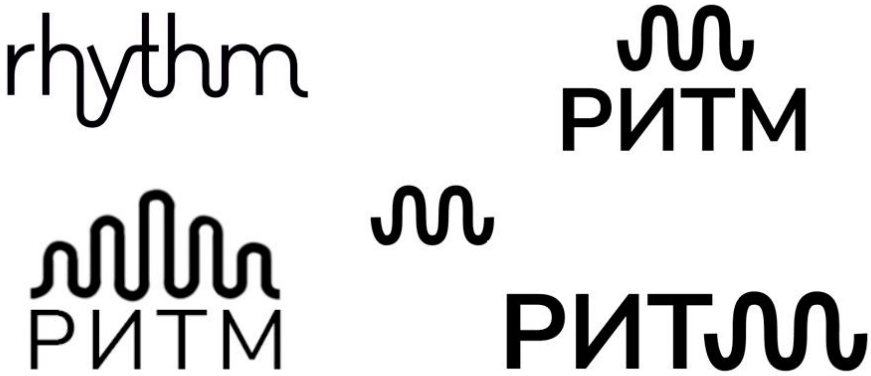



Рис. 3.6. Пошукові ескізи логотипу

Після ряду пробних ескізів логотипу (рис. 3.6.), що містили зазначені вище концепції, було вирішено відштовхуватись від образу хвилі як умовного графічного зображення ритму. Кінцевий варіант базується на концепції комбінованого логотипу з графічною буквою «М» у вигляді хвилі (рис. 3.7).

Скорочену версію логотипу можна буде використовувати у ролі фавікону для веб-застосунку, що розробляється.


Рис. 3.7. Опис та структура логотипу

Іконку для веб-застосунку, що буде зображатися у браузерах в адресному рядку поруч з заголовком сайту можна побачити на рис. 3.8.


Рис. 3.8. Фавікон сайту

### 3.4. Створення дизайну сторінки «Головна»

При відкриванні веб-сайту, що розробляється, першим, що побачить користувач, буде головна сторінка. Дослідивши веб-застосунки-аналоги, а також існуючі дизайнерські рішення в *Community Figma*, було вирішено перейняти найбільш популярні рішення. Спочатку необхідно скласти структуру даних, які будуть відображатися користувачам – на головній сторінці повинна бути сама важлива інформація, основна навігація, показані рекомендації та новинки для користувача.

Базова структура сторінки буде складатися з трьох частин: навігація, банер та рекомендації (рис. 3.9).

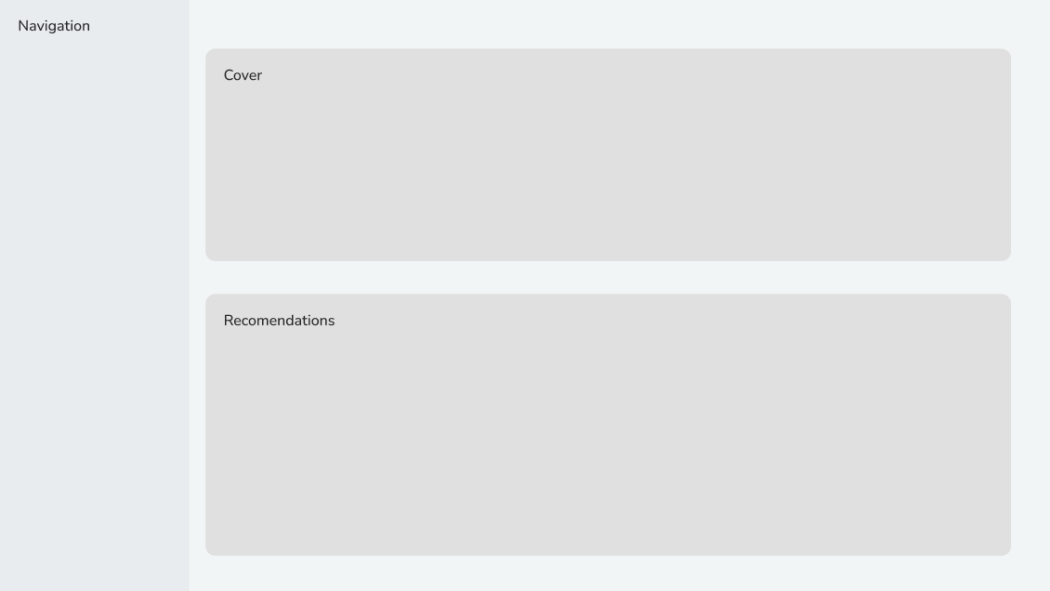



Рис. 3.9. Базова структура головної сторінки

### 3.4.1. Навігація

Зазвичай меню та навігація мають вигляд, так званого, «бургер»-меню, але в даному дизайні було вирішено зробити меню у вигляді сайдбару, таким чином, користувач завжди буде мати швидкий доступ до навігації. В останньому пункті «Плейлисти» будуть відображатись перші 3 плейлисти користувача. Також саме бокова панель буде містити логотип сайту.

Розроблений дизайн для бокової панелі зображений на рис. 3.10.

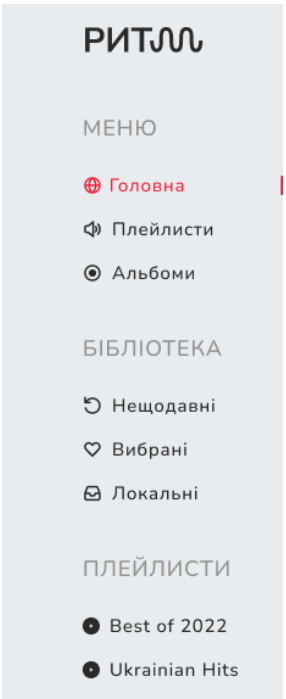



Рис. 3.10. Бокова панель веб-додатку

### 3.4.2. Банер

На головній сторінці буде розміщено банер з інформацією про новинки у світі музики. Цей банер буде містити в собі слайдер – компонент слайд-шоу для перегляду серії слайдів, в даному випадку слайдами будуть виступати нові популярні композиції та альбоми на сайті.

Сам слайд складається з обкладинки альбому чи композиції – зображення на задньому фоні, а також назва і виконавець.

У верхньому лівому куті знаходяться кнопки управління слайдером.

Банер з реальною інформацією зображено нижче на рис. 3.11.

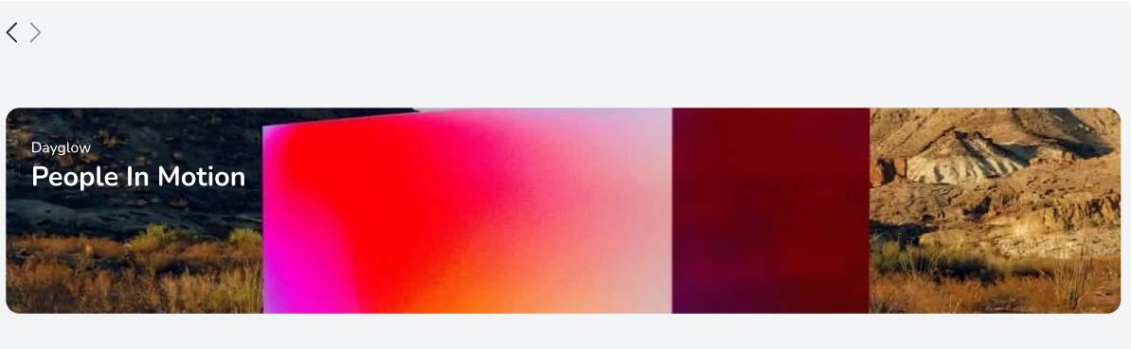



Рис. 3.11. Дизайн банеру

### 3.4.3. Дизайн для компоненту композиції

Компонент композиції повинен містити таку інформацію, як обкладинка, назва композиції, ім’я виконався, а також тривалість у хвилинах. Також компонент містить кнопку для початку програвання треку, кнопку у вигляді сердечка для додавання композиції у «вибрані» і кнопку «…» для додаткових дій, таких як додавання композиції в плейлист.

Дизайн макету композиції зображений на рис. 3.12.


Рис. 3.12. Макет композиції

### 3.4.4. Рекомендації

Блок Рекомендацій складається з двох частин:

1. «Для Вас» – колекція рекомендованих до прослуховування композиції, зібраних на основі вподобань користувача.

2. «Нещодавні» – список останніх композицій, що прослухав користувач.

Розроблений інтерфейс блоку «Рекомендації» можна побачити на рис. 3.13.

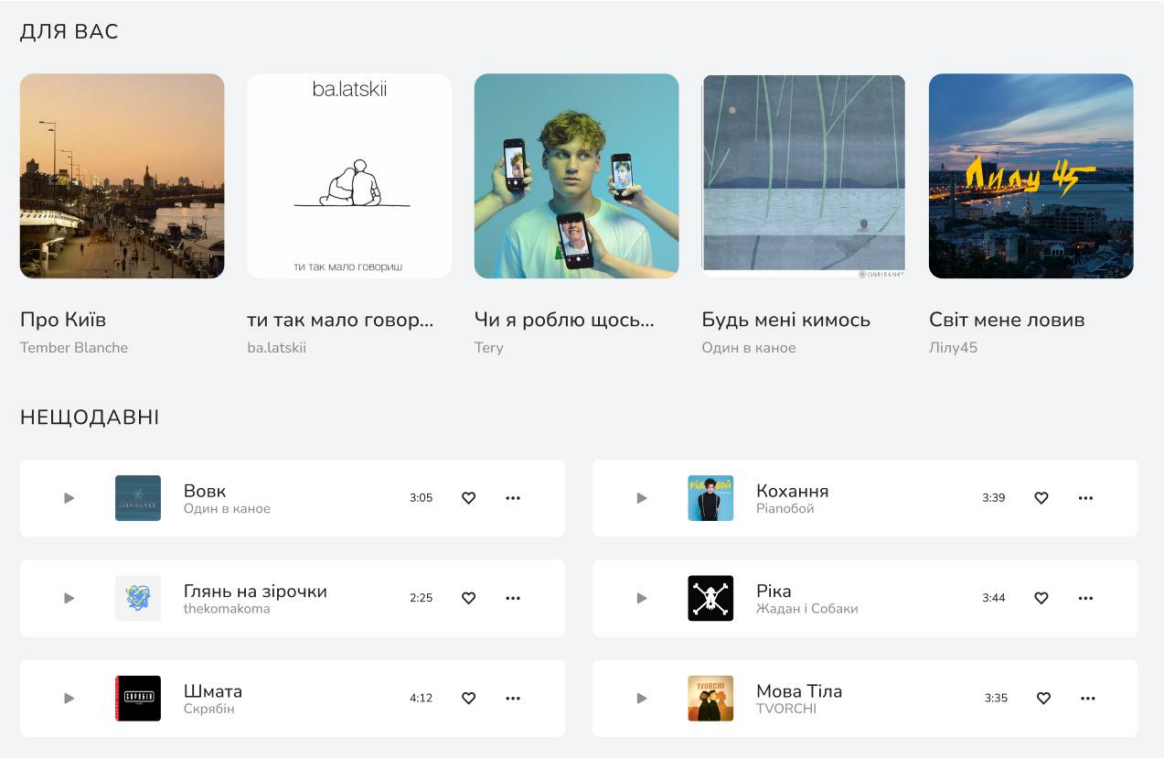



Рис. 3.13. Блок «Рекомендації»

### 3.4.5. Дизайн головної сторінки

Кінцевий дизайн сторінки «Головна» зображений нижче на рис. 3.14. Висота розробленої сторінки виходить довшою за стандартну висоту дисплеїв, тому передбачено, що буде доступна можливість вертикального гортання сторінки.

