

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОЄКТУВАННЯ ТА РОЗРОБЛЕННЯ ВЕБЗАСТОСУНКУ ДЛЯ
ОБМІНУ ЦИФРОВИМИ РЕСУРСАМИ ДЛЯ СТВОРЕННЯ МУЗИКИ
DESIGN AND DEVELOPMENT OF A WEB APPLICATION FOR EXCHANGING
DIGITAL RESOURCES FOR MUSIC CREATION

Виконав(ла): студент(ка) 4 курсу, групи КНТ-132

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ДЄДОВ Д.В.

(ПРИЗВИЩЕ та ініціали)

Керівник ЛЕОЩЕНКО С.Д.

(ПРИЗВИЩЕ та ініціали)

Рецензент ПОЛЯКОВ М.О.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
 Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2026 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ДЄДОВА Данила Вікторовича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Проектування та розроблення вебзастосунку для обміну цифровими ресурсами для створення музики.

Design and Development of a Web Application for Exchanging Digital Resources for Music Creation

керівник проєкту (роботи) д-р філос., доцент, ЛЕОЩЕНКО Сергій Дмитрович,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026 року
 № 139

2. Строк подання студентом проєкту (роботи) 10 червня 2026 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали та методи. 3. Проектування та розробка системи. 4. Експлуатація та тестування вебзастосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ЛЕОЩЕНКО С.Д., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання « 20 » квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	1 Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

_____ Данило ДЄДОВ
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Сергій ЛЕОЩЕНКО
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
153 с., 15 табл., 90 рис., 3 дод., 26 джерел.

ВЕБЗАСТОСУНОК, МУЗИКА, СОЦІАЛЬНІ МЕРЕЖІ, КЛІЄНТ-СЕРВЕР, ОБМІН, TYPESCRIPT, NODEJS, REACT, TAILWIND, MONGODB.

Об'єкт дослідження – процес розроблення веборієнтованого програмного забезпечення для роботи з цифровими ресурсами для створення музики.

Предмет дослідження – методи, моделі та програмні засоби реалізації вебзастосунків для пошуку, зберігання, завантаження та обміну цифровими музичними ресурсами.

Мета роботи — скорочення часу доступу користувачів до цифрових ресурсів для створення музики шляхом розробки вебзастосунку, що забезпечує централізоване зберігання, пошук, завантаження та обмін такими ресурсами.

Матеріали, методи та технічні засоби: мови програмування Typescript, мова розмітки HTML, мова стилів CSS, фреймворки Nest.js, Next.js, бібліотеки React, Tailwind, база даних MongoDB, хмарне сховище Cloudflare R2, персональний комп'ютер з процесором Intel Core i5 3210M під управлінням операційної системи Windows 10.

Результати. Створено функціональний вебзастосунок, який забезпечить користувачам засоби для додавання на сайт, пошуку, збереження, завантаження та редагування музичних ресурсів.

Висновки. Розроблений вебзастосунок забезпечить користувачам засоби для додавання на сайт, пошуку, збереження, завантаження та редагування музичних ресурсів.

Галузь використання – музична індустрія.

ABSTRACT

Explanatory note to the diploma qualification work of the bachelor: 153 pages, 15 tables, 90 figures, 3 appendixes, 26 sources.

WEB APPLICATION, MUSIC, SOCIAL NETWORKS, CLIENT-SERVER, EXCHANGE, TYPESCRIPT, NODEJS, REACT, TAILWIND, MONGODB.

The object of the study is the process of development of web-oriented software for working with digital resources for creating music.

The subject of the study is methods, models and software tools for implementing web applications for searching, storing, downloading and sharing digital music resources.

The goal of the work is to reduce the time users have to access digital resources for creating music by developing a web application that provides centralized storage, search, download and sharing of such resources.

Materials, methods and technical means: Typescript programming language, HTML markup language, CSS style language, Nest.js, Next.js frameworks, React, Tailwind libraries, MongoDB database, Cloudflare R2 cloud storage, personal computer with an Intel Core i5 3210M processor running the Windows 10 operating system.

The results. Created A functional web application has been developed that will provide users with the means to add, search, save, download and edit musical resources to the site.

Conclusions. The developed web application will provide users with tools for adding, searching, saving, downloading and editing musical resources to the site.

The field of use is the music industry.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок.....	8
Вступ.....	9
1 Аналіз предметної області.....	11
1.1 Ресурси для створення музики.....	11
1.2 Огляд існуючих вебзастосунків які виконують розповсюдження цифрових ресурсів для створення музики.....	15
1.3 Постановка завдання роботи.....	22
1.4 Висновки за розділом 1.....	28
2 Матеріали та методи.....	30
2.1 Вибір та обґрунтування мови програмування.....	30
2.2 Вибір та обґрунтування середовища розробки.....	35
2.4 Вибір та обґрунтування СУБД.....	41
2.5 Вибір та обґрунтування хмарного сховища для зберігання файлів.....	43
2.6 Вибір та обґрунтування платформ для хостингу.....	45
2.7 Висновки за розділом 2.....	47
3 Проєктування та розробка системи.....	50
3.1 Проєктування архітектури системи.....	50
3.2 Опис алгоритмів роботи програми.....	59
3.3 Проєктування бази даних застосунку.....	65
3.4 Графічний користувацький інтерфейс.....	72
3.5 Розробка серверної частини застосунку.....	83
3.6 Розробка клієнтської частини застосунку.....	91
3.7 Висновки за розділом 3.....	101
4 Експлуатація та тестування вебзастосунку.....	104
4.1 Призначення та умови застосування застосунку.....	104
4.2 Опис роботи з програмою.....	104
4.3 Автоматизоване тестування розробленого застосунку.....	106

4.4 Висновки за розділом 4.....115

Висновки.....117

Перелік джерел посилань.....120

Додаток А Технічне завдання.....122

Додаток Б Текст програми.....130

Додаток В Слайди презентації.....142

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
BPM	– Beats Per Minute;
CLI	– Command Line Interface;
CSS	– Cascade Style Sheets;
DAW	– Digital Audio Workstation;
DB	– Database;
GUI	– Graphical User Interface;
HTML	– Hypertext Markup Language;
HTTP	– Hypertext Transfer Protocol;
IDE	– Integrated Development Environment;
JS	– Javascript;
JSON	– Javascript Object Notation;
MVC	– Model View Controller;
REST	– Representational State Transfer;
REST	– Representational State Transfer;
TS	– Typescript;
VST	– Virtual Studio Technology;
ПЗ	– програмне забезпечення;
СУБД	– система управління базою даних.

ВСТУП

Музична індустрія є одним із напрямів цифрової економіки, що динамічно розвивається. Зростання кількості музичних виконавців, продюсерів та інших учасників процесу створення музики супроводжується постійним збільшенням обсягів цифрових ресурсів, які використовуються під час творчої діяльності. До таких ресурсів належать семпли, набори семплів, пресети для VST-плагінів та інші допоміжні матеріали, що спрощують і прискорюють процес створення музичного контенту.

Незважаючи на наявність значної кількості онлайн-платформ для пошуку та розповсюдження музичних ресурсів, більшість із них спеціалізуються лише на окремих типах контенту або обмежують користувачів ресурсами конкретних постачальників. Це ускладнює процес пошуку та отримання необхідних матеріалів і збільшує час, необхідний для роботи з ними.

Об'єктом дослідження є процес розроблення та експлуатації веборієнтованого програмного забезпечення для роботи з цифровими ресурсами для створення музики.

Предметом дослідження є методи, моделі та програмні засоби реалізації вебзастосунків для пошуку, зберігання, завантаження та обміну цифровими музичними ресурсами.

Метою роботи є скорочення часу доступу користувачів до цифрових ресурсів для створення музики шляхом розробки вебзастосунку, що забезпечує централізоване зберігання, пошук, завантаження та обмін такими ресурсами.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати існуючі платформи для розповсюдження цифрових музичних ресурсів;
- розробити дизайн, архітектуру, функціональні можливості вебзастосунку для розповсюдження та обміну музичних ресурсів;
- реалізувати розроблену систему та провести її тестування.

Результатом цієї роботи буде створення функціонального вебзастосунку, який забезпечить користувачам зручні засоби для додавання ресурсів на сайт, їх редагування або видалення, пошуку ресурсів за різними параметрами, їх зберігання та завантаження.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У цьому розділі проводиться докладний аналіз предметної області, що стосується цифрових ресурсів пов'язаних зі створенням музики та способами їх розповсюдження. Він включає огляд основних понять пов'язаних з даною предметною областю, огляд існуючих програмних рішень, їхніх переваг та недоліків, а також формальну постановку завдань роботи.

1.1 Ресурси для створення музики

1.1.1 Історія цифровізації створення музики

У минулому, створення та запис музики на професійному рівні вимагало використання великої кількості дорогого обладнання — мікрофонів, підсилювачів, мікшерних пультів, педалей ефектів, драм-машин, синтезаторів та багатьох інших пристроїв. Але з плином часу, з появою широко розповсюджених персональних комп'ютерів та з поступовим зростанням їх обчислювальної потужності почали з'являться як обладнання яке використовували цифрові програмні засоби для різних цілей (наприклад синтезатори здатні відтворювати короткі скомпресовані аудіо файли для імітації традиційних інструментів), так і безпосередньо комп'ютерні програми для створення та/або обробки аудіо файлів. Наприклад, наприкінці 1980-х та у 1990-х роках популярними були програми які дозволяли через CLI накладати ефекти такі як реверберацію, компресію тощо на аудіо файли, такі як SoX чи Goldwave.

Приблизно в цей же час відбулось створення перших програм які дозволяли створювати повноцінні музичні композиції повністю на комп'ютері. Першими з таких програм були так звані «Трекери», які надавали обмежену кількість каналів звуку здатних програватися одночасно та обмежені можливості конструювання форм звукових хвиль математичним шляхом або використання сильно скомпресованих та обмежених за довжиною аудіо файлів — семплів. Відомим представником цієї категорії програм є ImpulseTraker.

З часом втім, обчислювальна потужність та обсяги пам'яті комп'ютерів досягли точки, в якій програми обробки звуку змогли дозволити собі використання не-скомпресованих аудіо файлів у майже необмеженій кількості, проводячи при цьому обробку звуку ефектами у реальному часі. З появою цих можливостей з'явилась і нова категорія програм для створення музики — DAW (Digital Audio Workstation). Такі програми дозволяють виконувати широкий спектр задач пов'язаних зі створенням аудіо файлів, записом звуку, його обробкою та редагуванням. Відомими ранніми DAW є програми Cubase та Protools [1]. На сьогоднішній день цей вид програм став домінуючим способом створення музики. Серед сучасних DAW можна виділити Ableton Live [2], FL Studio [3], Reaper [4] та багато інших.

1.1.2 Пресети для VST плагінів

Іншим важливим проривом у звукових технологіях стало створення формату VST (Virtual Studio Technology) компанією Steinberg [5]. Цей формат визначає вид програм, які використовуються як плагіни всередині DAW, і найчастіше представляють собою цифрову імітацію фізичного звукового обладнання або інструментів. VST плагіни можна умовно поділити на 3 типи:

- генератори, які при своїй роботі створюють звуковий сигнал; найчастіше це синтезатори, які використовують математичні перетворення, найчастіше адитивний синтез, для створення заданих користувачем форм та комбінацій аудіо хвиль, які далі відтворюються завдяки програмним осциляторам; також велику частину ринку плагінів займають плагіни які представляють собою детальне відтворення фізичних музичних інструментів, зберігаючи в собі записи відтворення звуку цими інструментами і кожному можливому стані з кожним можливим параметром;

- ефектори, які приймають на вхід існуючий звуковий сигнал, проводять певну його обробку та повертають результат; найчастіше такі плагіни

відповідають за ефекти еквалайзера, реверберації, ділею, компресії, але можуть виконувати і будь-яку іншу форму обробки;

– утилітарні плагіни, які виконують будь-які іншу функції, наприклад форми візуалізації звукового сигналу, або створення ланцюгів обробки поєднуючи між собою інші плагіни.

Оскільки багато VST плагінів мають комплексний та багатофункціональний користувацький інтерфейс, майже кожен сучасний плагін містить в собі також функціонал для так званих «пресетів» (presets) – файлів, які зберігають в собі значення кожного параметра стану плагіна, і дозволяють відтворити цей стан в будь який момент використання плагіну. Подібний функціонал походить ще з ранніх аналогових синтезаторів, які довгий час не мали можливості відновити одного разу налаштований стан, змушуючи користувачів запам'ятовувати або вручну записувати значення параметрів. Пізніше це було помічено розробниками синтезаторів та функціонал пресетів було створено, і пізніше перенесено на VST плагіни.

1.1.3 Семплювання та набори семплів

Іншою важливою концепцією необхідною для розуміння предметної області розроблюваного застосунку є семпли (samples) та їх набори (packs).

В контексті музики, семплювання — це повторне використання частини (семплу) фрагменту звукового запису в іншому записі. Семпли можуть містити такі елементи, як ритм, мелодія, мова або звукові ефекти. Семпл може містити лише фрагмент звуку або більшу частину музики, як-от перкусію чи мелодію. При своєму зародженні у 1960-х та 1970-х роках семплювання відбувалось завдяки спеціально створеним пристроям, які дозволяли семплювати лише маленьку кількість коротких записів, які втім можна було зациклювати, дозволяючи створити імітацію довгого звуку через зациклення семплу довжиною лише в кілька секунд. Цю техніку почали активно використовувати у

1970-х здебільшого для створення різноманіття у звуках синтезаторів а також для одноразових звукових записів та шумів.

Пізніше, коли семплери (пристрою для семплювання) стали здатні зберігати та відтворювати більш довгі секції звуку, семпли, особливо записи коротких барабанних соло секцій (drum breaks) стали використовувати як не просто незначне доповнення а як основний елемент композиції у таких музичних жанрах як хіп-хоп та пізніше у значній частині електронної музики.

З розповсюдженням семплювання в створенні музики почали з'являтися і спеціально створені набори семплів — спочатку на касетах, пізніше на CD-дисках, а зараз і просто у цифрових архівах. Наявність таких наборів облегшувала роботу музикантів пропонуючи набори пов'язані за певною тематикою, жанром, категорією змісту тощо.

Також, набори семплів майже завжди гарантували вільні права на комерційне використання, невідмінну від традиційного семплювання окремих композицій вручну, яке вимагає згоди авторів використаних семплів на використання, що часто є майже неможливим для більшості непрофесійних авторів музики.