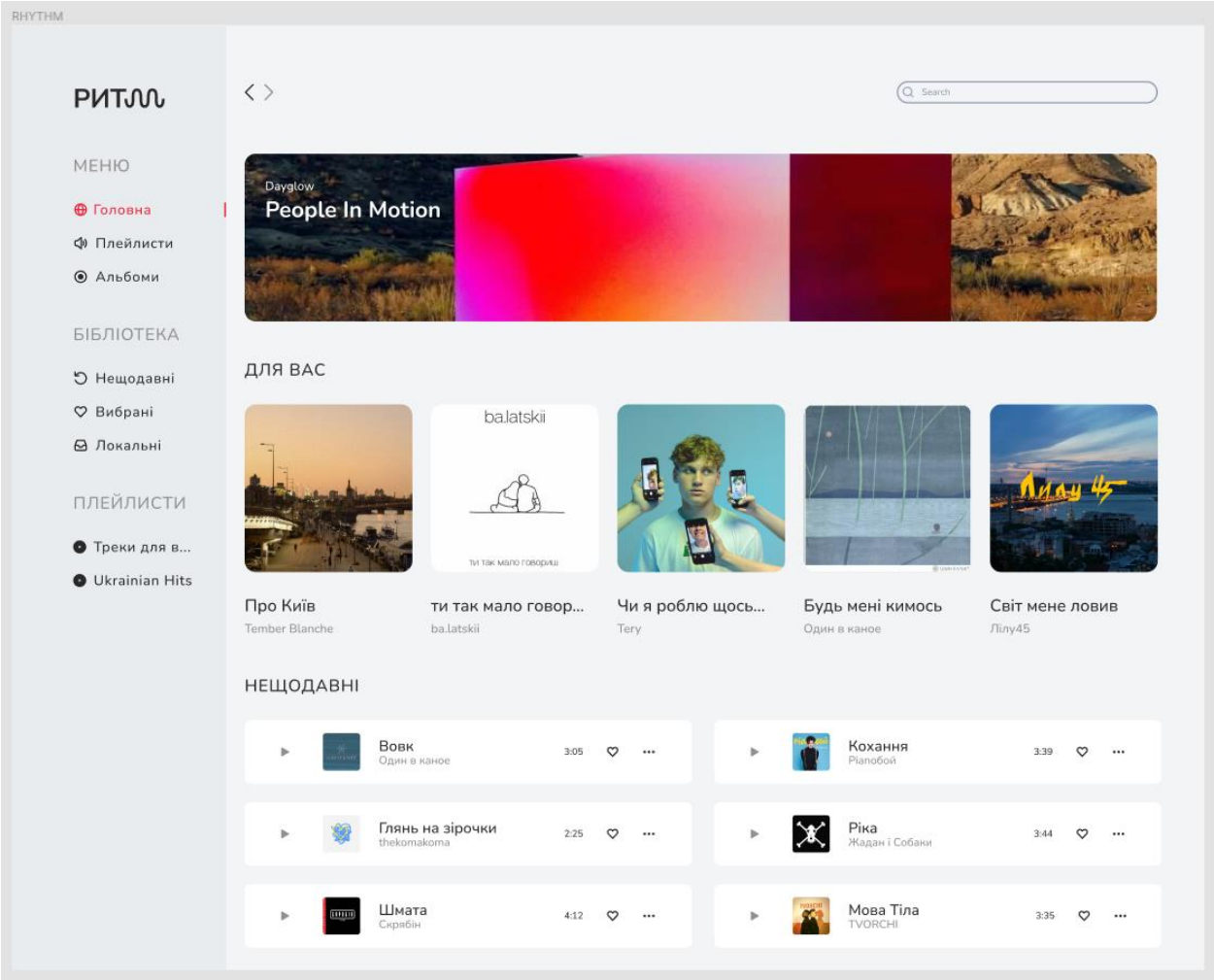



Рис. 3.14. Інтерфейс головної сторінки

## 3.5. Створення сторінки «Плейлисти»

Сторінка лістингу плейлистів буде містити список усіх користувацьких плейлистів. Її створення починається зі створення макету для картки плейлисту (рис. 3.15).

Для досягнення візуальної відмінності плейлистів і композицій було вирішено, що картка плейлисту буде мати квадратну форму з розмірами 180x180 пікселів і коротку інформацію про назву, автора і кількість композицій внизу.

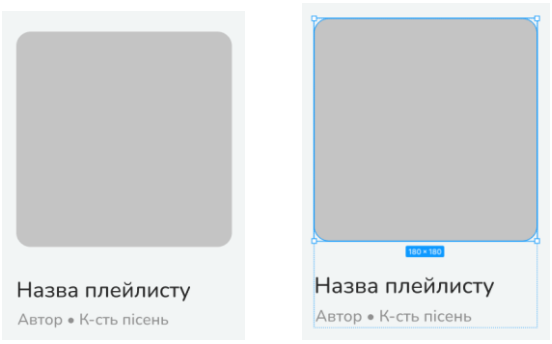



Рис. 3.15. Макет картки плейлисту

Картки будуть формувати список плейлистів та мають розміщуватися з інтервалом в 20 пікселів. Макет списку плейлистів зображений на рис. 3.16.

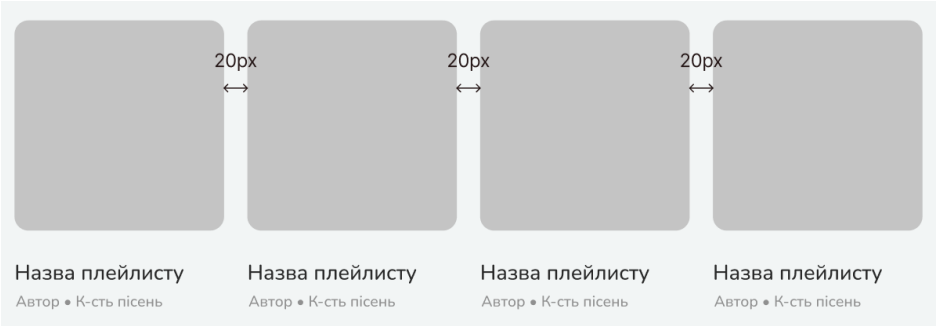



Рис. 3.16. Макет списку плейлистів

Готовий дизайн сторінки лістингу плейлистів надано на рис. 3.17.

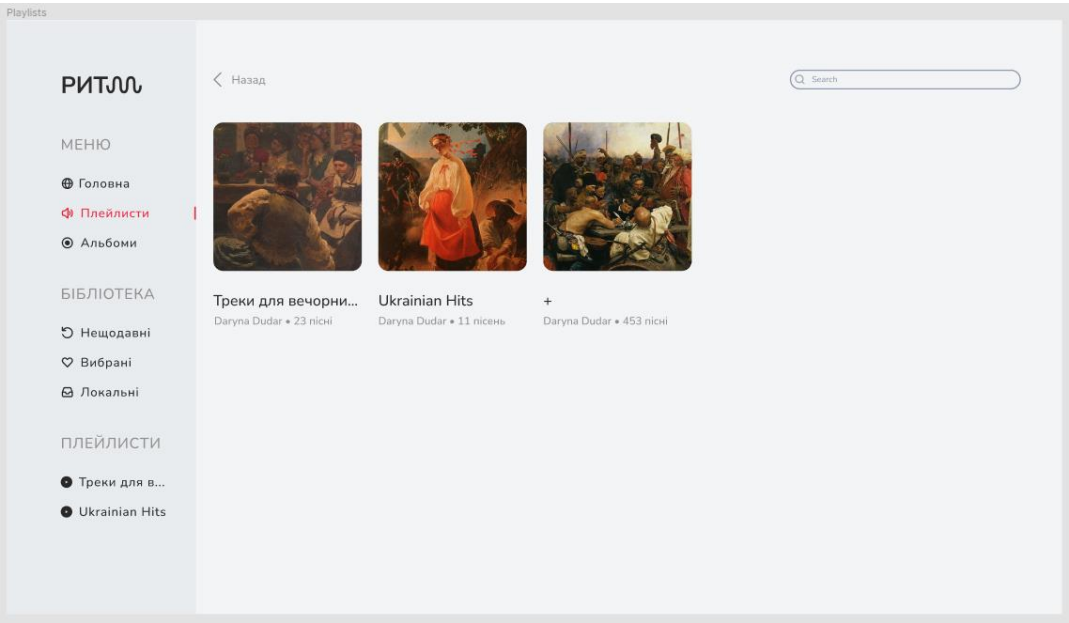



Рис. 3.17. Інтерфейс сторінки лістингу плейлистів

### 3.6. Створення дизайну сторінки плейлисту

Однією з найкращих практик дизайну є слідування правилу «використання знайомих користувачу стандартів», що вже було згадано. Це правило також стосується базової структури сторінок. Дослідивши web-застосунки-аналоги можна зробити висновок, що усі вони дуже подібні, а тому було вирішено використати подібну структуру.

Отже, сторінка плейлисту буде містити список треків, а також обкладинку з наступною інформацією (рис. 3.18):

– назва добірки;

– конфіденційність (приватний – «Для Вас», публічний – «Для усіх»);

– автор (тільки для публічних плейлистів);

– кількість треків:

– тривалість в годинах.


Рис. 3.18. Вигляд обкладинки плейлисту

На усіх сторінках, окрім «Головної» буде відображена кнопка «Назад» (рис. 3.19)

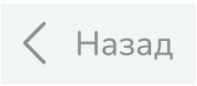



Рис. 3.19. Кнопка «Назад»

Розроблений дизайн сторінки плейлисту зображено на рис. 3.20.

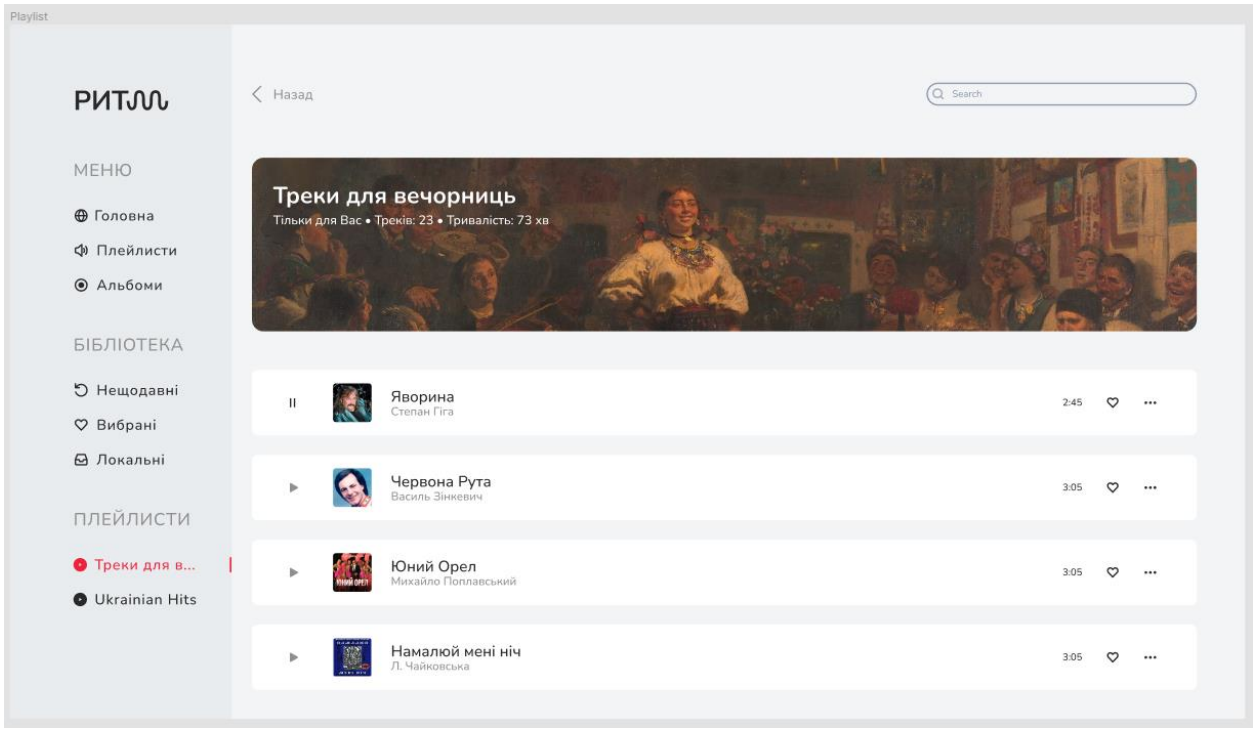



Рис. 3.20. Інтерфейс сторінки плейлисту

## 3.7. Створення дизайну сторінки «Нещодавні»

Сторінка «Нещодавні» буде містити список нещодавно прослуханих композицій, розділених за часом. Інтерфейс сторінки надано на рис. 3.19.

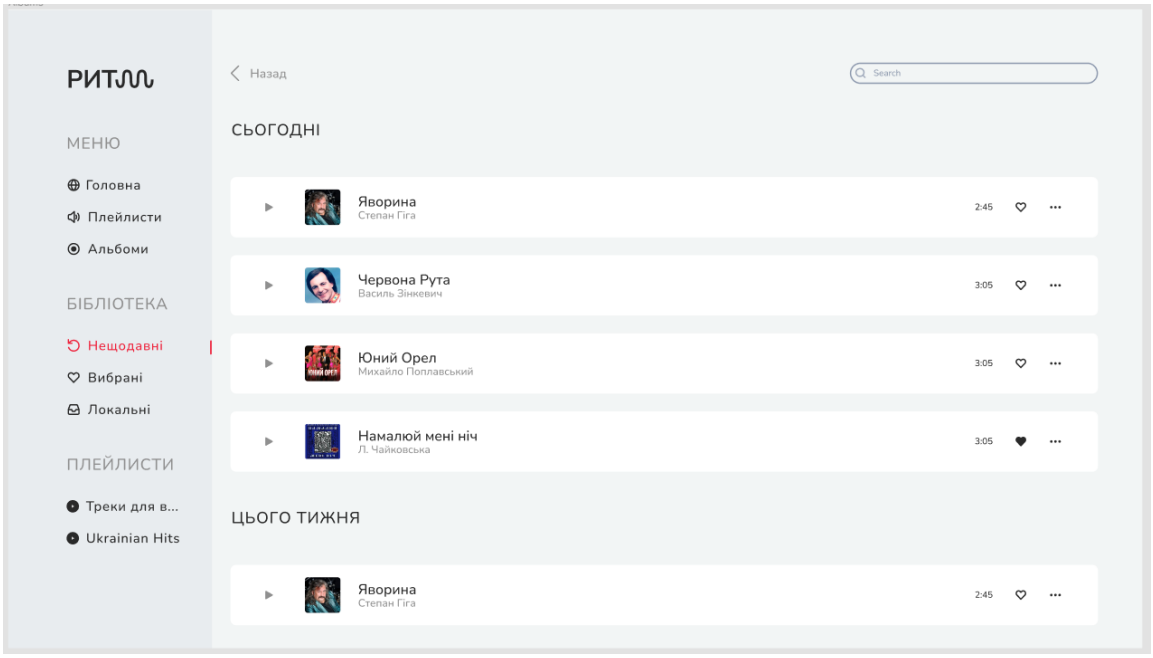



Рис. 3.19. Інтерфейс сторінки плейлисту

## 3.8. Створення дизайну сторінки «Вибрані»

Сторінка «Вибрані» - це по великому рахунку просто автоматично зібраний плейлист, що складається з усіх композицій, які були позначені. Саме тому вигляд цієї сторінки нічим не відрізняється від звичайного плейлисту.

Кінцевий інтерфейс сторінки «Вибрані» продемонстровано на рис. 3.20.

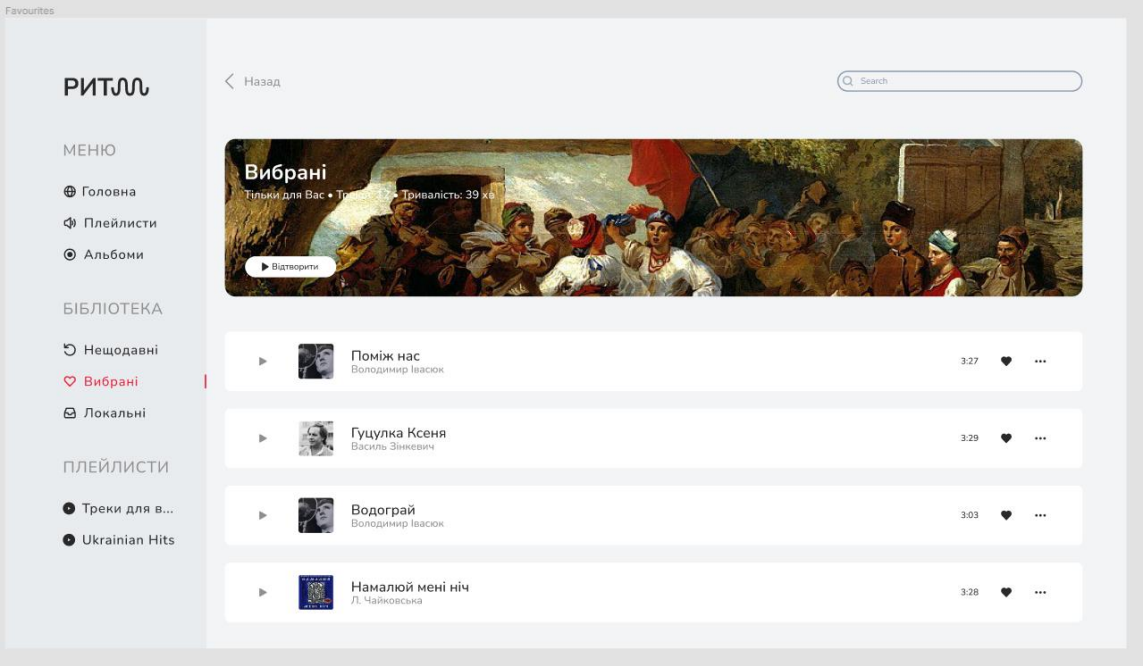



Рис. 3.20. Інтерфейс сторінки «Вибрані»

## ВИСНОВОК ДО РОЗДІЛУ 3

Третій розділ випускної кваліфікаційної роботи присвячений проектуванню користувацького інтерфейсу веб-застосунку.

Дизайн веб-застосунку спроектовано за допомогою онлайн-сервісу для розробки інтерфейсів *Figma*.

Розробка продукту, який подобається користувачам, часто потребує якісного і продуманого *UI* та *UX* дизайну. Красивий і сучасний сайт може залучити аудиторію, створити хороше перше враження, але якщо користувачі не зможуть зрозуміти мету відвідування ресурсі або ж не розберуться в навігації, вони будуть доволі швидко покидати сайт. Так само і зручний і зрозумілий інтерфейс не буде приваблювати аудиторію, якщо візуально сайт застарійлий і нестильний. *UI/UX* дизайн – це як симбіоз візуального дизайну, інженерних навиків і тонкого розуміння психології.

Було розглянуто поняття логотипу, зроблено огляд його основних функцій та видів з наведенням конкретних прикладів. Розроблено основні концепції для логотипу додатку, наведено ряд пробних ескізів та описано кінцевий варіант логотипу.

Спроектовано та розроблено дизайни для усіх сторінок і розділів веб-додатку, а також для допоміжних компонентів.

# РОЗДІЛ 4
# ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-ДОДАТКУ

## 4.1. Підготовка до розробки додатку

Перш ніж перейти безпосередньо до розробки програмної реалізації кваліфікаційної роботи слід встановити і налаштувати усі необхідні засоби розробки.

Для початку розробки попередньо були встановлені такі інструменти:

– редактор коду *VS Code*, короткий опис якого був викладений в розділі 2.1;

– розширення *Volar* для редактору коду *VS Code* для зручності написання кода за допомогою фреймвоку *Vue.js*;

– додаток *GitHub Desktop* для пришвидшення роботи з системою контролю версій Git;

– актуальна версія *Node.js*;

– *Vue.js devtools* – розширення для браузера для дебаггінгу веб-застосунків, що написані за допомогою фреймворку *Vue.js*.

## 4.2. Файлова структура проекту

Проект кваліфікаційної роботи складається з наступних директорій:

– ~/node-modules/

Автоматично згенерований каталог для зберігання модулей, що з’являється після виконання команди «*npm install*». Папка містить усі встановлені залежності для вашого проекту;

– ~/public/

Директорія, в якій міститься файл index.html і фавікон для веб-додатку. index.html – це сторінка, що надає точку входу в html та забезпечуючи елемент для завантаження Vue-файлів, а також імпорт main.js для ініціалізації сайту;

– ~/src/

Назва каталогу походить від англійського слова “source” і означає вихідний код. Дана директорія призначена для зберігання вихідного коду перед мінімізацією, конкатенацією чи іншою компіляцією, яка використовується для читання/редагування коду;

– ~/src/api/

Каталог містить усі служби, які забезпечують зв’язок між додатком *Vue* та *API* (backend);

– ~/src/assets/

Директорія, що призначена для зберігання медіа файлів для web-додатку;

– ~/src/components/

Папка компонентів містить кожен окремий компонент web-додатку. Зазвичай, для кожного компоненту використовується своя власний каталог, в якому містить файл компоненту і стилі до нього;

– ~/src/router/

Даний каталог зберігає всі файли, що пов’язані з роутингом. Це може бути просто index.js із маршрутизатором і маршрутами в одному місці (у цьому випадку гарною ідеєю буде зберегти цей файл у кореневому каталозі src);

– ~/src/store/

Директорія, що зберігає глобальні стани web-додатку і працює безпосередньо з *VueX*. В даному каталозі також будуть зберігатися кореневий стан, дії, мутації та геттери;

– ~/src/styles/

Каталог використовується для зберігання таблиць стилів для web-додатку. Він може містити теми, Bootstrap та будь-які інші стилі. Дану директорію встановлено в стандартному Vue skeleton.