1.1.4 Види ресурсів в розроблюваному застосунку

З огляду на розділи 1.1.1-1.1.3 можна виділити наступні види цифрових ресурсів для створення музики, можливості обміну якими будуть реалізовані в розроблюваному вебзастосунку:

- пресети для VST плагінів — готові набори налаштувань які дозволяють зберігати та швидко отримувати готові звуки для інструментів, відповідні до ситуації налаштування ефектів;

- семпли — окремі звуки чи відрізки аудіо будь-якого походження, звуки перкусійних (ударних) інструментів або короткі вирізки з інших композицій;

- набори (sample racks) – набори семплів. Найчастіше використовуються як набори різних частин однієї перкусійної системи (drum kits).

1.2 Огляд існуючих вебзастосунків які виконують розповсюдження цифрових ресурсів для створення музики

Незважаючи на відсутність на ринку застосунків які б надавали послуги відносно усіх видів продуктів, існує достатня кількість сайтів які спеціалізуються на тому чи іншому виді. Серед них можна зазначити:

- tracklib: Сайт який надає користувачам доступ до каталогу ліцензованої музики на базі підписки. Користувач може завантажити певну кількість ліцензованих треків з сайту в залежності від рівню підписки, та легально використовувати їх у власній комерційній музиці;

- splice: сайт який здебільшого надає користувачам окремі семпли на базі підписки. Splice укладає контракти з окремими продюсерами, які створюють для нього бібліотеки семплів;

- looperman: сайт на якому зареєстровані користувачі можуть викладати власні семпли та безкоштовно завантажувати семпли інших користувачів;

- cymatics: сайт на якому здебільшого можна купувати окремі набори семплів(drum kits and sample packs);

- pluginboutique: сайт який продає окремі плагіни та набори пресетів; б presetshare: сайт на якому користувачі безкоштовно діляться окремими пресетами один з одним;

- presetshare: сайт на якому користувачі безкоштовно діляться окремими пресетами один з одним.

1.2.1 Tracklib

Tracklib [6] (рис 1.1) є вебзастосунком, який надає користувачам доступ до великої бібліотеки ліцензованих музичних композицій для створення семплів та використання у власних музичних проєктах. Платформа орієнтована переважно на музичних продюсерів, які потребують легального використання фрагментів існуючих треків без порушення авторських прав. Сервіс працює за

моделлю підписки та дозволяє завантажувати визначену кількість треків залежно від обраного тарифного плану.

Перевагами Tracklib є висока якість музичних матеріалів, зручний інтерфейс пошуку композицій, а також можливість легального ліцензування семплів для комерційного використання. До недоліків належить обмеження кількості завантажень у межах підписки та відносно висока вартість ліцензій для окремих користувачів.

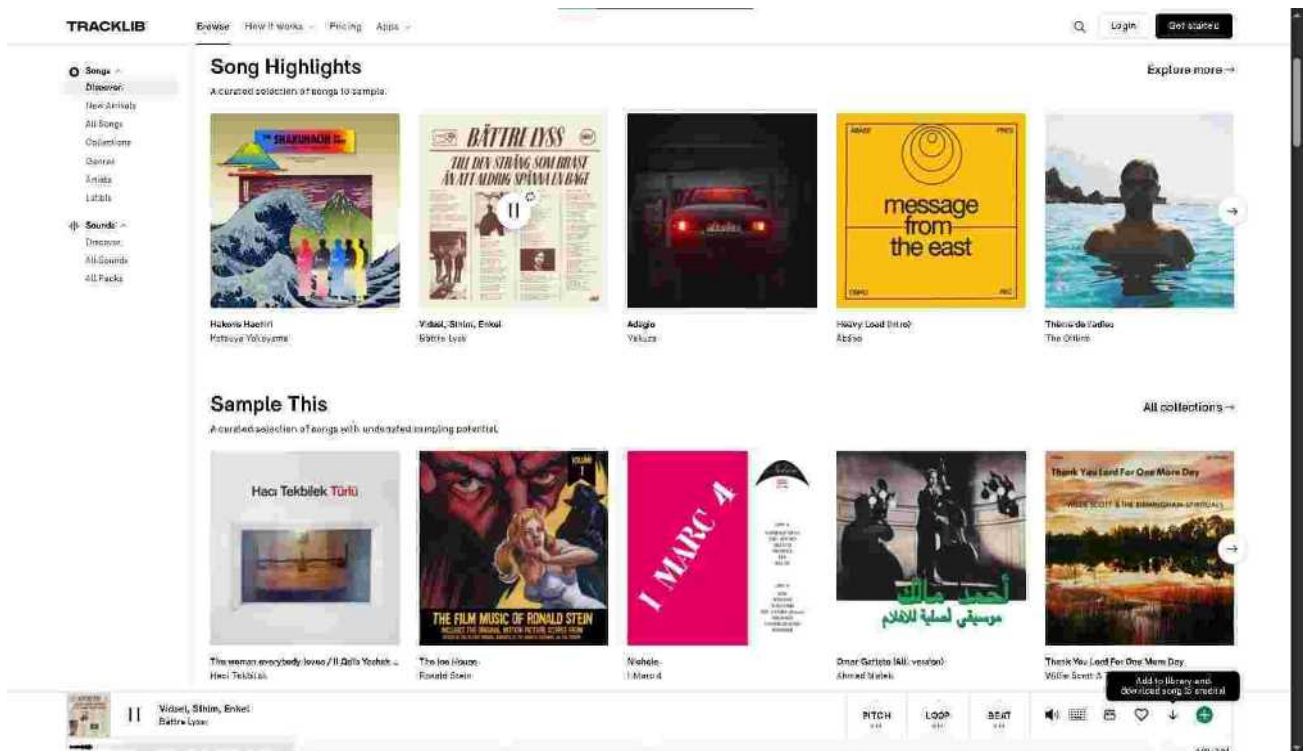


Рисунок 1.1 — Інтерфейс вебзастосунку Tracklib [6]

1.2.2 Splice

Splice [7] (рис. 1.2) це онлайн-сервіс для розповсюдження цифрових ресурсів для музичного виробництва, зокрема семплів, пресетів та плагінів. Застосунок орієнтований на користувачів, яким необхідний швидкий доступ до великої кількості професійних аудіо матеріалів без необхідності купівлі окремих наборів.

Основною перевагою Splice є велика бібліотека високоякісних семплів та тісна інтеграція з популярними DAW-системами, що значно спрощує робочий процес музичного продюсера. Водночас суттєвим недоліком є залежність від підписки, через що доступ до більшості функцій обмежується після її завершення, а також нечітка ліцензійна політика щодо власності семплів, яка має історію випадків, коли два автори використовували один семпл, і одному з них вдавалось досягти юридичних прав на композицію, тим самим блокуючи авторські права другого, не зважаючи на заяви splice про вільне використання.

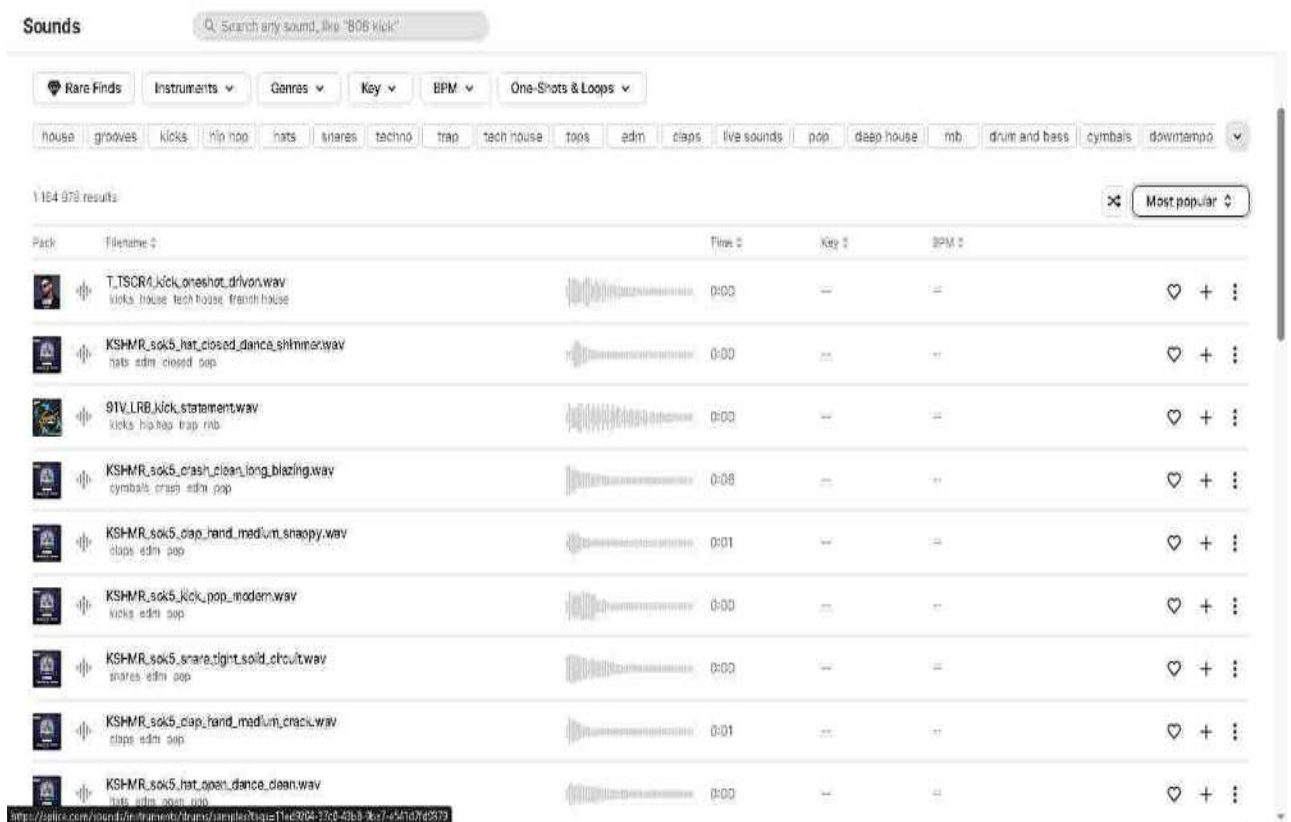


Рисунок 1.2 — Splice [7]

1.2.3Looperman

Looperman [8] (рис. 1.3) представляє собою вебзастосунок для обміну аудіо семплами між користувачами. Зареєстровані користувачі можуть безкоштовно завантажувати власні лупи, вокал та інші звукові матеріали, а також використовувати ресурси, створені іншими учасниками спільноти.

Перевагою даного застосунку є повністю безкоштовний доступ до контенту та активна спільнота користувачів. Проте Looperman має менш сучасний інтерфейс та нерівномірну якість контенту, оскільки матеріали створюються самими користувачами без професійної модерації.

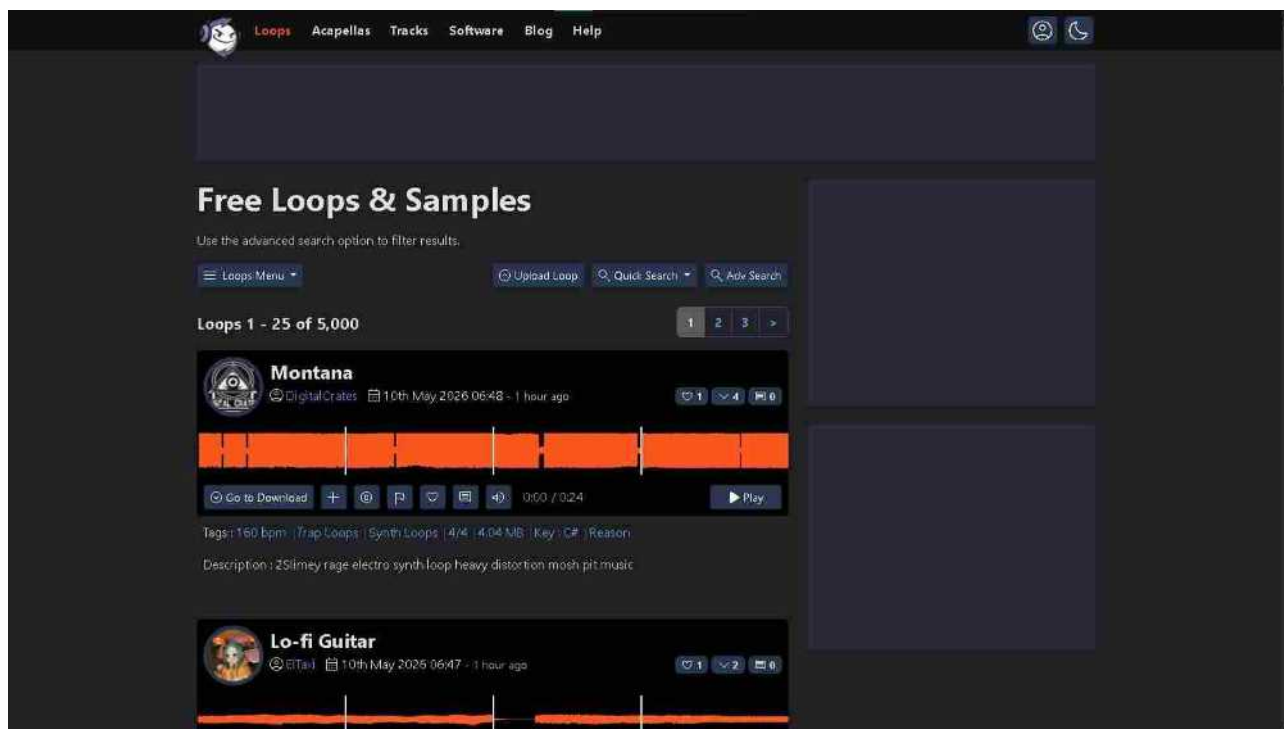


Рисунок 1.3 — Looperman [8]

1.2.4 Cymatics

Cymatics [9] (рис. 1.4) є платформою для продажу цифрових ресурсів для музичного продакшену. Застосунок спеціалізується переважно на наборах семплів, drum kits, MIDI-паках та окремих аудіо плагінах.

Перевагами Cymatics є велика кількість безкоштовних матеріалів, сучасний дизайн інтерфейсу та висока популярність серед музичних продюсерів електронної музики. До недоліків можна віднести орієнтованість переважно на окремі музичні жанри, а також агресивну маркетингову політику сервісу.