– ~/src/dist/

Директорія з мінімізованим кодом, що з'являється після виконання команди «npm run build» і містить готовий до продакшну проект.

Крім вищеперелічених каталогів, директорія ~/src/ містить наступні файли:

– *App.vue* – компонент найвищого рівня, шо містить увесь додаток. Це головний файловий компонент, який є батьківськми для всіх інших компонентів;

– *main.js* – зазвичай це файл JavaScript, який ініціює цей кореневий компонент елементом на вашій сторінці. Він також відповідає за налаштування плагінів та сторонніх компонентів, які ви можете використовувати у своєму проєкті.

Коренева директорія проекту містить наступні файли:

– *package.json* – цей файл містить різні метадані, що стосуються проекту. Даний файл використовується для надання інформації *npm*, яка дозволяє йому ідентифікувати проект, а також обробляти залежності проекту. Він також може містити інші метадані, такі як опис проекту, версія проекту в певному дистрибутиві, інформація про ліцензію, навіть дані конфігурації - все це може бути життєво важливим як для npm, так і для кінцевих користувачів пакету;

– *babel.config.js* – файл, що містить конфігурацію для усього проекту. *Babel* – це ланцюжок інструментів, який в основному використовується для перетворення коду ECMAScript 2015+ у зворотно сумісну версію JavaScript у поточних і старих браузерах або середовищах;

– *README.md* – це файли *Markdown*, які описують каталог. *GitHub* і *Gitiles* відображають його, коли користувач переглядає каталог.

Файлова структура проекту зображена на рис. 4.1.

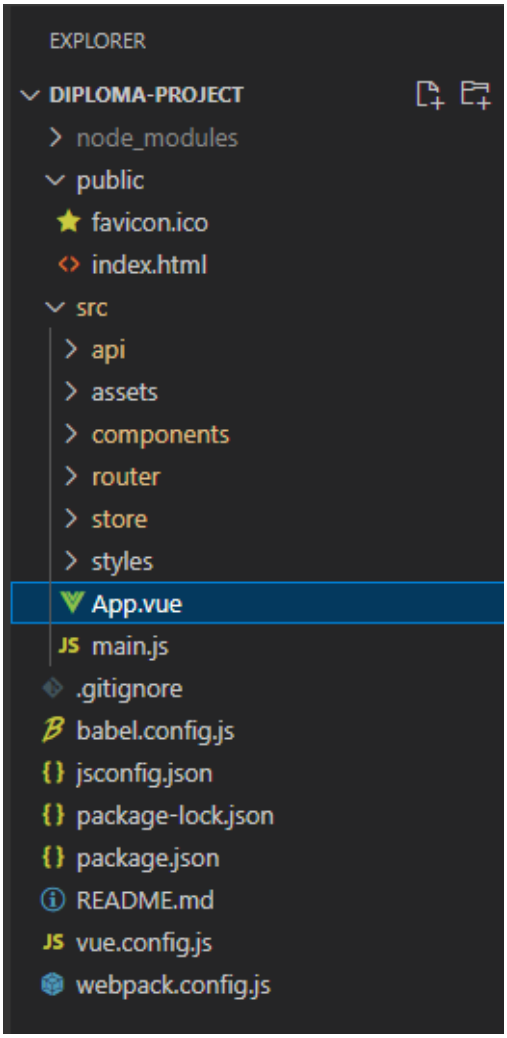



Рис. 4.1. Файлова структура проекту

## 4.3. Проектування та розробка БД

Основна дані web-застосунку зберігається в базі даних. Для правильного і коректного проектування бази даних необхідно дотримуватися деяких правил, які включають створення таблиць та впровадження відношень між ними, забезпечуючи захист даних та роблячи базу даних більш гнучкою.

Необхідно визначити основні сутності та їх атрибути, про які необхідно зберігати дані. В розробленому web-застосунку основними сутностями є:

– користувач (*User*) – містить інформацію про зареєстрованого користувача;

– виконавець (*Artist*) – містить інформацію про виконавця, назву та короткий опис;

– альбом (*Album*) – включає інформацію про альбом, назву, виконавця, кількість треків і дату релізу;

– пісня (*Song*) – сутність пісні, що містить інформацію з назвою, назвою альбому, ім’ям виконавця та датою релізу;

– плейлист (*Playlist*) – містить інформацію пов’язану з користувацьким плейлстом, таку як його назва, кому належить даний плейлист, кількість пісень в ньому та масив ідентифікаторів цих пісень.

Кожна сутність представлена в базі даних окремою таблицею. Стовпцями таблиці є атрибути кожної сутності. Дані, що вносяться до таблиці, записуються в новий рядок – кожна клітинка відповідає за встановлене значення певного атрибуту для кожного окремого об’єкту.

Схема описаних сутностей та їх атрибутів представлена на рис. 4.2.

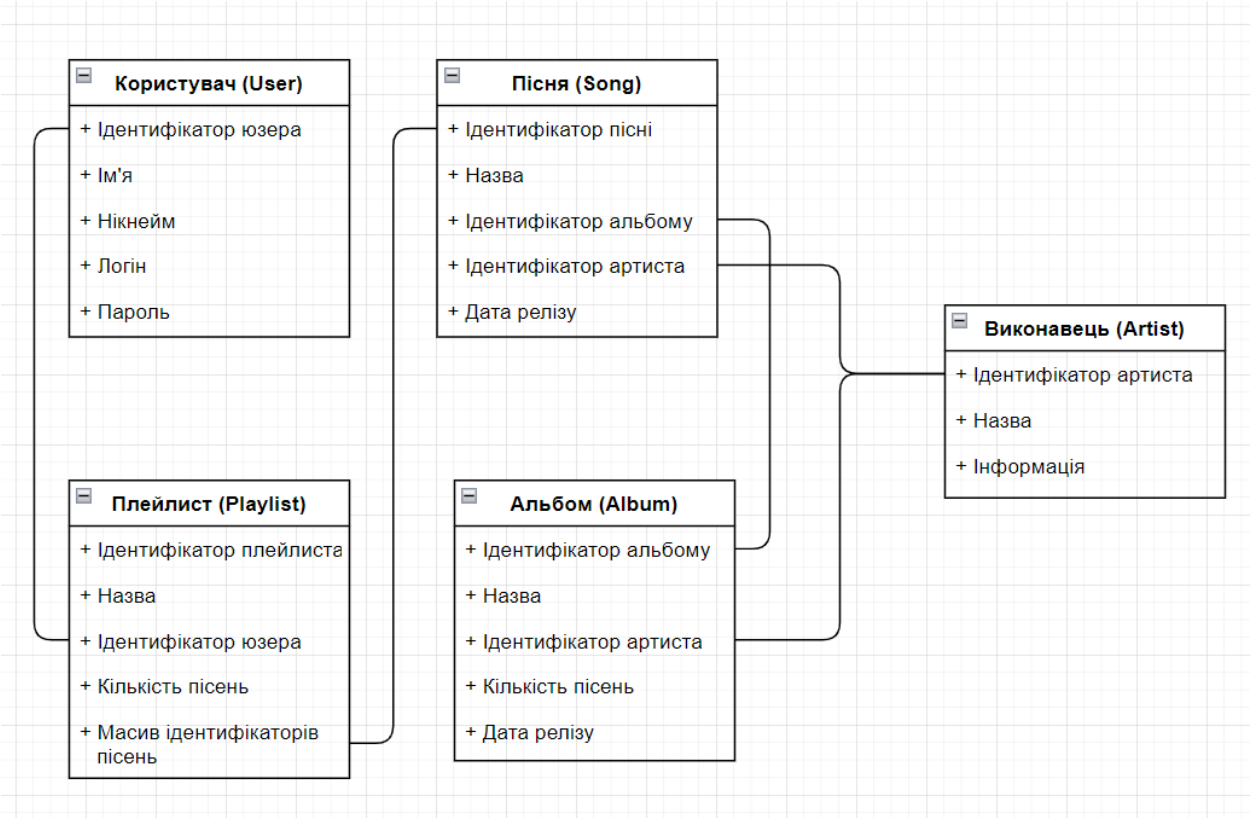



Рис. 4.2. Схема сутностей бази даних web-застосунку

## 4.4. Розробка проекту

### 4.4.1. Створення проекту

Для створення нового проекту необхідно запустити наступну команду в командному рядку:

*vue create* *назва проекту*

Далі буде запропоновано вибрати пресет налаштувань (рис. 4.3). Можна вибрати стандартний пресет (*default*), який додає *Babel* + *ESLint*, або користувацькі налаштування («*Manually select features*») для вибору бажаних можливостей у новому проекті.


Рис. 4.3. Пресети налаштувань на вибір

Налаштування за замовчуванням відмінно підходять для швидкого прототипування нового проекту, тоді як налаштування вручну надає більше опцій, які можуть знадобитися.

Далі буде створена директорія з проектом з обраними налаштуваннями.

Базовий проект містить стандартну сторінку привітання *Vue*.

### 4.4.2. Базові файли

Кожен додаток Vue починається зі створення нового екземпляру додатку за допомогою функції *createApp()* (рис.4.4).

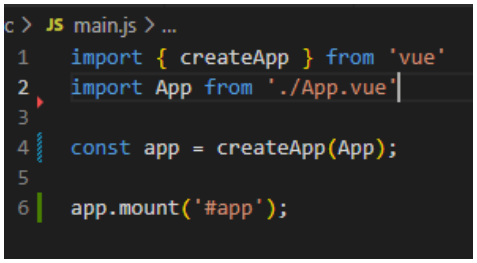



Рис. 4.4. Файл main.js

Екземпляр додатку не буде рендерити щось, допоки метод *mount()* не буде викликаний. Він очікує аргумент-контейнер, який може бути актуальним *DOM* елементом або рядком-селектором.

Головний компонент *App.vue* в базовому проекту *Vue* має код, що зображений на рис. 4.5.