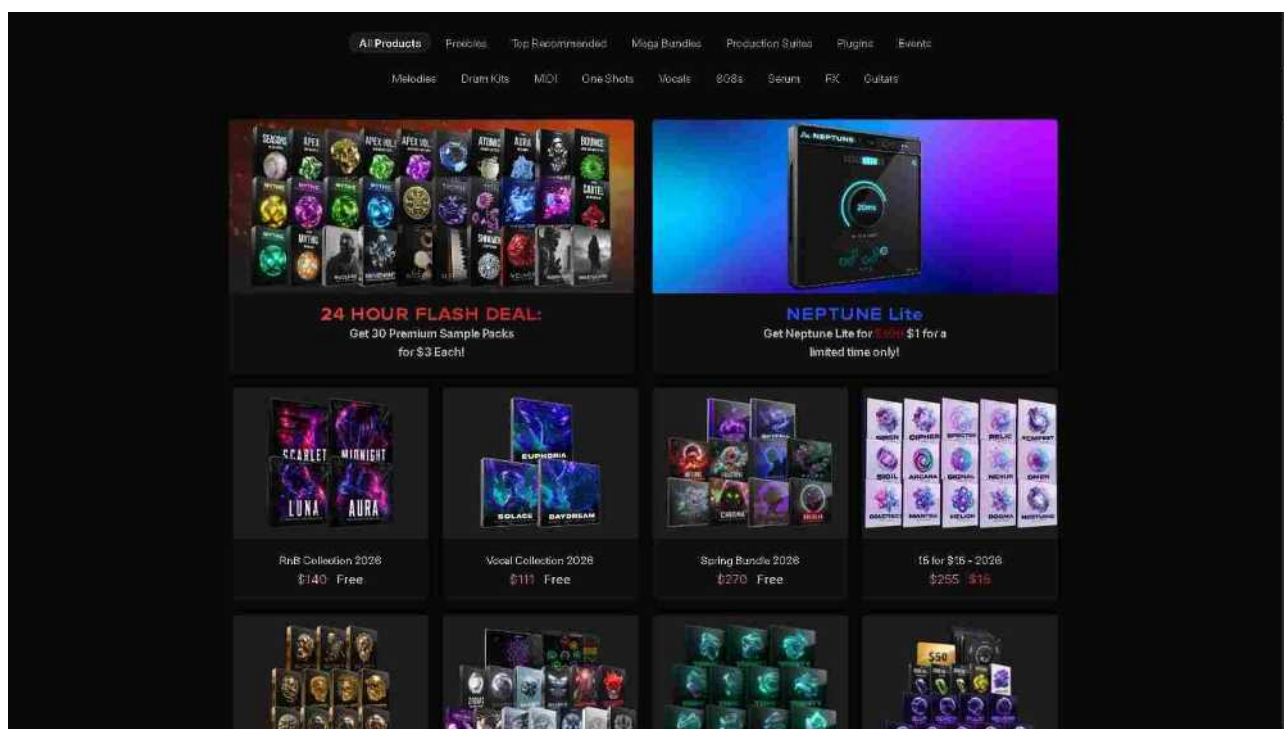


Рисунок 1.4 – Cymatics [9]

1.2.5 Pluginboutique

Plugin Boutique [10] (рис. 1.5) це онлайн-магазин цифрових аудіо плагінів, віртуальних інструментів та пресетів для музичного виробництва. Платформа об'єднує продукти великої кількості розробників та надає можливість купувати програмне забезпечення для різних DAW-систем.

Основними перевагами Pluginboutique є широкий вибір програмних продуктів, регулярні знижки та наявність детального опису й демонстраційних матеріалів для кожного плагіна. Недоліками є складність навігації сайтом, відсутність єдиної системи підписки для доступу до контенту та концентрація лише на програмному забезпеченні.

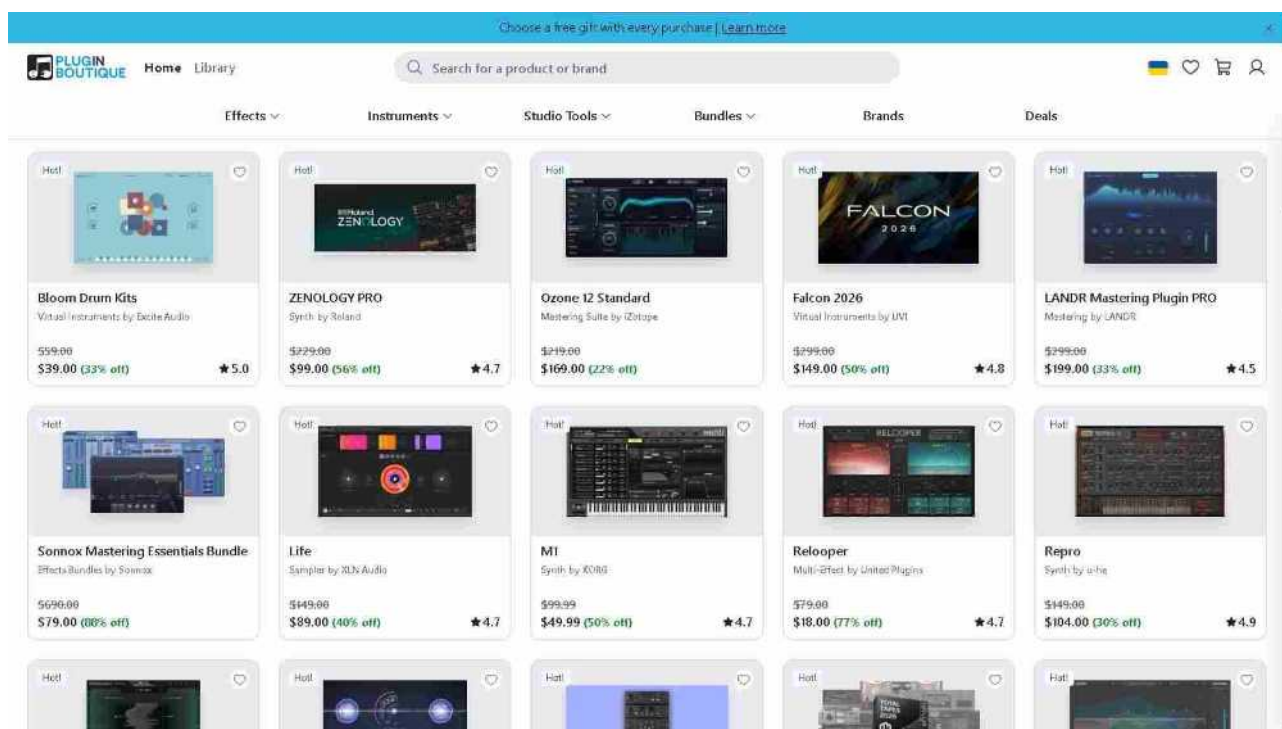


Рисунок 1.5 – Pluginboutique [10]

1.2.6 Presetshare

PresetShare [11] (рис. 1.6) це вебзастосунок для безкоштовного обміну пресетами між користувачами. Платформа дозволяє публікувати та завантажувати пресети для популярних синтезаторів і аудіо плагінів.

Перевагами Presetshare є простота використання, безкоштовний доступ до контенту та наявність спільноти користувачів, які регулярно оновлюють бібліотеку пресетів. До недоліків належать невисокий рівень модерації контенту, відсутність гарантії якості пресетів, обмежений функціонал пошуку та російське походження платформи.

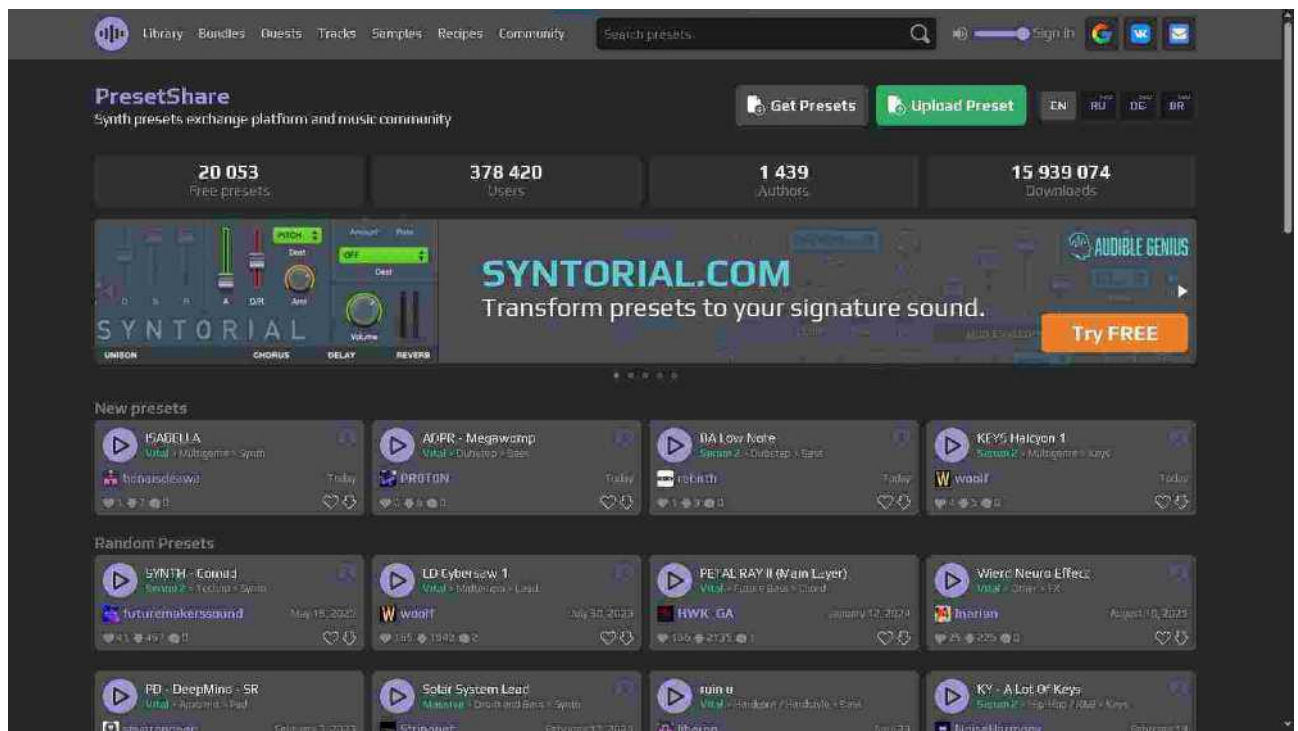


Рисунок 1.6 – Presetshare [11]

1.2.7 Порівняльна таблиця існуючих та розроблюваного програмного забезпечення

У таблиці 1.1 наведено порівняння функціоналу, переваг та недоліків застосунків, розглянутих у розділах 1.2.1 — 1.2.6.

Таблиця 1.1 – Порівняння застосунків для шифрування

Характеристика	Tracklib	Splice	Looperman	Cymatics	Pluginboutique	Presetshare
Цінова політика	Підписка	Підписка	Немає	Окремі продукти, деякі безкоштовні	Окремі продукти, деякі безкоштовні	Немає
Простота використання	Висока	Висока	Низька	Середня	Низька	Середня
Інтегрованість з daw	Висока	Дуже висока	Низька	Низька	Середня	Низька
Тип продукції	Ліцензована треки	Семпли, плагіни, пресети	Семпли	Набори, плагіни	Плагіни, пресети	Пресети, семпли

Продовження таблиці 1.1

Характеристика	Tracklib	Splice	Looperman	Cymatics	Pluginboutique	Presetshare
Розмір бібліотек	Високий	Високий	Низький	Середній	Високий	Середній
Підтримка	Висока	Висока	Маленька	Маленька	Середня	Низька
Якість продуктів	Висока	Висока	Низька	Середня	Середня	Низька

1.3 Постановка завдання роботи

В рамках даної курсової роботи буде розроблено вебзастосунок для розповсюдження цифрової музичної продукції, який забезпечуватиме функціонал для додавання на сайт, редагування або видалення, пошуку, зберігання та завантаження семплів, їх наборів або пресетів для VST-плагінів. Застосунок буде відповідати наступним вимогам:

- зручність та простота: інтерфейс буде інтуїтивно зрозумілим та простим у використанні, що дозволить працювати з системою без спеціальної підготовки і з високою швидкістю;
- облік ресурсів: зберігання та обробка інформації про продукти кожного виду;
- облік користувачів: зберігання інформації про користувачів, їх облікові дані та дані про придбані та/або завантажені ресурси;
- оформлення завантажень: оновлення інформації про користувача після кожного завантаження, а також оновлення статистики про завантажений продукт;
- оформлення додавання ресурсу автором: створення нового продукту автором, завантаження його до сайту та відповідне оновлення даних про автора та продукти у базі даних;
- видалення своїх ресурсів за необхідністю;
- редагування ресурсів доданих на сайт користувачем;
- пошук продуктів: отримання списків ресурсів за назвою та/або різними іншими параметрами які їх характеризують.

1.3.1 Цілі та межі розробки

Мета застосунку: основна ціль застосунку полягає у створенні платформи для авторів музики, яка буде давати можливість ділитися власними, шукати та завантажувати різні види допоміжних матеріалів, таких як семпли, їх набори та пресети. Розроблений застосунок матиме чотири основні розділи `profile`, `explore`, `saved`, `upload`.

Розділ `profile` буде містити статистику та дані про профіль користувача, керування обліковим записом тощо.

Розділ `explore` дозволить користувачам шукати необхідні матеріали за назвою, автором, та набором параметрів матеріалу, сортувати отримані результати та зберігати обрані матеріали. При збереженні матеріалу він з'являється на сторінці «збережене» у профілі користувача, де користувач в свою чергу може завантажити матеріал на свій пристрій. Якщо ж користувач не є зареєстрованим, матеріал одразу пропонується завантажити на комп'ютер.

Розділ `saved` буде містити списки збережених матеріалів, розділених за видом. Цей розділ також надаватиме можливість пошуку, сортування та завантаження. Окрім цього, під час розробки даного розділу буде зроблено спробу створити `drag-and-drop` функціонал, який дозволить перетаскувати збережені файли напряму у середовище створення музики. При вдалій реалізації такого функціоналу, розроблений застосунок дозволить обійти часту проблему з якою стикаються автори, коли завантажені локально матеріали займають десятки гігабайтів пам'яті. Якщо ж зберігати робочі матеріали на сервері розробленого застосунку, фактично використовуючи його як хмарне сховище, користувачі зможуть уникнути цієї проблеми, і зберігати локально лише ті матеріали які реально використовуються, а ті які лише «можливо колись» будуть використані залишити на сервері застосунку.

Розділ `upload` буде містити сторінки для додавання власних матеріалів до платформи. Користувачу буде запропоновано заповнити форму з інформацією про матеріал, або зчитати інформацію з метаданих файлу.

Таким чином, межі проекту можна визначити наступним чином.

Включено:

- реєстрація та логін користувача;
- логін через сторонні провайдери;
- обмежене використання застосунку і гостьовому режимі;
- перегляд статистики профілю;
- пошук матеріалів за вказаними фільтрами та сортування результатів пошуку;
- додавання обраних матеріалів до розділу «збережене»;
- завантаження матеріалів з розділу «збережене» на комп'ютер;
- пошук за фільтрами та сортування збережених матеріалів;
- перетягування файлів з застосунку на комп'ютер користувача, з інтеграцією з середовищами створення музики;
- додавання власних матеріалів до застосунку;
- ввід бажаної інформації про додані матеріали;
- автоматичне зчитування інформації про додані матеріалу з метаданих файлу.

Не включено в даний процес розробки, але можливі на наступних її ітераціях доповнення:

- адаптація застосунку під мобільні пристрої;
- крос-браузерність застосунку;
- інтеграція платних ресурсів від сторонніх поставників;
- монетизацію та/або інтеграцію платіжних системні;
- аналітика матеріалів та користувачів, наприклад популярність;
- користувацький інтерфейс для адміністратора;
- локалізація застосунку для української мови.

1.3.2 Функціональні вимоги вебзастосунку

Авторизація та реєстрація:

- створення облікового запису прив'язаного до самого застосунку;
- створення облікового запису через використання сторонніх провайдерів, таких як google, github, discord тощо;
- вхід до застосунку через дані введені при реєстрації, якщо реєстрацію було проведено через сам застосунок;
- вхід до застосунку через сторонні провайдери.

Додавання матеріалів:

- зареєстрований користувач має мати змогу додавати семпли, набори та пресети;
- при додаванні семплу, користувач може вказати назву, темп, використаний(і) інструмент(и), жанр, темп, тональність, короткий опис, теги; при цьому лише назва є обов'язковою для введення; довжина, розмір файлу та дата створення файлу зчитуватимуться застосунком з метаданих аудіо файлу;
- при додаванні набору, користувач завантажує цілу папку, яка містить аудіо файли та декілька рівнів вкладених папок; альтернативно, користувач може створювати папки на сторінці застосунку, обираючи які файли додавати до якої папки, створюючи структуру набору;
- при додаванні пресету, користувач має вказати назву, VST плагін до якого належить пресет, і опціонально жанр, «інструмент», короткий опис.

Пошук матеріалів:

- користувач може шукати (explore) семпли, набори та пресети;
- користувач може фільтрувати матеріали, шукаючи їх за назвою та/або будь-яким набором параметрів які мають відповідні матеріали, наприклад жанром;
- користувач може сортувати отримані результати, які за замовченням йдуть в порядку від найбільшої до найменшої кількості завантажень, при чому

сортувати можливо за будь-якими з наявних параметрів; Якщо ж певний матеріал відповідає заданим фільтрам, але не містить частини інформації за якою відбувається сортування, такі матеріали будуть розміщені в кінці списку;

– користувач може завантажити будь-який знайдений матеріал; для цього, обраний у розділі explore матеріал можна буде «зберегти», що перенесе його у розділ «збережене», звідки користувач вже може його завантажити.

Профіль користувача:

– користувач має ім'я, зображення профілю, кількість завантажених, кількість доданих матеріалів, дати створення облікового запису, останніх завантажень та додавань, списки завантажених та доданих матеріалів;

– додані раніше матеріали можна видалити з застосунку;

– користувач має змогу перейти до сторінки «збережене», обрати там розділ за типом матеріалу, і отримати список збережених матеріалів, які він може переглядати, фільтрувати, шукати, сортувати, видаляти та завантажувати;

– користувач має змогу перейти до сторінки «мої матеріали», де йому буде надано аналогічних список матеріалів, які створив він сам.

Для створених вимог також було створено візуалізацію у вигляді діаграми прецедентів, зображених на рисунку 1.7.

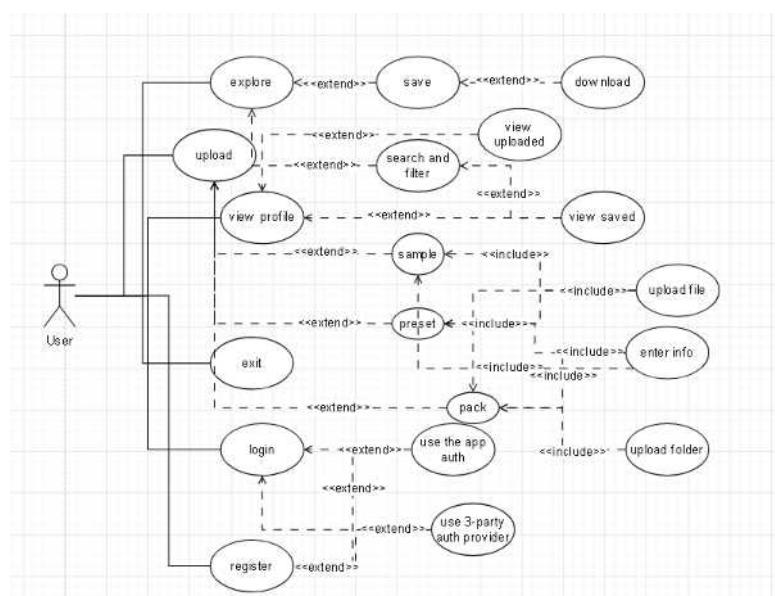


Рисунок 1.7 — Діаграма прецедентів розроблюваного застосунку

1.3.3 Вимоги до інтерфейсу застосунку

У цьому розділі наведено опис вимог до складу сторінок користувацького інтерфейсу розроблюваного застосунку та основні вимоги до їх стилізації.

Основними стилістичними рішеннями, які визначено на даному етапі розробки, є:

- використання темної теми як основної;
- кольором фону сторінок є #333333;
- кольором тексту є #C9C9C9;
- кольором виділених елементів та акцентування є #00F0FF;

– якщо якийсь елемент сторінки потрібно акцентувати, наприклад при наведенні курсором, колір тексту змінюється на #2C2C2C а колір фону елементу змінюється на #00F0FF;

- усі заголовки використовують шрифт Suprapower;
- увесь звичайний текст використовує шрифт Programme.

Список сторінок який планується розробити:

- /explore/samples;
- /explore/packs;
- /explore/presets;
- /saved/samples;
- /saved/packs;
- /saved/presets;
- /upload/sample;
- /upload/pack;
- /upload/preset;
- /created/samples;
- /created/packs;
- /created/presets;

- /profile;
- /saved;
- /uploaded;
- /login;
- /register.

1.3.4 Вимоги до продуктивності та експлуатації

У розроблюваному застосунку існує 4 чинники, які можуть суттєво вплинути на швидкість роботи і погіршити досвід користувача. Ними є швидкість виконання запитів до бази даних, швидкість додавання файлів, швидкість завантаження файлів з застосунку на комп'ютер та швидкість рендеру сторінки, яка може погіршуватись на сторінках, які відображають багато аудіо файлів, або використовують візуалізацію звукових хвиль файлів.

У ході розробки планується виділити час на оптимізацію цих процесів, застосувати засоби оптимізації запитів до бази даних, рендеру елементів сторінок, швидкість обробки файлів та комунікації сервера з клієнтом.

Щодо експлуатації, застосунок розроблюється для використання у браузерях на основі chromium на екранах розміру десктопних комп'ютерів.

1.4 Висновки за розділом 1

У даному розділі було проведено аналіз предметної області, пов'язаної з цифровими ресурсами для створення музики та способами їх розповсюдження через вебзастосунки. Розглянуто історію цифровізації процесу створення музики, розвиток DAW, технологій семплювання та VST-плагінів, а також визначено основні види ресурсів, які використовуються сучасними музичними продюсерами — семпли, набори семплів та пресети.

У ході аналізу існуючих вебзастосунків було досліджено сервіси Tracklib, Splice, Looperman, Cymatics, Plugin Boutique та PresetShare. Встановлено, що кожен із розглянутих сервісів спеціалізується лише на окремих типах цифрових ресурсів або окремих аспектах роботи з ними. Частина платформ орієнтується на продаж контенту, інші — на підпискову модель доступу або безкоштовний обмін файлами між користувачами. При цьому жоден із проаналізованих сервісів не забезпечує єдиного комплексного середовища для роботи одночасно із семплами, наборами семплів та пресетами.

Також було визначено основні переваги та недоліки існуючих рішень. До переваг можна віднести великі бібліотеки контенту, інтеграцію з DAW та зручний пошук матеріалів. Серед недоліків виділено залежність від підписок, нерівномірну якість користувацького контенту, обмежені можливості безкоштовного використання, а також відсутність універсального функціоналу для роботи з різними видами музичних ресурсів в межах одного застосунку.

На основі проведеного аналізу було сформульовано постановку задачі та визначено основні вимоги до розроблюваного вебзастосунку. Запропонований застосунок повинен забезпечувати можливість пошуку, завантаження, збереження, додавання, редагування та видалення цифрових музичних ресурсів різних типів. Окрім цього, було визначено функціональні вимоги, межі розробки, структуру користувацького інтерфейсу та вимоги до продуктивності системи.

2 МАТЕРІАЛИ ТА МЕТОДИ

У цьому розділі буде проведено вибір та обґрунтування мов програмування, середовища розробки, фреймворків, СУБД та хмарних сховищ з приведенням порівняльних таблиць.

2.1 Вибір та обґрунтування мови програмування

2.1.1 Вибір мови для клієнтської частини веб-застосунку

Для розробки клієнтської частини вебзастосунку необхідно обрати мову програмування, яка забезпечить створення сучасного, інтерактивного та зручного користувацького інтерфейсу. Оскільки застосунок міститиме велику кількість складних елементів інтерфейсу таких як пошук і сортування ресурсів, таблиці з аудіо матеріалами, drag-and-drop функціонал, форми завантаження файлів, динамічні сторінки та взаємодію з сервером у реальному часі, мова програмування повинна забезпечувати зручність підтримки великого обсягу коду та мінімізувати кількість можливих помилок.

Фактичним стандартом для розробки клієнтської частини вебзастосунків є JavaScript. Саме ця мова підтримується всіма сучасними браузерами без необхідності використання додаткових плагінів або компіляторів. Завдяки цьому JavaScript став основою сучасної веб розробки та використовується у більшості сучасних фронтенд-фреймворків і бібліотек.

Проте JavaScript має низку недоліків, які стають особливо помітними під час розробки великих або складних вебзастосунків. Основним із них є динамічна слабка типізація. Відсутність строгого контролю типів може призводити до появи помилок, які виявляються лише під час виконання програми. У великих проєктах це ускладнює підтримку коду, рефакторинг та взаємодію між різними модулями застосунку.

Для розробки вебзастосунку, який міститиме значну кількість взаємопов'язаних компонентів, обробку файлів, роботу з API та складні

структури даних, важливо забезпечити передбачуваність поведінки програми та безпечність роботи з даними. Саме тому доцільним є використання TypeScript який є розширенням JavaScript із підтримкою статичної типізації.

TypeScript [12] дозволяє явно визначати типи змінних, параметрів функцій та структур даних, що значно знижує ймовірність виникнення логічних помилок під час розробки. Завдяки перевірці типів ще на етапі компіляції значна частина потенційних помилок може бути виявлена до запуску застосунку. Це особливо важливо для вебзастосунку, який працюватиме з різними типами музичних ресурсів, великою кількістю форм та складною логікою взаємодії між компонентами інтерфейсу.

Окрім підвищення безпечності коду, TypeScript значно покращує зручність підтримки та масштабування проєкту. Наявність типів спрощує навігацію кодом, автоматичне доповнення у середовищах розробки, рефакторинг та командну роботу. Навіть попри те, що розроблюваний вебзастосунок не належить до надзвичайно великих або високонавантажених систем, використання TypeScript дозволить підтримувати структуру коду впорядкованою та зрозумілою під час подальшого розвитку проєкту.

Ще однією важливою перевагою TypeScript є його широка підтримка сучасною екосистемою веб розробки. Більшість сучасних бібліотек та фреймворків, таких як React, Next.js або Vue, мають повну підтримку TypeScript та рекомендують його використання як основний підхід до розробки. Це забезпечує доступ до великої кількості готових рішень, документації, прикладів використання та підтримки спільноти.

Також важливо враховувати специфіку предметної області розроблюваного застосунку. Вебзастосунок для розповсюдження музичних ресурсів міститиме значну кількість інтерактивних елементів: аудіо плеєри, візуалізацію хвильових форм, динамічні таблиці, системи пошуку та фільтрації, drag-and-drop взаємодію з файлами та роботу із хмарним сховищем. Реалізація таких можливостей потребує використання сучасних браузерних API та

бібліотек, для яких TypeScript забезпечує значно кращу інтеграцію та безпечність використання.

У таблиці 2.1 наведено порівняння JavaScript та TypeScript.

Таблиця 2.1 – Порівняння JavaScript та TypeScript

Характеристика	Javascript	Typescript
Типізація	Динамічна, слабка	Статична, сильна
Виявлення помилок	Переважно під час виконання	При компіляції
Безпечність коду	Нижча	Вища
Зручність підтримки великих проєктів	Обмежена	Висока
Підтримка IDE	Часткова	Повна

Отже, для розробки клієнтської частини вебзастосунку доцільно обрати TypeScript. Дана мова поєднує універсальність JavaScript із перевагами статичної типізації, що дозволяє створювати більш безпечний, зрозумілий та підтримуваний код. Окрім цього, TypeScript має широку підтримку сучасних фреймворків, активну спільноту та добре підходить для реалізації інтерактивного інтерфейсу вебзастосунку для роботи з цифровими музичними ресурсами.

2.1.2 Вибір мови для серверної частини веб-застосунку

Під час вибору мови програмування для серверної частини вебзастосунку необхідно враховувати специфіку розроблюваної системи. Серверна частина застосунку відповідатиме за авторизацію користувачів, обробку HTTP-запитів, взаємодію з базою даних, збереження та завантаження файлів, пошук ресурсів, обробку метаданих аудіо файлів та взаємодію з хмарним сховищем.