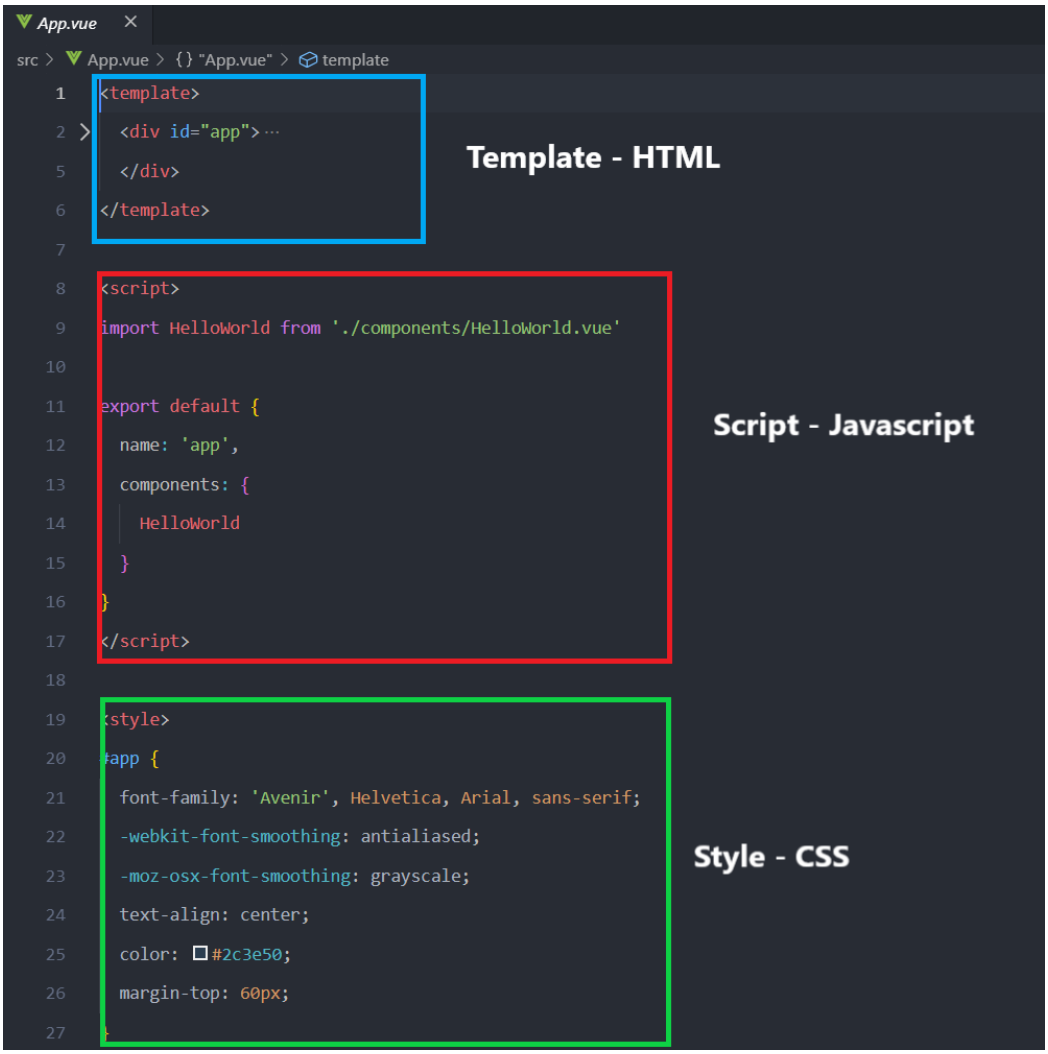



Рис. 4.5. Файл App.vue

Насправді, для додатку, що розробляється, базовий код є непотрібним, тому його необхідно видалити залишивши порожні теги template та script.

### 4.4.3. Налаштування роутингу

Для маршрутизації додатку буде використовуватися *VueRouter*, короткий опис якого зазначений у другому розділі.

Для інсталювання роутеру в проект необхідно виконати наступну команду в терміналі:

*npm install vue-route*

Після того як роутер буде установлено, можна починати його налаштовувати. Перш за все, щоб роутер почав працювати, його необхідно підключити до додатку, що розробляється за допомогою файлу main.js. Для цього необхідно скористатися функцією *app.use()* і передати першим параметром назву плагіну, в даному випадку це “router”.

Далі необхідно створити директорію під назвою router за шляхом ~/src/ і файл *index.js* – у цьому файлі і буде ініціалізований маршрутизатор додатку.

Тепер треба створити змінну, яка буде зберігати в собі усі роути майбутнього веб-додатку (рис.4.6).

```
const routes = [
  { path: "/", name: "home", component: MainComponent },
  { path: "/playlist", name: "playlist", component: Playlist },
  { path: "/playlists", name: "playlists", component: Playlists },
  { path: "/albums", name: "albums", component: Albums },
  { path: "/recent", name: "recent", component: Recent },
  { path: "/favourites", name: "favourites", component: Favourites },
  { path: "/local", name: "local", component: Local },
  { path: "/player", name: "player", component: Player },
  { path: "/:catchAll(.*)", redirect: "/home" },
];
```

Рис. 4.6. Роути

Елементи масиву routes називаються об’єктами маршруту. Об’єкт маршруту має ключ «*path*», що відповідає безпосередньо за шлях. Властивість «*component*» представляє, який компонент буде відображено, коли користувач відвідає даний шлях. Нарешті, властивість «name» представляє назву маршруту.

Перший маршрут має шлях «/», що означає, що це буде базова URL-адреса. Особливий синтаксис останнього шляху буде переадресовувати (властивість «*redirect*») з усіх шляхів, що не вказані в даному масиві, на головну сторінку. Це допоможе уникнути помилок.

Після налаштування шляхів необхідно створити екземпляр *Router*, який може використовуватися програмою Vue за допомогою функції *createRouter()* (рис. 4.7.). Рядок «*history*: *createWebHashHistory()*» дозволяє створити хеш-історію.

```
const router = new createRouter({
  history: createWebHashHistory(),
  routes,
});

export default router;
```

Рис. 4.7. Створення екземпляру Router

Останнім штрихом у створенні роутингу буде додавання тегу <router-view> у темплейт додатку. Даний тег можна помістити в будь-яку частину додатку, у випадку проекту, що розробляється, він буде поміщений безпосередньо у головний компонент *App.vue*.

Після усіх маніпуляцій компонент *App.vue* буде мати наступний вигляд (рис. 4.8.):

```
src > App.vue > Vetur > {} "App.vue" > scr
1   <template>
2     <router-view></router-view>
3   </template>
4
5   <script>
6   import './styles/style.scss'
7
8   export default {
9     name: 'App',
10  }
11  </script>
```

Рис. 4.8. Компонент App.vue

### 4.4.4. Створення сховища стану

Для початку роботи з патерном керування станом VueX необхідно інсталювати його в проект. Для цього треба запустити наступну команду в терміналі:

*npm install vuex@next –save*

У центрі кожної програми *Vuex* знаходиться store (сховище або стор). Сховище – це, по суті, контейнер, який зберігає стан програми. Існує дві речі, які відрізняють сховище *Vuex* від звичайного глобального об’єкта:

1. Стори Vuex реактивні. Коли компоненти *Vue* отримують з нього стан, вони реактивно й ефективно оновлюватимуться, якщо стан сховища зміниться;
2. Не можна напряму змінити стан сховища. Єдиний спосіб змінити стан стору – це явне внесення мутацій. Це гарантує, що кожна зміна стану залишає запис, який можна відстежити, і дає змогу використовувати інструменти, які допомагають розробникам краще розуміти код програми.

Створення стору починається з створення директорії для нього. Тож було створено каталог під назвою store за шляхом ~/src/ і файл *index.js* – у цьому файлі і буде налаштоване сховище стану додатку.

За допомогою функції *createStore()* необхідно створити нове сховище стану додатку (див. рис. 4.9.). В якості аргументів у функцію треба передати об’єкт з початковими значеннями стану, а також об’єкт з мутаціями (спеціальні функції, що призначені для зміни стану у сховищі).

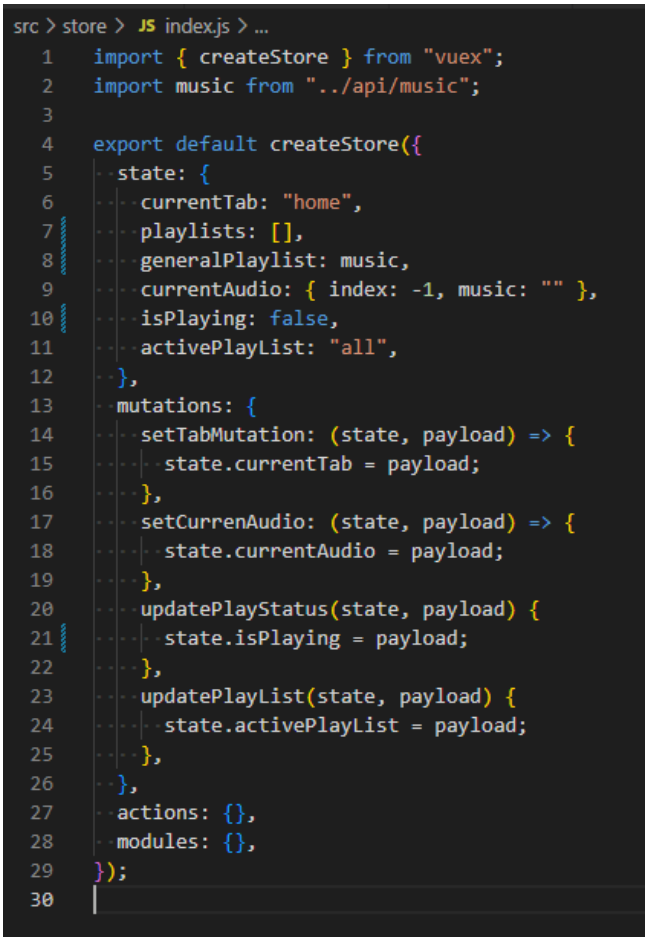



Рис. 4.9. Сховище стану index.js

Тепер, коли сховище стану створено, дуже зручно відслітковувати стан додатку за допомогою встановленого раніше розширення для браузеру *Vue.js devtools*. Щоб скористатися даним розширенням, необхідно відкрити *DevTools*, для цього можна натиснути кнопку F12. Далі необхідно перейти на вкладку *Vue* – це і є вікно інструменту *Vue.js devtools*.

Сховище стану додатку в *Vue.js* devtools зображено на рис. 4.10. На правій панелі можна побачити створений раніше об’єкт з початковими значеннями стану. В ході роботи над додатком ці значення будуть змінюватися.

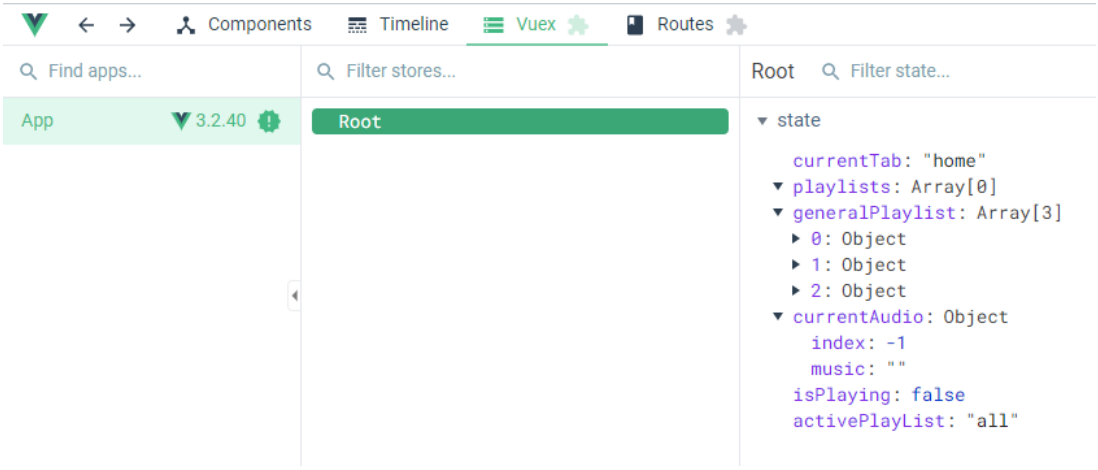



Рис. 4.10. Сховище стану в Vue.js devtools

### 4.4.5. Створення навігації

Для навігації буде слугувати компонент *Sidebar*. На рис. 4.11 зображений html-темплейт для даного компоненту.

Компонент містить в собі тег <img>, що ссилається на файл логотипу додатку, і список з пунктами меню і заголовків до них. Для динамісного виведення пунктів меню була використана спеціальна директива *v-for* [20]. При використанні цієї директиви обов'язковим є використання атрибуту *key*.

```
<template>
  <div class="sidebar">
    <div class="sidebar__content">
      <div class="logo">
        <img src="../../assets/Logo.svg" alt="RHYTHM logo">
      </div>
      <div v-for="(list, index) in navList" :key="index" class="sidebar__navigation navigation">
        <ul class="navigation__list">
          <p class="navigation__title">{{ list.title }}</p>
          <li v-for="(item, index) in list.items" :key="index" class="navigation__item"
            :class="currentTab == item.tab ? 'active' : ''">
            <a @click="changeTab(item.tab, item.routePath)">
              {{ item.itemTitle }}
            </a>
          </li>
        </ul>
      </div>
    </div>
  </div>
</template>
```

Рис. 4.11. Темплейт для Sidebar.vue

При натисканні на пункт меню спрацьовує метод *changeTab()*, що в якості опцій приймає *tab* (назва вкладки, на яку необхідно перейти), *routePath* (безпосередньо шлях вкладки, на яку необхідно перейти). Даний метод змінює шлях веб-додатку за допомогою бібліотеки *Vue Router*.

```
methods: {
    changeTab(tab, routePath) {
        if (this.currentTab != tab) {
            this.$router.push({path: tab})
            this.$store.commit("setTabMutation", routePath);
        }
    },
},
```

Рис. 4.12. Метод для навігації

### 4.4.6. Створення компоненту MainComponent

Даний компонент буде певною обгорткою для усіх сторінок та буде містити компонент меню і елемент для динамічного рендеру компонентів <component>.

Компонент <component> є синтаксичним цукром у шаблонах. Він не є справжнім компонентом і його не можна імпортувати. Це «мета-компонент» для відображення динамічних компонентів. Справжній компонент для рендеру визначається вхідним параметром *is*. Значення вхідного параметра *is* має бути рядок з іменем *HTML*-тега або ім’ям компонента.

Тег <component> дозволяє створити *SPA* додаток і не використовувати різні компоненти для окремих подібних по структурі сторінок.

У конкретному випадку веб-додатку, що розробояється, на місці тега <component> буде рендиритися активний розділ додатку, наприклад «Головна» сторінка.


Рис. 4.12. Файл MainComponent.vue

### 4.4.7. Створення компоненту Головної сторінки

Як вже було зазначено при проектуванні інтерфейсу Головної сторінки, вона буде складатися з компоненту обкладинки, а також списку рекомендацій.

Обкладинка – це слайдер. В мережі вже є багато готових рішень для каруселей, тому було вирішено використати готовий пакет для цього. Програмна реалізація компоненту *Cover* зображена на рис. 4.13.

```vue
<template>
  <div class="cover">
    <carousel :autoplay="true"
              :perPage="1"
              :navigationEnabled="true">
      <slide v-for="item in coverList" :key="'cover-' + item.composition">
        <a :href="item.url" class="cover__item item">
          <img :src="item.imgUrl" :alt="item.composition">
          <div class="item__data">
            <p class="item__performer">{{ item.performer }}</p>
            <p class="item__composition">{{ item.composition }}</p>
          </div>
        </a>
      </slide>
    </carousel>
  </div>
</template>

<script>
import { Carousel, Slide } from 'vue-carousel';

export default {
  name: 'cover',
  components: {
    Carousel,
    Slide
  },
  computed: mapState({
    coverList: state => state.coverList,
  }),
}
```

Рис. 4.13. Компонент Cover

В свою чергу компонент рекомендацій складається з двох інших блоків – безпосередньо рекомендації, а також список нещодавно прослуханих композицій.

Програмна реалізація даного компоненту наведена на рис. 4.14 – 4.15.

```vue
<template>
  <div class="recommendations">
    <recent :recentList="recentList" />
    <for-you :forYou="forYou" />
  </div>
</template>
```

Рис. 4.14. Темплейт компоненту Recommendations

```vue
<script>
import Recent from '/components/recent/Recent.vue';
import ForYou from '/components/forYou/ForYou.vue';

export default {
  name: 'cover',
  components: {
    Recent,
    ForYou
  },
  computed: mapState({
    recentList: state => state.recentList,
    forYou: state => state.forYou,
  }),
}
```

Рис. 4.15. Скрипт компоненту Recommendations

### 4.4.1. Створення компоненту Плеєру

Основна ідея проекту реалізована у компоненті *Player*. Даний компонент є найбліш об’ємним і найбільш складним в усьому додатку і саме тому код програмної реалізації до нього наведено у додатку А.

При ввімкненій композиції знизу на усіх сторінках буде відображатися міні-версія плеєру, в якій достіпні дії перемикання композицій, коротка інформація про час композиції, її обкладинку, назву та виконавця.

В правій частині міні-плеєру знаходиться кнопка для розгортання компоненту. При натисканні на цю кнопку відкриється повна версія плеєру, що умовно поділений на дві колонки: в лівій частині знаходиться обкладинка активної комппозиції, а справа список активного плейлисту.

Зовнішній вигляд плеєру з активною композицією наведено нажче на рис. 4.16.

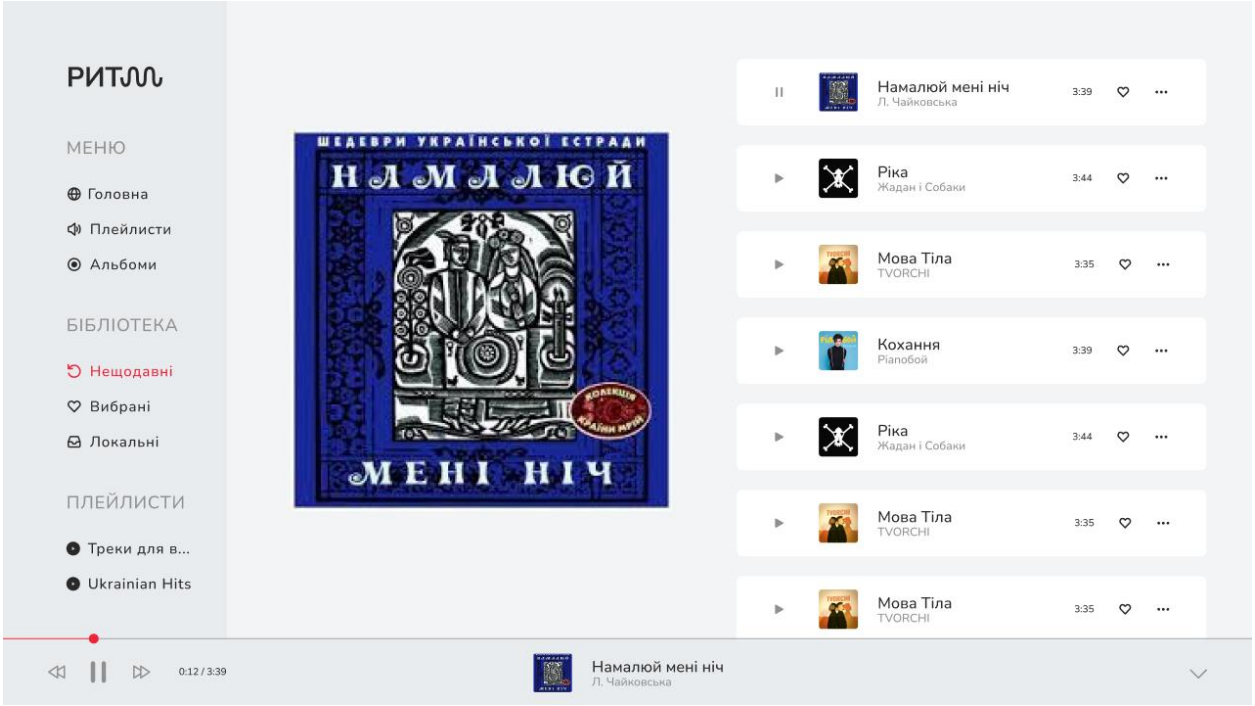



Рис. 4.16. Компонент Player

## 4.5. Тестування

Перед тим як випускати розроблений веб-додаток необхідно провести ряд заходів, спрямованих на пошук і виправлення помилок у веб-дизайні, верстці та покращення користувацького досвіду.

Тестування веб-додатку – це комплекс послуг, що вміщує в собі різноманітні тестування веб-додатку.

### 4.5.1. Основні види тестування сайтів

Задля отримання якісного продукту, необхідно проводити всебічне тестування додатків, саме тому існує декілька видів тестування. Основними видами тестування сайтів є:

– функціональне тестування;
– тестування юзабіліті (перевірка зручності користування);
– тестування сумісності (конфігураційне тестування);
– тест на продуктивність;
– перевірка безпеки.

### 4.5.2. Функціональне тестування

Функціональне тестування сайту – це один з найбільш масштабних видів робіт, з якого все починається. Основна мета – переконатися в коректності роботи основних функцій сайту. Чек-лист на цьому етапі виглядає наступним чином:

– перевірка посилань (внутрішніх, вихідних, ізольованих і т. д.);

– перевірка працездатності форм – в подальшому дозволить взаємодіяти з користувачами і отримувати від них дані;

– тест файлів *cookie* – необхідний для підтримки сеансів з авторизацією;

– валідація *HTML/CSS* – перевірка на відсутність серйозних синтаксичних помилок і доступу ресурсу для різних пошукових систем;

– перевірка бази даних – тут потрібно простежити за правильністю виконання запитів, а також вилучення та оновлення даних.

В результаті проведення даного виду тестування не було виявлено непрацездатних елементів чи будь який неточностей. Можна зробити висновок, що всі перечислені перевірки веб-додаток успішно проходить.