На відміну від високонавантажених систем реального часу або складних обчислювальних платформ, розроблюваний застосунок не потребує

надзвичайно складних технічних рішень або максимальної обчислювальної продуктивності. Основними вимогами до серверної частини є швидкість розробки, зручність підтримки, надійність, безпечність роботи з даними та простота інтеграції з клієнтською частиною.

Серед найбільш популярних мов програмування для серверної розробки можна виділити Java, C#, Go та TypeScript.

Java є однією з найпоширеніших мов для створення великих корпоративних систем. Вона забезпечує високу продуктивність, стабільність та підтримку багатопоточності. Однак Java має доволі складний синтаксис та вимагає значної кількості шаблонного коду, що може сповільнювати розробку невеликих та середніх вебзастосунків. Для розроблюваного проєкту, де основний акцент робиться на швидкості створення функціоналу та зручності підтримки, використання Java є менш доцільним.

C# у поєднанні з платформою .NET також є потужним рішенням для серверної розробки. Ця мова забезпечує хорошу продуктивність, строгий контроль типів та сучасні інструменти розробки. Проте екосистема C# більш орієнтована на корпоративні застосунки та потребує глибшої інтеграції з платформою .NET. Для відносно компактного вебзастосунку така інфраструктура може бути надлишковою.

Go є сучасною мовою програмування, орієнтованою на створення швидких та ефективних серверних застосунків. Її перевагами є висока продуктивність, проста модель паралелізму та невелике споживання ресурсів. Проте Go має обмежені можливості абстракції та менш розвинену екосистему для створення комплексних веб інтерфейсів і систем із великою кількістю бізнес-логіки. Також використання Go вимагало б підтримки окремої серверної мови, відмінної від клієнтської частини застосунку.

TypeScript у серверній розробці найчастіше використовується разом із платформою Node.js [13]. Такий підхід дозволяє використовувати одну мову програмування як для клієнтської, так і для серверної частин вебзастосунку. Це

суттєво спрощує структуру проєкту, зменшує кількість технологій, які необхідно підтримувати, та прискорює розробку.

Використання однієї мови на всіх рівнях застосунку також спрощує повторне використання типів даних, моделей та частини логіки між клієнтом і сервером. Наприклад, структури даних для опису семплів, пресетів або профілів користувачів можуть використовуватись одночасно в обох частинах системи без дублювання коду. Це знижує ймовірність помилок та підвищує узгодженість між компонентами застосунку.

Окрім цього, екосистема Node.js та TypeScript містить велику кількість сучасних фреймворків і бібліотек для роботи з вебзастосунками, файлами, базами даних, авторизацією та хмарними сервісами. Такі технології добре підходять для реалізації функціоналу, необхідного у розроблюваному застосунку: завантаження файлів, роботи з метаданими аудіо, пошуку ресурсів, реалізації REST API та взаємодії із клієнтською частиною.

У таблиці 2.2 наведено порівняння мов програмування для серверної частини вебзастосунку.

Таблиця 2.2 – Порівняння мов програмування для серверної частини

Характеристика	Typescript	Java	C#	Go
Швидкість розробки	Висока	Середня	Середня	Висока
Простота синтаксису	Висока	Нижча	Середня	Висока
Можливість використання на клієнтській стороні	Так	Ні	Ні	Ні
Підтримка сучасної веброзробки	Висока	Висока	Висока	Середня
Екосистема npm	Повна	Відсутня	Відсутня	Обмежена
Зручність підтримки невеликих проєктів	Висока	Середня	Середня	Висока

Продовження таблиці 2.2

Характеристика	Typescript	Java	C#	Go
Продуктивність	Середня	Висока	Висока	Висока
Популярність у веброзробці	Дуже висока	Висока	Висока	Висока

Отже, для розробки серверної частини вебзастосунку доцільно обрати TypeScript у поєднанні з платформою Node.js. Такий вибір забезпечує використання єдиної мови програмування для всієї системи, що значно спрощує розробку та підтримку проєкту. Статична типізація TypeScript підвищує безпечність коду та дозволяє виявляти помилки ще на етапі компіляції. Окрім цього, сучасна екосистема Node.js і TypeScript забезпечує доступ до великої кількості бібліотек та інструментів, необхідних для реалізації функціоналу вебзастосунку для розповсюдження цифрових музичних ресурсів.

2.2 Вибір та обґрунтування середовища розробки

Для розробки сучасного вебзастосунку важливим етапом є вибір інтегрованого середовища розробки (IDE), яке забезпечує ефективну роботу як із клієнтською, так і серверною частинами проєкту. Оскільки в даному проєкті мова програмування TypeScript використовується як для frontend-частини, так і для backend-частини застосунку, середовище розробки повинно підтримувати сучасні JavaScript/TypeScript-технології, інструменти автоматизації, системи контролю версій та засоби налагодження.

Після аналізу сучасних середовищ розробки було обрано для порівняння такі рішення: Visual Studio Code [14], Sublime Text, Neovim та Visual Studio.

Visual Studio Code це безкоштовне багатоплатформне середовище розробки від Microsoft, яке поєднує легкість текстового редактора та функціональність повноцінної IDE. VS Code має вбудовану підтримку TypeScript, що є важливою перевагою для цього проєкту, оскільки TypeScript використовується як на клієнтській, так і на серверній стороні. Середовище

підтримує сучасні frontend-фреймворки та backend-інструменти для Node.js, а також має велику екосистему розширень, зокрема ESLint, Prettier, Docker та інші. Важливою перевагою є інтегрований термінал, підтримка налагодження застосунків та вбудована інтеграція з Git. Завдяки високій швидкодії та помірному споживанню ресурсів Visual Studio Code є зручним рішенням для повноцінної веброзробки.

Sublime Text це легкий та швидкий текстовий редактор, який підтримує JavaScript та TypeScript через систему плагінів. Його основними перевагами є висока продуктивність та мінімалістичний інтерфейс. Однак можливості Sublime Text значно обмежені у порівнянні з сучасними IDE. Для роботи з TypeScript, Node.js, автоматичним форматуванням, тестуванням та налагодженням необхідно встановлювати велику кількість сторонніх плагінів. Крім цього, середовище не має настільки глибокої інтеграції із сучасними інструментами веброзробки, як Visual Studio Code.

Neovim це сучасна модифікація текстового редактора Vim, орієнтована на досвідчених користувачів та ентузіастів. Завдяки великій кількості плагінів Neovim може бути налаштований для повноцінної роботи з TypeScript та Node.js. Редактор забезпечує високу продуктивність та мінімальне споживання ресурсів. Проте для ефективного використання Neovim необхідне глибоке налаштування середовища, встановлення та конфігурація численних плагінів, а також знання командного інтерфейсу. Через складність освоєння та підтримки конфігурації Neovim є менш практичним вибором для виконання дипломного проєкту.

Visual Studio це повнофункціональне інтегроване середовище розробки від Microsoft. Воно має широкий набір інструментів для створення програмного забезпечення, підтримку налагодження, тестування та інтеграції із системами контролю версій. Visual Studio найбільш орієнтоване на розробку застосунків мовою C# та платформою .NET. Хоча підтримка JavaScript і TypeScript також присутня, середовище є більш вимогливим до системних ресурсів та менш зручним для веброзробки на Node.js у порівнянні з Visual Studio Code. Крім

цього, значна частина функціоналу Visual Studio орієнтована на великі корпоративні проекти, що є надлишковим для даної роботи.

Порівняння основних середовищ розробки наведено у таблиці 2.3.

Таблиця 2.3 – порівняння середовищ розробки

Характеристика	Visual Studio Code	Sublime Text	Neovim	Visual Studio
Підтримка typescript	Висока	Через плагіни	Через плагіни	Висока
Підтримка Nodejs	Висока	Обмежена	Висока	Висока
Наявність засобів автодоповнення та рефакторингу	Висока	Базові	Через плагіни	Висока
Інтеграція з git	Вбудована	Через плагіни	Через плагіни	Вбудована
Підтримка дебагу	Вбудована	Обмежена	Через плагіни	Вбудована
Підтримка розширень	Дуже велика	Обмежена	Велика	Велика
Продуктивність	Висока	Дуже висока	Дуже висока	середня
Зручність налаштування	Висока	Середня	Низька	Висока

Проведений аналіз показав, що найбільш доцільним середовищем розробки для даного проекту є Visual Studio Code. Це середовище забезпечує повноцінну підтримку TypeScript як для клієнтської, так і для серверної частини застосунку, має вбудовані засоби налагодження та інтеграцію з Git, підтримує сучасні інструменти веброботи та дозволяє гнучко розширювати функціональність за допомогою плагінів. Важливою перевагою також є безкоштовність, мультиплатформеність та оптимальне співвідношення між функціональністю і продуктивністю, що робить Visual Studio Code найкращим вибором для реалізації даного веб проекту

2.3 Вибір та обґрунтування фреймворків

2.3.1 Вибір фреймворку для клієнтської частини вебзастосунку

При виборі фреймворку для клієнтської частини сучасного вебзастосунку важливо враховувати не лише популярність технології, а й її архітектурні можливості, продуктивність, підтримку серверного рендерингу (SSR), масштабованість та зручність розробки. У даному проєкті клієнтська частина передбачає динамічний інтерфейс, роботу з даними у реальному часі, авторизацію користувачів та інтеграцію з API, тому доцільно використовувати сучасні рішення на основі бібліотеки react.

Серед основних фреймворків і бібліотек для фронтенд-розробки можна виділити React (у поєднанні з Next.js), Angular, Vue.js та Svelte. Кожен із них має власну архітектурну філософію та підхід до побудови інтерфейсу.

React [15] у поєднанні з Next.js [16] є одним із найпотужніших рішень для сучасної веброзробки. react забезпечує компонентний підхід до побудови інтерфейсу, що значно полегшує масштабування та повторне використання коду. У свою чергу Next.js розширює можливості react, додаючи серверний рендеринг (SSR), статичну генерацію сторінок (SSG), гібридні режими рендерингу та вбудовану маршрутизацію.

Завдяки цьому Next.js є особливо ефективним для SEO-орієнтованих застосунків, таких як системи бронювання, де важлива швидкість завантаження сторінок та індексація контенту. Додатково Next.js підтримує API routes, що дозволяє частково реалізувати backend-логіку без окремого сервера для простих сценаріїв.

Angular є повноцінним фреймворком, який надає все необхідне «з коробки»: DI (dependency injection), маршрутизацію, HTTP-клієнт, форми та вбудовану структуру проєкту. Його головною перевагою є строгість архітектури та орієнтація на великі корпоративні системи. Однак складна структура, висока навчальна крива та надмірна «важкість» роблять Angular менш гнучким у порівнянні з React + Next.js для швидкої розробки сучасних UI.

Vue.js є легшим та більш інтуїтивним фреймворком, який забезпечує високу швидкість розробки та простоту входу. У поєднанні з Nuxt.js (аналог Next.js для Vue) він також підтримує SSR та SSG. Проте екосистема Vue є менш поширеною у великих enterprise промислових проєктах, а кількість готових рішень та бібліотек поступається екосистемі react.

SvelteKit пропонує компіляцію компонентів у чистий JavaScript без віртуального DOM. Це забезпечує високу продуктивність, однак екосистема та кількість інструментів все ще поступаються Next.js, що може ускладнити довгострокову підтримку проєкту.

Порівняльну характеристику клієнтських фреймворків наведено у таблиці 2.4.

Таблиця 2.4 – Порівняння клієнтських фреймворків

Характеристика	Next.js	Angular	Vue.js/nuxt	Sveltekit
Архітектура	Компонентна	Монолітна	Компонентна	Компіляційна
SSR/SSG	Повна підтримка	Обмежена	Повна підтримка	Повна підтримка
Продуктивність	Висока	Середня	Висока	Дуже висока
SEO	Відмінна	Середня	Відмінна	Відмінна
Навчальна крива	Помірна	Висока	Низька	Низька
Екосистема	Дуже велика	Висока	Середня	Низька
Гнучкість	Висока	Низька	Висока	Висока

У результаті аналізу можна зробити висновок, що React у поєднанні з Next.js є оптимальним вибором для клієнтської частини вебзастосунку. Це рішення забезпечує баланс між продуктивністю, масштабованістю та гнучкістю архітектури. Використання Next.js дозволяє значно покращити швидкість завантаження сторінок та користувацький досвід завдяки SSR та SSG, що є критично важливим для вебзастосунку бронювання.

2.3.2 Вибір фреймворку для серверної частини вебзастосунку

Серверна частина вебзастосунку повинна забезпечувати стабільну роботу API, високу продуктивність, масштабованість, підтримку асинхронних операцій, інтеграцію з базами даних та зручність тестування. Оскільки в проєкті використовується TypeScript, доцільно розглядати Node.js-фреймворки.

Серед основних варіантів для серверної розробки можна виділити Express.js, Fastify, NestJS та Koa.js.

Express.js є найпопулярнішим мінімалістичним фреймворком для Node.js. Він забезпечує базову функціональність для створення REST API, але не має строгих архітектурних правил. Це дає гнучкість, але ускладнює масштабування великих проєктів, оскільки розробник самотійно відповідає за структуру застосунку.

Fastify є більш сучасною альтернативою Express, орієнтованою на високу продуктивність. Він забезпечує швидку обробку запитів, підтримку плагінної архітектури та низькі затримки. Однак Fastify залишається низькорівневим фреймворком і не надає повної архітектурної структури для великих проєктів.

Koa.js це мінімалістичний фреймворк від команди Express, який використовує сучасні можливості JavaScript (async/await). Він є більш гнучким, але також вимагає самотійного проєктування архітектури та підключення додаткових бібліотек.

NestJS є сучасним прогресивним фреймворком для Node.js, побудованим на TypeScript і натхненним архітектурою Angular. Він надає чітку модульну структуру, підтримує dependency injection, декоратори, middleware, guards, pipes та interceptors.

Основною перевагою NestJS є його архітектурна організованість, що дозволяє будувати масштабовані та легко підтримувані серверні застосунки. Він також підтримує інтеграцію з Express або Fastify як базовими HTTP-адаптерами, що дозволяє комбінувати продуктивність і гнучкість.