### 4.5.3. Тестування додатку на сумісність

На цьому етапі перевіряється адаптивність сайту. Це так зване нефункціональне тестування, яке дозволяє побачити, наскільки коректно відображається ресурс на різних розширеннях, браузерах і пристроях.

Проблематика даного виду тестування заключається в тому, що не всі користувачі оновлюють версії браузера чи операційної системи, а у веб-розробка постійно покращується, розробляються нові інструменти, які не завжди підтримуються старими версіями ПО.

Висновок по синхронізації з браузером *Google Chrome*: веб-додаток пройшов всі тести. Додаток коректно відображається та працює так, як було задумано. Працездатність підсилює ще те, що додаток відпрацьовує коректно

при усіх тестах, які входять в стандартний набір тестів веб-додатків в браузері Google Chrome.

Також було проведено тестування додатку за допомогою браузерів *Mozilla Firefox і Safari.* Результати тестування аналогічні до тестування в браузері *Google Chrome.*

### 4.5.4. Тестування веб-ресурсів: юзабіліті

Перевірка зручності використання допомагає оцінити сприйняття дизайну сайту і його функціонал користувачем. Сайт повинен бути послідовним і чітко структурованим.

На цьому етапі критерії перевірки включають такі пункти:

– навігаційне тестування – перевірка доступу до меню, сторінок, блоків, полів і кнопок. Важливо, щоб структура була логічною і зрозумілою;

– перевірка контенту – перевіряється граматика і орфографія, розміщення заголовків, а також відповідність розмірів зображень;

– комфорт користування – перевірка структури і наявності зайвих елементів.

В результаті проведення даного виду тестування можна зробити висновок, що всі перечислені вище перевірки веб-додаток успішно проходить: навігаційне меню знаходиться в постійному доступі і структуроване, контент не містить граматичних чи орфографічних помилок, користуватися додатком комфортно, так як при розробці були залучені уже відомі користувачам елементи.

# ВИСНОВОК ДО РОЗДІЛУ 4

Четвертий розділ випускної кваліфікаційної роботи присвячений опису програмної реалізації веб-додатку, а також його тестуванню.

В даному розділі описано дії, що необхідно виконати перед початкою розробкою веб-додатку. Для початку розробки попередньо були встановлені такі інструменти: редактор коду *VS Code*, розширення *Volar* для редактору коду *VS Code* для зручності написання кода за допомогою фреймвоку *Vue.js*, додаток *GitHub Desktop* для пришвидшення роботи з системою контролю версій *Git*, актуальна версія *Node.js*, розширення для браузера *Vue.js devtools*.

Було надано файлову структуру проекту з коротким описом призначення усіх найважливіших директорій і файлів.

Безпосередньо написання будь-якої програмної реалізації починається з встановлення проекту. Фреймворк *Vue* дозволяє швидко створити проект з стандартними налаштуваннями за допомогою виконання команди *vue create* в командному рядку. Після виконання цієї команди буде створено базовий проект *Vue*.

Для розробки користувацького інтерфейсу, проектування якого було описано в розділі 3, було застосовано технології і засоби описані в другому розділі. Це дозволило розробити зручний та інтуїтивно зрозумілий користувацький інтерфейс.

Для опису логіки функціонування web-додатку використано мову програмування *JavaScript*, фреймворк *Vue* та його бібліотеки.

Важливою частиною розділу стало тестування розробленого додатку, перевірка веб-додатку на його працездатність. Було проведено повний стек стандартних тестів веб-додатку, результати яких були успішно пройдені.

# ВИСНОВКИ

Музика є одним з найдревніших видів мистецтва і з розвитком технологій стала невід'ємною частиною життя більшості людей. Як і інші сфери діяльності людства і будь-який вид мистецтва, музика розвивається і тісно сплітається з іншими областями людської діяльності та сучасними технологіями (у тому числі і з мережею інтернет).

На сьогоднішній день розважально-музичний ринок заполонили такі гіганти як *YouTube Music*, *Apple Music*, *Spotify*.

В результаті проведеного аналізу додатків-аналогів було виявлено, що практично неможливо знайти музичні веб-додатки малого формату, а тим паче додатки від українських розробників.

Ґрунтуючись на потребах та ритмі сучасного суспільства та задля задоволення його потреб було прийнято рішення розробити браузерний музичний плеєр.

На початковому етапі роботи було проаналізовано наукову та методичну літературу, що допомогло поглибити знання з проектування та розробки web-додатків, а також вдосконалити навички використання мови програмування JavaScript.

Проведено аналіз додатків-аналогів, після якого було сформовано представлення про web-сервіси музичного типу. Результатом проведеного аналізу стали визначені основні вимоги до програмної реалізації кваліфікаційної роботи, з'явилося основне бачення розробки користувацького інтерфейсу і подальші етапи розробки.

Всі дані, отримані на основі аналізу, дозволили коректною та якісно спроектувати і реалізувати веб-додаток .

Було докладно розглянуто і описано такі технології і засоби як графічний редактор *Figma*, *HTML*, *CSS*, мова програмування *JavaScript*, фреймворк *Vue,* що базується на мові програмування *JavaScript*, бібліотеки *Vue Router* та *VueX,* система контролю версій Git. Розглянуто найпопулярніші

фреймворки для web-розробки і проведено їх порівняльний аналіз, результати якого зведені в таблицю.

Розглянуті та використані технології і засоби для розробки проекту показали, що їх правильний підбір може значно спростити процес проектування та розробки, а також підвищити їх якість.

Розглянуто поняття *UI* та *UX* дизайну, описано їх відмінності та основні концепції. Розроблено користувацький інтерфейс для усіх елементів та сторінок web-додатку.

В даній роботі було розглянуто основні можливості фреймворку *Vue.js*, а саме: робота з props, що дає змогу розробляти багаторазові компоненти; роутинг, який дозволяє динамічно налаштовувати *url*-адреси; загальне сховище станів; директива *v-for*, що дозволяє відображати компоненти як списки; *v-if* – умовні оператори, що дають можливість динамічно рендерити компоненти або елементи на сторінці.

Розроблений web-застосунок протестований на наявність помилок в програмному коді та неточностей, і був покращений, за допомогою їх усунення.

Програмний продукт спроектовано та розроблено з урахуванням можливостей майбутнього розширення функціоналу.

В майбутньому розроблений web-додаток можна покращити за допомогою впровадження наступних функцій:

– покращити функціональність плеєру;

– розробити адміністративну панель для зручності адміністрування веб-додатку.

Досягнуто усіх цілей, виконано поставлені задачі, мета роботи досягнута, а програмний продукт готовий до використання.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. DIGITAL 2022: GLOBAL OVERVIEW REPORT [Electronic resource] – Access mode: https://datareportal.com/reports/digital-2022-global-overview-report
2. Музика [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki
3. Вітт Стівен. Як музика стала вільною. Цифрова революція та перемога піратства – Київ : Наш Формат, 2017 р. – 360 ст.
4. Розвиток цифрової музики [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/515738/
5. Jon Duckett. HTML & CSS: Design and Build Web Sites - Indianapolis, Indiana, 2011 – s. 514.
6. Основні вимоги до дизайну сайту [Електронний ресурс] – Режим доступу: https://ua.waykun.com/articles/osnovni-vimogi-do-dizajnu-sajtu.php
7. The State of UX Design in 2022 [Electronic resource] – Access mode: https://www.wix.com/blog/2021/12/ux-design-trends/
8. Синтаксичний цукор – Вікіпедія [Електронний ресурс]. – 2022. Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/
9. Лекція 3. Основні поняття мови HTML та структура документів [Електронний ресурс] – Режим доступу: https://sites.google.com/site/vivcaemowebdizajndistancijno/html/lekcia-3-osnovni-ponatta-movi-html-ta-struktura-dokumentiv
10. Переваги стилів [Електронний ресурс] – Режим доступу: https://cutt.ly/3tzwAmn
11. An Introduction to JavaScript [Electronic resource] – Access mode: https://javascript.info/intro
12. Найпопулярніші Front-end Framework [Електронний ресурс] – Режим доступу: https://stackdiary.com/front-end-frameworks/

13. Front-end Frameworks [Electronic resource] – Access mode: https://2021.stateofjs.com/en-US/libraries/front-end-frameworks

14. Бібліотека VueX [Електронний ресурс] – Режим доступу: https://vuex.vuejs.org/

15. GIT [Електронний ресурс] – Режим доступу: https://git-scm.com/book

16. UI/UX design [Електронний ресурс] – Режим доступу: https://www.coursera.org/articles/ui-vs-ux-design

17. Website vs web application: what’s the difference and what to choose? [Electronic resource]. Access mode: https://lanars.com/blog/website-vs-web-applicationwhat-s-the-difference-and-what-to-choose

18. What Is a Single Page Application and Why Do People Like Them so Much? [Electronic resource] – Access mode: https://www.bloomreach.com/en/blog/2018/07/whatis-a-single-page-application.html

19. Умовний рендеринг | Vue.js [Електронний ресурс] – Режим доступу: https://ua.vuejs.org/guide/essentials/conditional.html

20. Рендеринг списків | Vue.js [Електронний ресурс] – Режим доступу: https://ua.vuejs.org/guide/essentials/list.html