Додатково NestJS має вбудовану підтримку:

- роботи з базами даних (TypeORM, Prisma, Mongoose);
- валідації запитів;
- тестування;
- мікросервісної архітектури.

Порівняльну характеристику серверних фреймворків наведено у таблиці 2.5.

Таблиця 2.5 – Порівняння серверних фреймворків

Характеристика	Express	Fastify	Кoa	Nest
Архітектура	Мінімалістична	Заснована на плагінах	Мінімалістична	Модульна
Продуктивність	Висока	Дуже висока	Висока	Висока
Масштабованість	Середня	Середня	Середня	Дуже висока
Підтримка typescript	Часткова	Часткова	Часткова	Повна
Структурованість коду	Низька	Середня	Низька	Висока
Наявність dependency injection	Немає	Немає	Немає	Вбудована
Підтримка великих проєктів	Обмежена	Середня	Обмежена	Висока

2.4 Вибір та обґрунтування СУБД

Під час вибору системи управління базами даних для розроблюваного вебзастосунку необхідно враховувати специфіку предметної області. У межах цього проєкту система повинна зберігати відомості про користувачів, цифрові ресурси різних типів, метадані аудіо файлів, теги, статистику завантажень, а також дані про вкладені структури наборів семплів. Через це база даних має бути достатньо гнучкою, щоб підтримувати різні структури документів без надмірної складності схеми. MongoDB Atlas є документною NoSQL-базою

даних із гнучкою схемою, яка дозволяє поступово змінювати модель даних і наближати її до об'єктів у коді. MongoDB [17] також підтримує schema validation для окремих полів, якщо для них потрібно жорсткіше обмеження.

Для серверної частини застосунку разом із MongoDB доцільно використовувати Mongoose. Це ODM для Node.js, який надає schema-based підхід до моделювання даних, вбудовані механізми type casting, validation, query building і hooks. Окремою перевагою є офіційна підтримка TypeScript, завдяки чому схеми та моделі даних зручніше використовувати в проєкті, побудованому на TypeScript як на клієнті, так і на сервері. Як альтернативу можна розглядати Firebase, Supabase та MySQL.

Cloud Firestore у складі Firebase є NoSQL-документною базою, де дані зберігаються у документах і колекціях; вона орієнтована на веб- і мобільні застосунки, а також має вбудовані механізми realtime-оновлення та offline-підтримки. Realtime Database Firebase також є NoSQL-хмарною базою з акцентом на синхронізацію даних у реальному часі.

Supabase, у свою чергу, надає повноцінну Postgres-базу для кожного проєкту, а також Realtime, резервні копії, розширення, Auth і Storage. Це зручне комплексне рішення, однак у межах даного проєкту воно є більш “платформним” і менш вузькоспеціалізованим для гнучкого зберігання різномірних документів музичних ресурсів.

MySQL є реляційною СУБД, у якій дані зберігаються у таблицях, рядках і стовпцях, що добре підходить для строго структурованих даних, але вимагає більшої дисципліни під час моделювання змінних за складом об'єктів.

Порівняння основних СУБД наведено у таблиці 2.6.

Отже, для розроблюваного вебзастосунку доцільно обрати MongoDB Atlas як основну СУБД, та Mongoose як інструмент моделювання даних на сервері. Такий вибір забезпечує гнучке зберігання різних типів музичних ресурсів, зручну роботу з неоднорідними метаданими та просту інтеграцію з TypeScript-стеком. Для проєкту, у якому структура ресурсів може відрізнятися залежно від типу об'єкта, документна модель є більш практичною за реляційну.

Таблиця 2.6 – Порівняння СУБД

Характеристика	MongoDB atlas	Firebase firestore/realtime database	Supabase (PostgreSQL)	MySQL
Модель даних	Документи, гнучка схема	Документи/ Колекції	Реляційна Postgres-модель	Реляційні Таблиці
Гнучкість Структури	Висока	Висока	Середня	Низька
Підтримка Валідації	Schema validation Та mongoose Validation	Є правила Доступу, але Модель Жорсткіша	Rls та політика SQL	Constraints та Реляційні Обмеження
Зручність для typescript/node.js	Висока через mongoose	Інтеграція sdk	Орієнтована на postgres- екосистему	Orm/драйверна
Відповідність проєкту	Висока	Висока	Середня	Середня

2.5 Вибір та обґрунтування хмарного сховища для зберігання файлів

Для вебзастосунку, який оперує аудіофайлами, наборами семплів та пресетами, окремо від бази даних необхідне надійне хмарне сховище для файлів. Файли мають бути доступні для завантаження, перегляду та повторного використання, а система повинна підтримувати великі обсяги даних і стабільну доставку контенту. Cloudflare R2 [18] є S3-сумісним об'єктним сховищем, побудованим на глобальній мережі Cloudflare; у документації вказано, що сервіс не має egress fees, є strongly consistent і розрахований на високу надійність та гнучку мультихмарну архітектуру.

S3-сумісність є важливою перевагою, оскільки R2 можна використовувати через широкий спектр S3 SDK, бібліотек і інструментів, змінюючи переважно лише endpoint. Це спрощує інтеграцію з Node.js-застосунком і зменшує ризик технологічної прив'язки до одного провайдера. Для проєкту, що працює з великими аудіофайлами та їх регулярним завантаженням, відсутність egress fees є суттєвою перевагою порівняно з класичними схемами платного вихідного трафіку.

Як альтернативи можна розглядати Cloud Storage for Firebase та Supabase Storage. Firebase Storage призначене для зберігання й розповсюдження користувацького контенту, такого як зображення й відео, і використовує bucket у Google Cloud Storage; воно також підтримує завантаження з веббраузера та мобільних клієнтів. Supabase Storage надає сховище файлів, тісно інтегроване з Postgres і Row Level Security. Обидва варіанти зручні, але вони краще підходять як частина власної екосистеми відповідних платформ, тоді як Cloudflare R2 є більш універсальним об'єктним сховищем із сильним акцентом на сумісність та економічність передачі даних.

Порівняння основних хмарних сховищ наведено у таблиці 2.7.

Таблиця 2.7 – Порівняння хмарних сховищ

Характеристика	Cloudflare R2	Firebase storage	Supabase storage
Тип сховища	Об'єктне S3-сумісне сховище	Хмарне сховище в google cloud storage bucket	Сховище файлів, інтегроване з postgres
Сумісність з S3	Так	Ні	Ні
Вартість передачі даних	Безкоштовна	Залежить від правил google cloud	Залежить від тарифної моделі supabase
Масштабованість	Висока, глобальна мережа cloudflare	Висока, google cloud інфраструктура	Висока, у межах платформи supabase
Зручність інтеграції з node.js	Висока через S3 api	Висока через sdk firebase	Висока через інтеграцію з supabase-екосистемою
Відповідність проєкту	Висока	Середня	Середня

Отже, для розроблюваного вебзастосунку найбільш доцільним є використання Cloudflare R2. Цей сервіс забезпечує S3-сумісність, стабільне зберігання файлів і зручну інтеграцію з сервером на Node.js, а відсутність egress fees є особливо корисною для платформи, що передбачає часте завантаження аудіо ресурсів користувачами.

2.6 Вибір та обґрунтування платформ для хостингу

2.6.1 Вибір платформи для хостингу клієнтської частини вебзастосунку

Під час вибору платформи для розгортання клієнтської частини вебзастосунку важливо враховувати такі фактори, як швидкість доставки контенту користувачу, підтримка сучасних frontend-фреймворків (зокрема Next.js), простота CI/CD інтеграції, масштабованість та вартість використання. Оскільки клієнтська частина проекту реалізується із застосуванням React та Next.js, платформа для хостингу повинна мати нативну або глибоку підтримку серверного рендерингу (SSR) та статичної генерації сторінок (SSG).

Серед основних рішень для хостингу фронтенд-застосунків можна виділити Netlify, Vercel, Firebase Hosting та GitHub Pages.

Netlify є однією з найпопулярніших платформ для розгортання сучасних вебзастосунків. Вона забезпечує автоматичне розгортання з Git-репозиторію, підтримує CI/CD процеси, CDN-розповсюдження контенту та просту інтеграцію з API. Netlify добре підходить для статичних сайтів і SPA-застосунків, однак його підтримка складних SSR-сценаріїв Next.js є менш гнучкою у порівнянні з конкурентами, особливо при використанні серверних функцій або динамічного рендерингу.

Vercel є платформою, яка була спеціально створена командою Next.js і тому має найкращу інтеграцію з цим фреймворком. Вона забезпечує автоматичну оптимізацію SSR, ISR (Incremental Static Regeneration), edge-функції та глобальний CDN. Vercel дозволяє максимально ефективно використовувати можливості Next.js без додаткових конфігурацій, що робить її фактично еталонним рішенням для подібних проєктів.

Firebase Hosting також може використовуватися для розгортання фронтенд-застосунків. Його основною перевагою є глибока інтеграція з іншими сервісами Firebase, такими як Authentication та Firestore. Однак у випадку

складного SSR-застосунку або кастомної серверної логіки він є менш гнучким, оскільки орієнтований переважно на статичні або напівстатичні вебсайти.

GitHub Pages є простим рішенням для хостингу статичних сайтів. Воно підходить для базових проєктів, але не підтримує серверний рендеринг, складну маршрутизацію або серверні функції, що робить його непридатним для сучасних Next.js застосунків із динамічним контентом.

У результаті аналізу можна зробити висновок, що найбільш доцільним вибором для хостингу клієнтської частини вебзастосунку є Vercel. Це зумовлено його глибокою інтеграцією з Next.js, автоматичною підтримкою SSR та SSG, високою продуктивністю доставки контенту через глобальний CDN та простотою розгортання. Використання Vercel дозволяє мінімізувати складність DevOps-налаштувань та зосередитися на розробці функціональності застосунку.

2.6.2 Вибір платформи для хостингу серверної частини вебзастосунку

Серверна частина вебзастосунку відповідає за обробку API-запитів, роботу з базою даних, авторизацію користувачів, бізнес-логіку та інтеграцію з зовнішніми сервісами. Відповідно, платформа для її розгортання повинна забезпечувати стабільну роботу Node.js-застосунків (зокрема NestJS), підтримку змінних середовища, масштабованість, можливість автоматичного деплою та достатній рівень продуктивності.

Серед популярних платформ для хостингу backend-застосунків можна виділити Render, Heroku, Railway та AWS (EC2 / Elastic Beanstalk).

Render є сучасною платформою для розгортання вебзастосунків, яка підтримує Node.js, автоматичний деплой із Git, Docker-контейнери, фонові сервіси та бази даних. Її головною перевагою є простота використання та відсутність складної конфігурації, що дозволяє швидко розгорнути NestJS API без необхідності ручного налаштування серверної інфраструктури. Render також забезпечує автоматичне масштабування та моніторинг стану сервісів.

Heroku є однією з найстаріших платформ для хостингу застосунків. Вона має просту модель деплою та підтримує різні мови програмування. Однак у сучасних умовах її безкоштовні можливості обмежені, а продуктивність і гнучкість поступаються більш сучасним рішенням, таким як Render або Railway.

Railway є більш сучасною альтернативою Heroku, яка забезпечує швидкий деплой, інтеграцію з GitHub, автоматичне масштабування та зручний інтерфейс керування сервісами. Railway добре підходить для невеликих і середніх проєктів, однак його безкоштовний тариф має обмеження щодо ресурсів, що може бути критичним при зростанні навантаження.

Amazon Web Services надає найширші можливості для розгортання серверної частини через сервіси EC2 або Elastic Beanstalk. AWS забезпечує максимальну гнучкість, масштабованість та контроль над інфраструктурою, однак вимагає значно складнішого налаштування, глибоких знань DevOps та часу на конфігурацію. Для навчального або середнього за складністю проєкту це рішення є надлишковим.

У результаті аналізу можна зробити висновок, що оптимальним вибором для хостингу серверної частини вебзастосунку є Render. Це пояснюється його простотою використання, підтримкою Node.js та NestJS, автоматичним деплоєм з Git, вбудованими засобами моніторингу та достатнім рівнем масштабованості. Render дозволяє швидко розгорнути backend-сервіс без складної інфраструктурної підготовки, що є важливою перевагою для даного проєкту.

2.7 Висновки за розділом 2

У межах розділу 2 було здійснено комплексний аналіз та обґрунтування вибору технологічного стеку для розробки вебзастосунку для роботи з цифровими музичними ресурсами. Розгляд охоплював мови програмування, середовища розробки, фреймворки, системи управління базами даних, хмарні сховища та платформи хостингу клієнтської і серверної частин системи.

За результатами аналізу встановлено, що використання TypeScript є найбільш доцільним рішенням як для клієнтської, так і для серверної частин застосунку. Це забезпечує єдиний технологічний стек, підвищує безпечність коду завдяки статичній типізації та спрощує підтримку і масштабування проекту.

Для середовища розробки обрано Visual Studio Code як найбільш збалансоване рішення, що поєднує високу продуктивність, широку підтримку TypeScript та Node.js, а також розвинену екосистему розширень і зручні інструменти для командної роботи.

У якості клієнтського фреймворку обрано React у поєднанні з Next.js, що забезпечує компонентну архітектуру, високу продуктивність, підтримку SSR та SSG, а також покращену SEO-оптимізацію та користувацький досвід. Для серверної частини обрано NestJS, який надає структуровану модульну архітектуру, підтримку dependency injection та добре масштабується у межах складних вебзастосунків.

Серед систем управління базами даних найбільш доцільною визначено MongoDB Atlas у поєднанні з Mongoose. Такий підхід забезпечує гнучку документну модель даних, що є оптимальною для зберігання різномірних музичних ресурсів та метаданих, а також зручну інтеграцію з Node.js і TypeScript.

Для зберігання файлів обрано Cloudflare R2, що забезпечує S3-сумісність, високу масштабованість та економічну ефективність завдяки відсутності плати за вихідний трафік. Це є особливо важливим для застосунку, орієнтованого на роботу з великими обсягами аудіофайлів.

Щодо хостингу клієнтської частини визначено платформу Vercel, яка забезпечує оптимальну інтеграцію з Next.js, підтримку SSR/SSG, високу швидкість доставки контенту та простоту розгортання. Для серверної частини обрано Render як просту та ефективну платформу для деплою Node.js/NestJS застосунків із підтримкою автоматичного розгортання та масштабування.

3 ПРОЄКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ

3.1 Проєктування архітектури системи

Розробка архітектури програми є важливим етапом у процесі створення вебзастосунку. Архітектура програми визначає структуру, компоненти та взаємодію між ними, що визначає робочі процеси, ефективність та масштабованість застосунку.

У цьому розділі буде розглянуто процес розробки архітектури програми, визначено ключові компоненти та їх взаємозв'язки, а також обговорено стратегії та підходи до створення ефективної та масштабованої архітектури для розроблюваного вебзастосунку. UML (Unified Modeling Language) діаграми дуже популярні та зручні діаграми для проєктування, саме вони будуть в подальшому використані при проєктуванні вебзастосунку. Для створення більшості з наведених далі діаграм було використано інструмент PlantUML [19].

Далі на рисунках 3.1 — 3.9 наведено створення в ході проєктування системи діаграми:

- класів застосунка;
- компонентів застосунка;
- діаграму яка наводить спрощене представлення усіх прецедентів системи;
- діаграму прецедентів для сторінки /created/samples, тобто відображення та редагування семплів, створених користувачем;
- діаграму прецедентів сторінки /explore/samples, тобто відображення, пошук, фільтрація, сортування та зберігання усіх доступних на сайті семплів;
- діаграму прецедентів для сторінки /upload/pack, тобто додавання нового набору до сайту, його розділ на папки та підпапки та завантаження до нього файлів;
- діаграму діяльності системи.

Користувачі, можуть зберігати/видаляти збережені ресурси, якими можуть бути семпли, пресети та набори, причому набори складаються з багатьох файлів та папок, семпли мають один аудіо файл, та пресети мають два файли (сам пресет та демонстраційний аудіофайлів). Усі папки та файли зберігаються у хмарному сховищі Cloudflare R2 Bucket.

Користувачі можуть реєструватись та заходити в систему, в якій вони виконують CRUD операції з матеріалами різних типів, а саме: додавання нових матеріалів, редагування та видалення доданих матеріалів, перегляд/пошук/фільтрування усіх матеріалів, зберігання, перегляд та завантаження збережених.

Далі наведено діаграму компонентів системи (рис. 3.2) та абстрактну діаграму прецедентів системи (рис. 3.3).

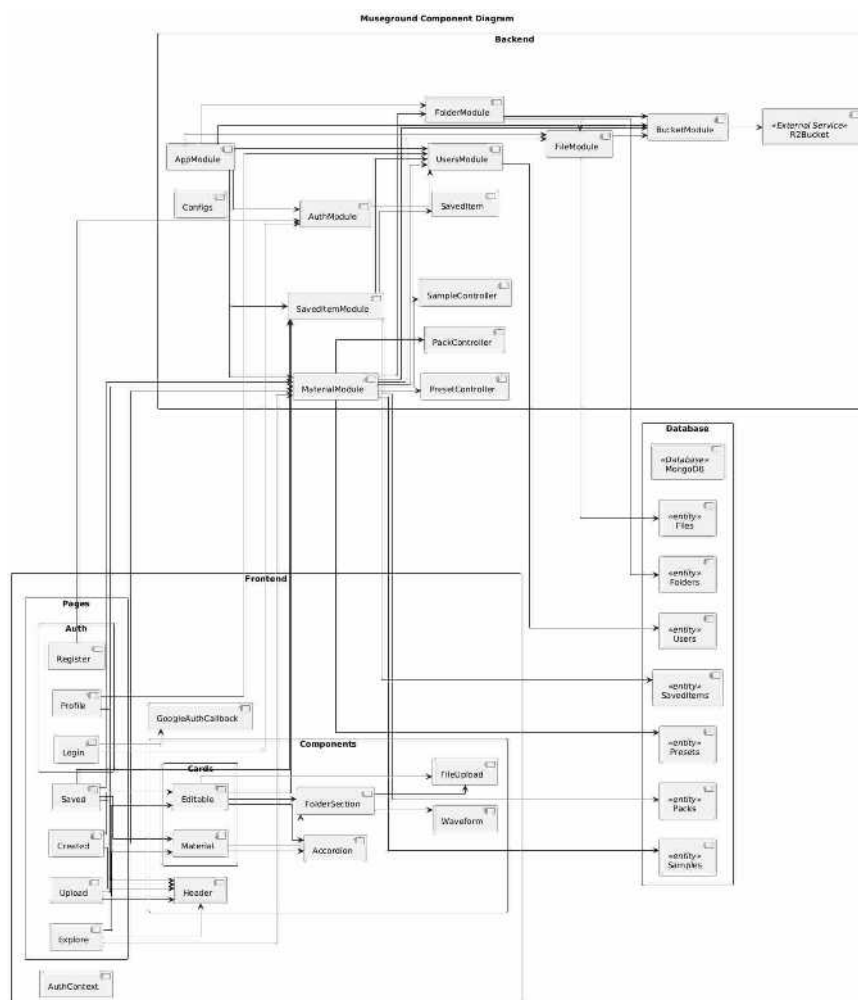


Рисунок 3.2 — Діаграма компонентів системи

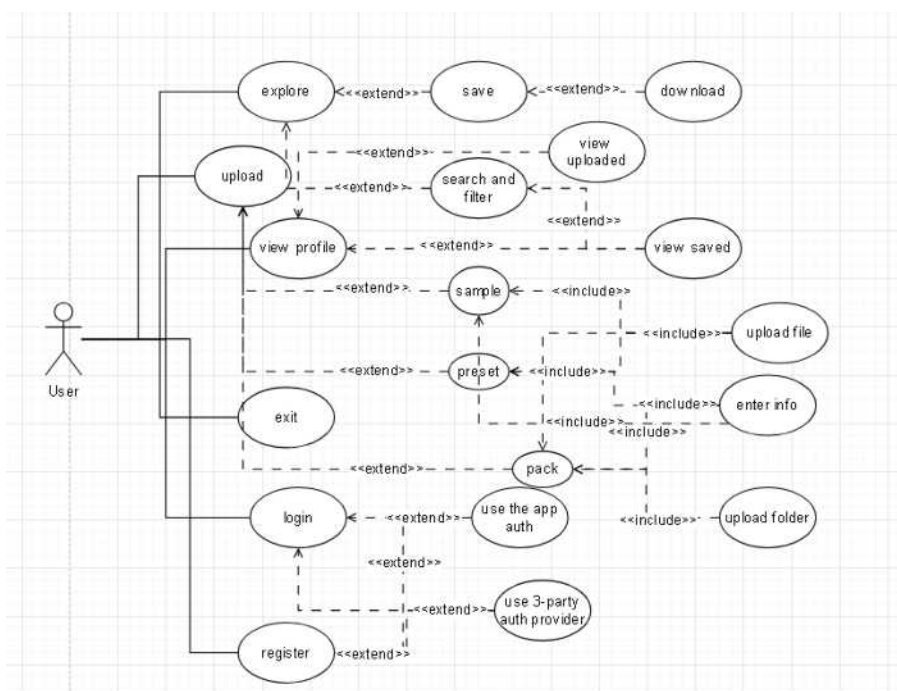


Рисунок 3.3 — Абстрактна діаграма прецедентів системи

Далі наведено деталізовані моделі прецедентів «upload pack» (додавання нового набору), «explore samples» (перегляд, пошук, фільтрування, збереження семплів) та «created samples» (перегляд, редагування та видалення семплів, створених користувачем), а також текстовий опис кожного з цих прецедентів.

Опис потоків подій upload pack наведено далі за текстом. Діаграму прецедентів upload pack наведено на рисунку 3.4.

Основний потік (створення пака):

- User відкриває сторінку створення пака (Open Create Pack Page);
- вводить назву нового пака (Enter pack title);
- натискає кнопку створення (Click Create Pack button);
- Client перевіряє, чи назва заповнена (Validate pack title);
- якщо все добре — клієнт надсилає запит на сервер (Send create request to server);
- Server створює порожній запис пака у базі даних (Create empty pack in DB);
- потім створює кореневу папку у сховищі R2 (Create root folder in R2);
- після успішного створення сервер повертає результат, і клієнт відображає порожню структуру пака (Display empty pack structure) та кнопки для редагування (Display add/delete buttons).

Помилкові потоки:

- відсутня назва пака: якщо користувач натискає кнопку створення без введення назви, клієнт показує повідомлення про помилку (Show error: no title);
- помилка створення пака: якщо сервер не може створити запис або папку, клієнт показує повідомлення про помилку (Show error: pack creation failed).

Альтернативні потоки (керування файлами/папками):

- додавання папки або файлу: користувач додає елемент, клієнт відправляє запит, сервер створює записи у DB та R2, клієнт оновлює відображення структури; У разі помилки показується відповідне повідомлення;
- видалення папки чи файлу: користувач натискає «Delete», клієнт надсилає запит, сервер видаляє записи у DB та R2, клієнт оновлює інтерфейс. Якщо операція не вдалася — показується повідомлення про помилку;
- програвання файлу: користувач вибирає файл, клієнт програє його, у разі помилки виводиться Show error: playback failed.

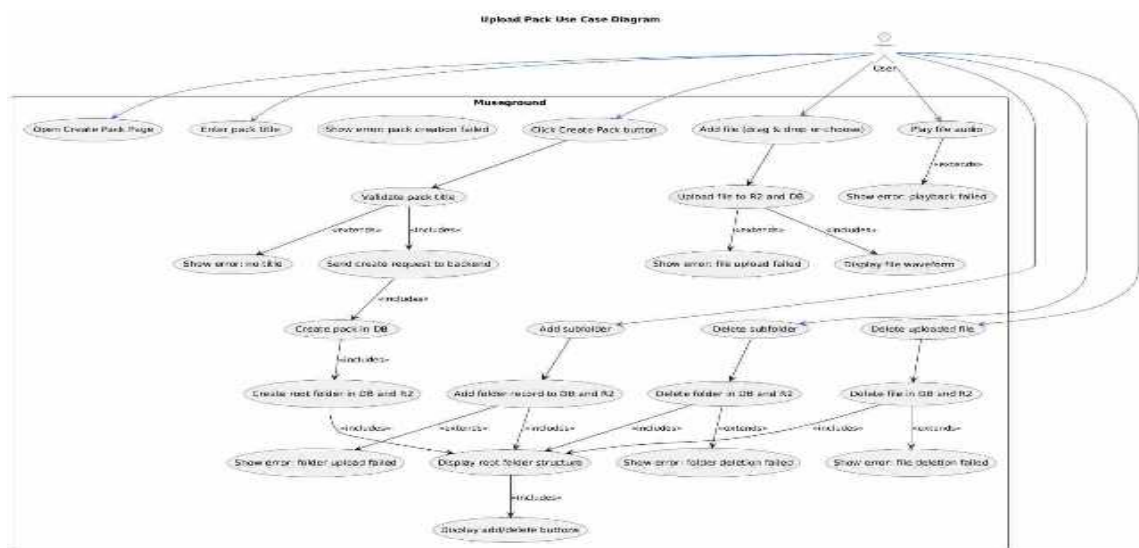


Рисунок 3.4 — Діаграма прецеденту upload pack

Далі наведено опис потоків подій для сторінки «explore samples».

Основний потік:

- користувач заходить на сторінку «explore sample»;
- сторінка завантажує усі семпли сайту;
- сторінка відображує форму для фільтрування;
- кожен семпл відображає кнопку play/pause, назву, візуалізіцію звукової хвилі, ім'я автора, тональність, інструменти, жанри, темп у bpm, слайдер гучності та слайдер пітча (висоти звуку);

– для кожного семплу, перевіряється чи є він збережений користувачем. Якщо семпл НЕ збережений, відображається кнопка «зберегти». Якщо семпл збережений, відображається лейбл «збережено» та кнопка «завантажити»;

– для кожного семплу, перевіряється чи є поточний користувач автором семплу. Якщо так, відображається лейбл «створено вами» та кнопка «видалити».

Альтернативні потоки:

– користувач може в будь-якому наборі та порядку вводити дані у форму для фільтрування семплів. Пошук семплів шукає семпли які задовольняють одразу усім введеним параметрам, якщо ті параметри було заповнено. При цьому, текстові поля підтримують частинний пошук. Якщо в полі «instruments» певного семплу вказано «acoustic guitar», і в поле інструменти в формі пошуку вказано «guitar», програма знайде цей семпл, за умови виконання інших параметрів пошуку. Для отримання результатів пошуку після введення параметрів користувачу потрібно клікнути на кнопку «search»;

– користувач може сортувати таблицю отриманих семплів за будь-яким з параметрів семплу, за зростанням або спаданням;

– користувач може слухати отримані семпли;

– якщо користувач ВЖЕ має певний семпл «збереженим», він може скасувати збереження;

– якщо користувач є автором семплу, семпл стає доступним для видалення.

Повідомлення про помилки виводяться:

– якщо не виходить виявити чи є користувач автором;

– якщо при завантаженні сторінки не було завантажено усі семпли;

– якщо пошук не знайшов семплів;

– якщо не вийшло видалити семпл

– якщо не вийшло застосувати фільтр

– якщо не вийшло завантажити збережений семпл;

- якщо не вийшло зберегти семпл;
- якщо не вийшло скасувати збереження семплу.

Діаграма прецедентів «explore samples» наведена на рисунку 3.5.