# ДОДАТКИ

**Додаток А**

## Програмна реалізація компоненту *Player*

```
<template>
  <section class="active-music">
    <div class="logo" :class="isPlay ? 'logo-animation' : ''">
      <font-awesome-icon icon="music" />
      <div id="dummy"></div>
    </div>
    <span class="active-music-name" >
      {{ getActiveMusic.name }}

      <button class="btn-like" :class="checkBookMarkMusic ? 'liked' :
''"  @click="toggleLikeBtn" v-if="getActiveMusic.index !== -1">
        <font-awesome-icon icon="heart" />
      </button>
    </span>
    <span class="active-music-artist">{{ getActiveMusic.artist
}}</span>
    <div class="active-music-status">
      <input type="range" value="0" class="music-range"
ref="music_range" />
      <div class="music-times">
        <span class="active-time"
ref="currentTime">{{getCurrentTime(currentTime) }}</span>
        <span class="all-music-time" ref="duration">{{
getCurrentTime(durationTime) }}</span>
      </div>
    </div>
    <div class="active-music-options">
      <button class="btn btn-loop" :class="isLoop ? 'btn-underline' :
'' " @click="loopMusicToggleBtn">
        <font-awesome-icon icon="redo-alt" />
      </button>
      <button class="btn btn-prev" @click="prevMusic">
        <font-awesome-icon icon="step-backward" />
      </button>
      <button class="btn btn-play" @click="playToggleBtn">
        <font-awesome-icon :icon="['fas', 'pause-circle']" v-
if="getPlayingStatus"/>
        <font-awesome-icon icon="play-circle" v-else />
      </button>
      <button class="btn btn-next" @click="nextMusic">
        <font-awesome-icon icon="step-forward" />
      </button>
      <button class="btn btn-random" @click="randomMusicToggleBtn"
:class="isRandom ? 'btn-underline' : ''">
        <font-awesome-icon icon="random" />
```

```
      </button>
    </div>
    <div class="volume">
      <font-awesome-icon icon="volume-up" />
      <input type="range" value="100" class="volume-range"
ref="volume_range" />
    </div>
  </section>
</template>

<script>
import globalMixins from "@/mixins/globalMixins";
export default {
  name: "Player",
  mixins: [globalMixins],
  data() {
    return {
      player: new Audio(),
      musics: [],
      activeMusic: null,
      musicStatusKod: 0,
      isPlay: false,
      isLoop: false,
      isRandom: false,
      currentTime: 0,
      durationTime: 0,
    };
  },
  computed: {
    getActivePlayList() {
      return this.$store.state.activePlayList;
    },
    checkBookMarkMusic() {
      return
this.getBookMarksIndex.includes(this.getActiveMusic.index);
    },
  },
  watch: {
    getActivePlayList() {
      this.updatePlayList();
    },
    getPlayingStatus(newStatus) {
      this.musicStatusKod = newStatus ? 2 : 3;
    },
    getActiveMusic(value) {
      this.activeMusic = value;
      this.changeMusic();
      this.musicStatusKod = 1;
    },
    musicStatusKod() {
      this.checkPlayStatus();
```

```
  },
  getBookMarksIndex: {
    handler() {
      this.updatePlayList();
    },
    deep: true,
  },
},
mounted() {
  this.setVolumeFromLocalStorage();
  this.updatePlayList();
  this.addEventListener();
},
methods: {
  playToggleBtn() {
    if (this.musicStatusKod == 0) this.nextMusic();
    if (this.isPlay) {
      this.musicStatusKod = 5;
    } else {
      this.musicStatusKod = 4;
    }
  },
  nextMusic() {
    let newMusic = this.isRandom
      ? this.getNewMusic("random")
      : this.getNewMusic("next");
    this.checkSameMusic(newMusic);
  },
  prevMusic() {
    if (this.musicStatusKod == 0) return;
    let newMusic = this.getNewMusic("prev");
    this.checkSameMusic(newMusic);
  },
  toggleLikeBtn() {
    let index = this.getActiveMusic.index;
    if (index == -1) return;
    if (!this.checkExistsBookMarksIndex(index)) {
      this.$store.commit("addBookMarkMusic", index);
    } else {
      this.$store.commit("removeBookMarkMusic", index);
    }
  },
  loopMusicToggleBtn() {
    this.isLoop = !this.isLoop;
  },
  randomMusicToggleBtn() {
    this.isRandom = !this.isRandom;
  },
  updatePlayList() {
    let playList = this.getActivePlayList;
    if (playList == "all") {
```

```
        this.musics = this.getMusics;
      } else if (playList == "bookmarks") {
        this.musics =
          this.getBookMarks().length > 0 ? this.getBookMarks() :
this.getMusics;
      }
    },
    checkSameMusic(newMusic) {
      if (newMusic.index !== this.getActiveMusic.index) {
        this.setActiveMusic(newMusic);
      } else {
        this.play();
      }
    },
    checkPlayStatus() {
      switch (this.musicStatusKod) {
        case 1:
        case 2:
        case 4:
          this.play();
          this.updatePlayStatus(true);
          this.musicStatusKod = -1;
          break;
        case 3:
        case 5:
          this.pause();
          this.updatePlayStatus(false);
          this.musicStatusKod = -1;
          break;
        default:
          break;
      }
    },
    play() {
      let promise = this.player.play();
      if (promise !== undefined) {
        promise.catch(() => {
          this.player.play();
        });
      }
    },
    pause() {
      this.player.pause();
    },
    restartMusic() {
      this.player.currentTime = 0;
    },
    changeMusic() {
      this.player.src = this.getActiveMusic.file;
      this.$refs.music_range.style.pointerEvents = "auto";
    },
```

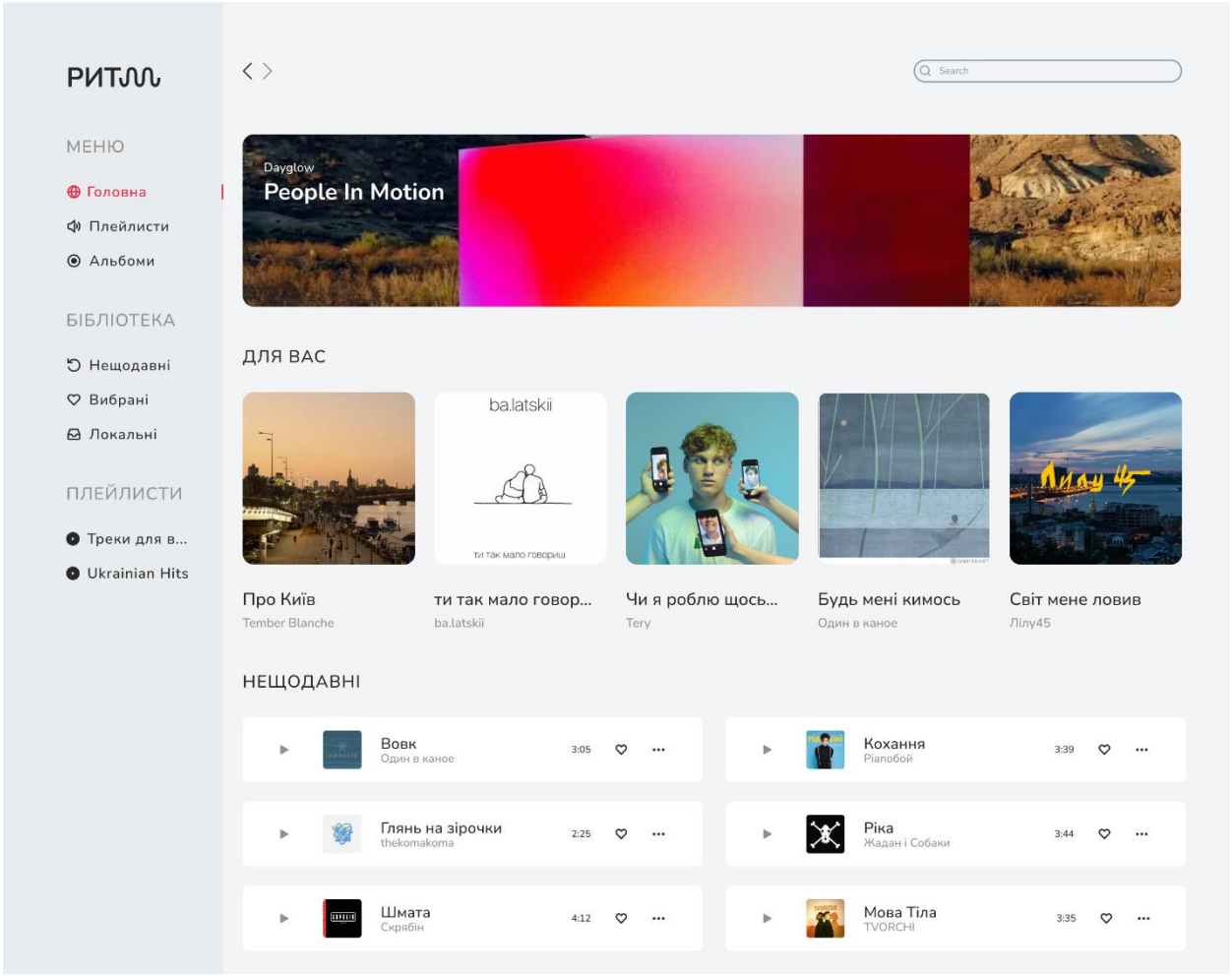
```
    getNewMusic(status = "next") {
      status = String(status).toLowerCase();
      let newIndex = this.getNewMusicIndex(status);
      return this.musics[newIndex];
    },
    getNewMusicIndex(status) {
      let currentIndex = this.musics.findIndex(
        (music) => music.index == this.getActiveMusic.index
      );
      let musicsCount = this.musics.length;
      var newIndex;
      switch (status) {
        case "next":
          newIndex = currentIndex == musicsCount - 1 ? 0 :
currentIndex + 1;
          break;
        case "prev":
          newIndex = currentIndex == 0 ? musicsCount - 1 :
currentIndex - 1;
          break;
        case "random":
          newIndex = Math.floor(Math.random() * musicsCount);
          break;
      }
      return newIndex;
    },
    addEventListener() {
      this.$refs.volume_range.addEventListener(
        "input",
        this.handleVolumeProgress
      );
      this.$refs.music_range.addEventListener(
        "input",
        this.updateMusicProgress
      );
      this.player.addEventListener("timeupdate",
this.handleMusicProgress);
      this.player.addEventListener("canplay", () => {
        this.durationTime = this.player.duration;
      });
    },
    handleMusicProgress() {
      if (this.isMusicOver()) {
        this.pause();
        this.restartMusic();
        if (this.isLoop) {
          this.play();
        } else {
          this.nextMusic();
        }
      }
```

```
      let percent = (this.player.currentTime / this.durationTime) *
100;
      let musicRange = this.$refs.music_range;
      musicRange.value = percent;
      this.updateProgressBColor(musicRange, percent);
      this.currentTime = this.player.currentTime;
    },
    updateMusicProgress(event) {
      let value = event.target.value;
      this.updateProgressBColor(event.target, value);
      this.updateCurrentTime(value);
    },
    handleVolumeProgress(event) {
      let value = event.target.value;
      this.updateProgressBColor(event.target, value);
      this.player.volume = value / 100;
      this.setLocalStorageData("volume", value);
    },
    updateCurrentTime(value) {
      const scrubTime = (value * this.durationTime) / 100;
      this.player.currentTime = scrubTime;
    },
    setVolumeFromLocalStorage() {
      let volume = parseFloat(this.getLocalStorageData("volume")) ||
100;
      let input = this.$refs.volume_range;
      this.player.volume = volume / 100;
      input.value = volume;
      this.updateProgressBColor(input, volume);
    },
    getLocalStorageData(dataName) {
      return localStorage.getItem(dataName);
    },
    setLocalStorageData(dataName, data) {
      localStorage.setItem(dataName, data);
    },
    checkExistsBookMarksIndex(index) {
      let bookmarks = this.$store.state.bookMarksMusicsIndex;
      return bookmarks.includes(index);
    },
    isMusicOver() {
      return Math.floor(this.currentTime) ==
Math.floor(this.durationTime);
    },
    getCurrentTime(time) {
      let hour = Math.floor(time / 3600);
      let minute = Math.floor(time / 60);
      let second = Math.floor(time - (hour * 3600 + minute * 60));
      hour = hour > 9 ? hour : `0${hour}`;
      second = second > 9 ? second : `0${second}`;
      minute = minute > 9 ? minute : `0${minute}`;
```

```
      let formatTime = hour > 0 ? `${hour}:${minute}:${second}` :
`${minute}:${second}`;
      return formatTime;
    },
  },
};
</script>
```

**Додаток Б**

# Зовнішній вигляд головної сторінки



**Додаток В**

## Зовнішній вигляд плеєру