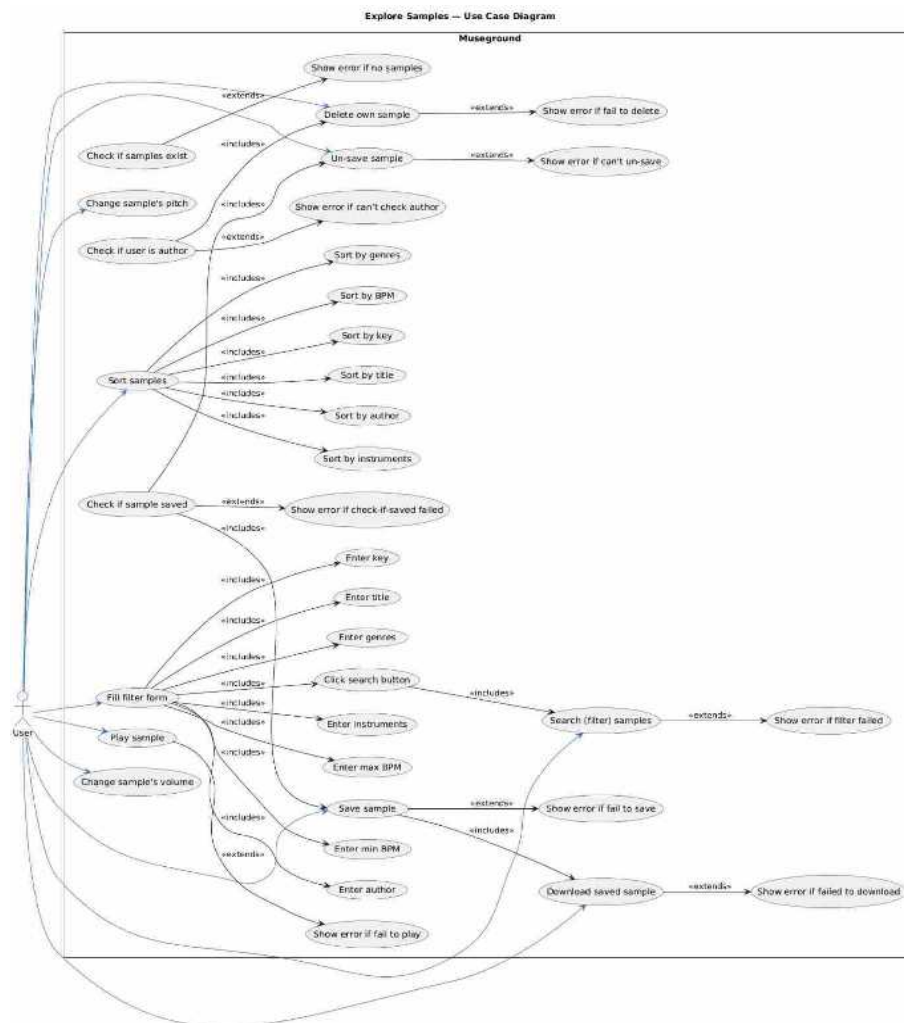


Рисунок 3.5 — Діаграма прецеденту explore samples

Опис потоків подій сторінки «created samples».

Основний потік:

- користувач відкриває сторінку «created samples»;
- сторінка завантажує семпли, створені поточним користувачем;
- сторінка відображає створені користувачем семпли у користувацькому інтерфейсі.

Кожен семпл відображає:

- доступні для редагування текстові поля назви, тональності, темпу, жанрів та інструментів семплу;

- кнопку для програвання/паузи аудіо семплу;

- візуалізацію звукової хвилі семплу;

- слайдери гучності та пітчу семплу;

- кнопку зберігання/скасування зберігання семплу;

- кнопку завантаження семплу;

- кнопку видалення семплу;

- кнопку «змінити аудіо файл»;

- кнопку «оновити інформацію».

Альтернативні потоки:

- користувач змінює текст в полях назви, тональності, темпу, жанрів, інструментів; користувач натискає кнопку «оновити інформацію»; оновлену інформацію відправлено до серверу;

- користувач заносить створений ним семпл до збережених або видаляє його з цього списку;

- користувач завантажує аудіо файл семплу;

- користувач натискає кнопку «змінити файл», обирає новий файл, новий файл оновлюється на сервері та виводиться на сторінці;

- користувач видаляє створений семпл.

Повідомлення про помилки виводяться якщо:

- при відкритті сторінки не вдалося завантажити створені користувачем семпли;

- не вдалося підвантажити/відобразити аудіо файл та візуалізацію хвилі;

- не вдалося зберегти чи скасувати збереження семплу;

- сталася помилка при завантаженні семплу;

- не вдалося видалити семпл;

- не вдалося оновити інформацію про семпл;

- не вдалося замінити аудіо файл семплу.

Діаграму прецедентів сторінки «created samples» наведено на рисунку 3.6. Також, на рисунку 3.7 наведено загальну абстрактну діаграму діяльності системи.

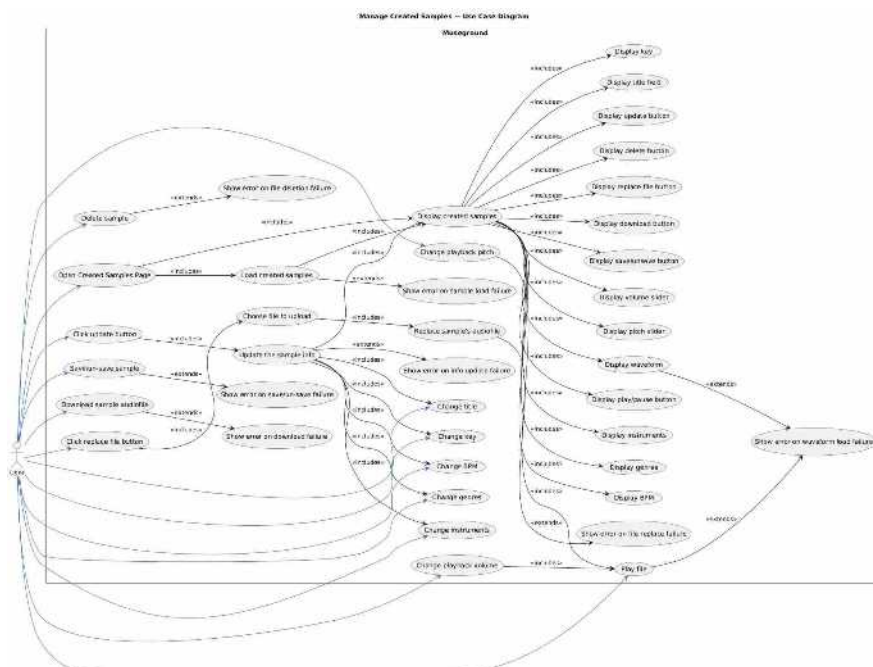


Рисунок 3.6 — Діаграма прецедентів сторінки created samples

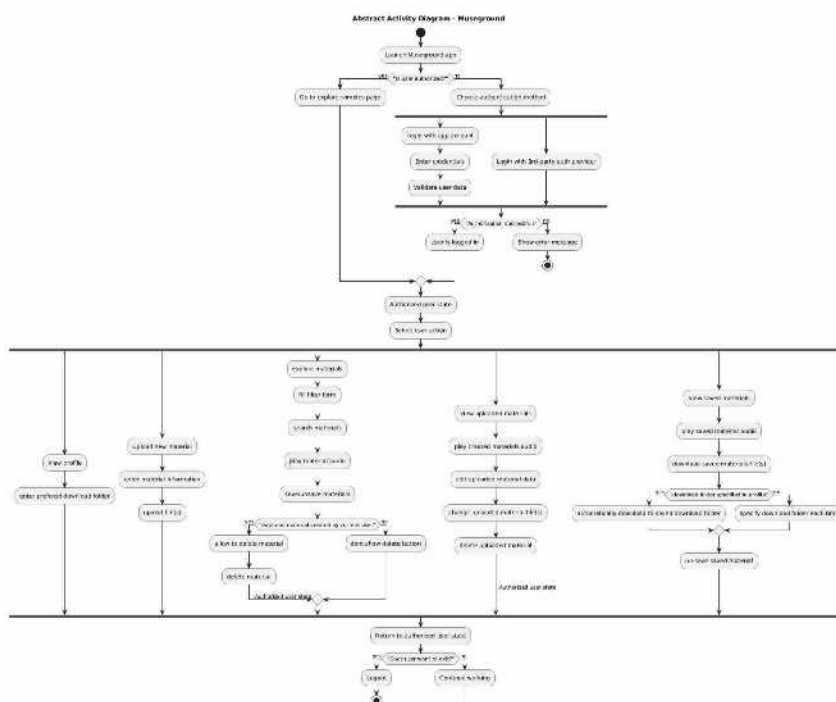


Рисунок 3.7 — Абстрактна діаграма діяльності системи

Далі наведено абстрактну діаграму діяльності системи, абстрактну діаграму кооперації системи, та діаграму послідовності системи.

3.2 Опис алгоритмів роботи програми

Як можна зрозуміти з попередній розділів, основними класами з яких складається розроблюваний застосунок є користувач, семпл, набір семплів та пресет. Основними процесами застосунку які взаємодіють з класом користувача є функціонал аутентифікації. Семпли, набори та пресети в свою чергу усі мають набір процесів які несуть спільний за сенсом функціонал для кожного з них — пошук користувачем, додавання користувачем до сайту, редагування або видалення.

Кожен з цих процесів включає в себе більше окремих дій, які відбуваються між клієнтом, сервером, базою даних та, в деяких випадках, хмарним сховищем даних або провайдером аутентифікації.

Тому, для ілюстрації цих процесів доцільно створити діаграми послідовностей UML, які добре підходять для відображення виконуваних застосунком дій, їх порядку, компонентів застосунку в яких вони відбуваються та їх потоків від одних до інших.

На наступних діаграмах (рисунки 3.8 — 3.15) зображено такі діаграми для демонстрації роботи аутентифікації застосунку, процесів пов'язаних з семплами та наборами. Операції з пресетами оминаються в даному розділі так як з точки зору застосунку єдина відмінність між ними та семплами це наявність двох файлів замість одного, тож діаграми семплів можна вважати описуючими і пресети також. Кожну з цих діаграм було створено інструментом PlantUML.

На рисунку 3.8 зображено послідовність операцій авторизації

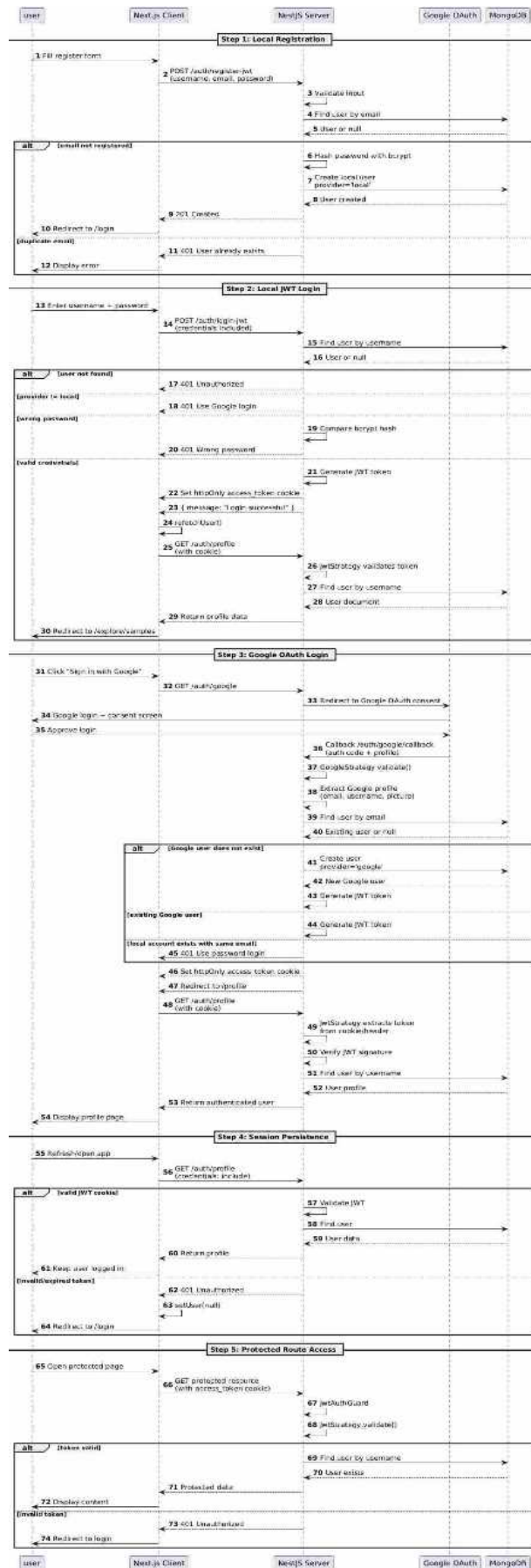


Рисунок 3.8 — Діаграма послідовності операцій аутентифікації

Рисунок 3.9 зображує діаграму послідовності можливих дій з семплами.

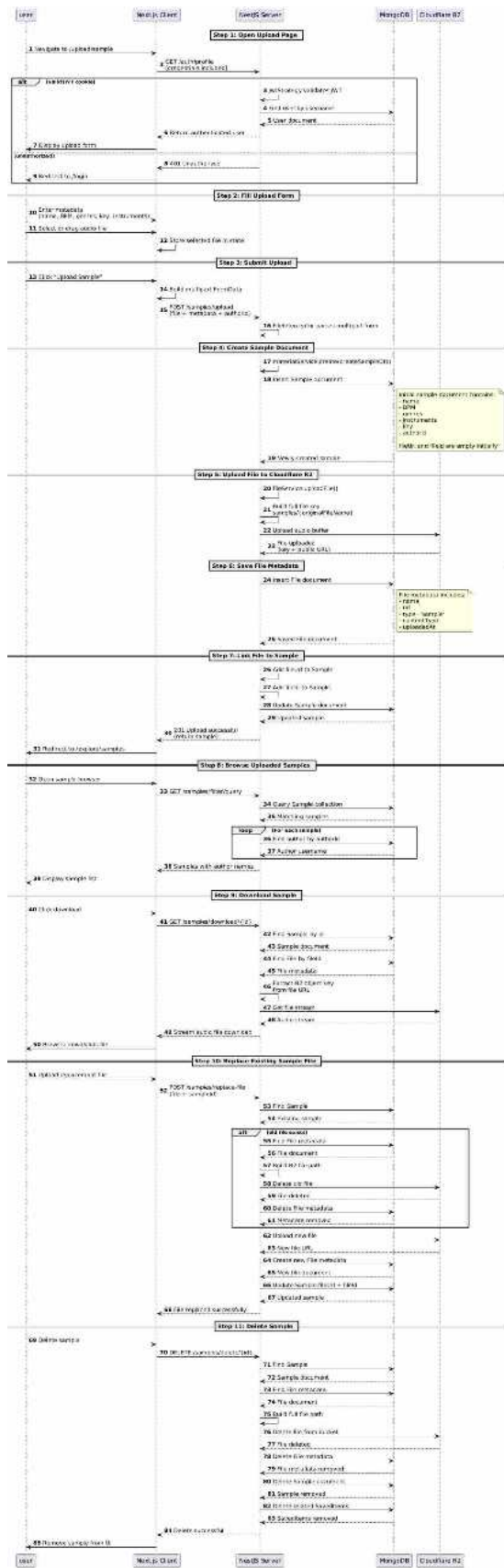


Рисунок 3.9 — Діаграма послідовності дій з семплами

Рисунки 3.10-3.12 зображують першу, другу та третю частини діаграми послідовності усіх можливих дій з наборами.

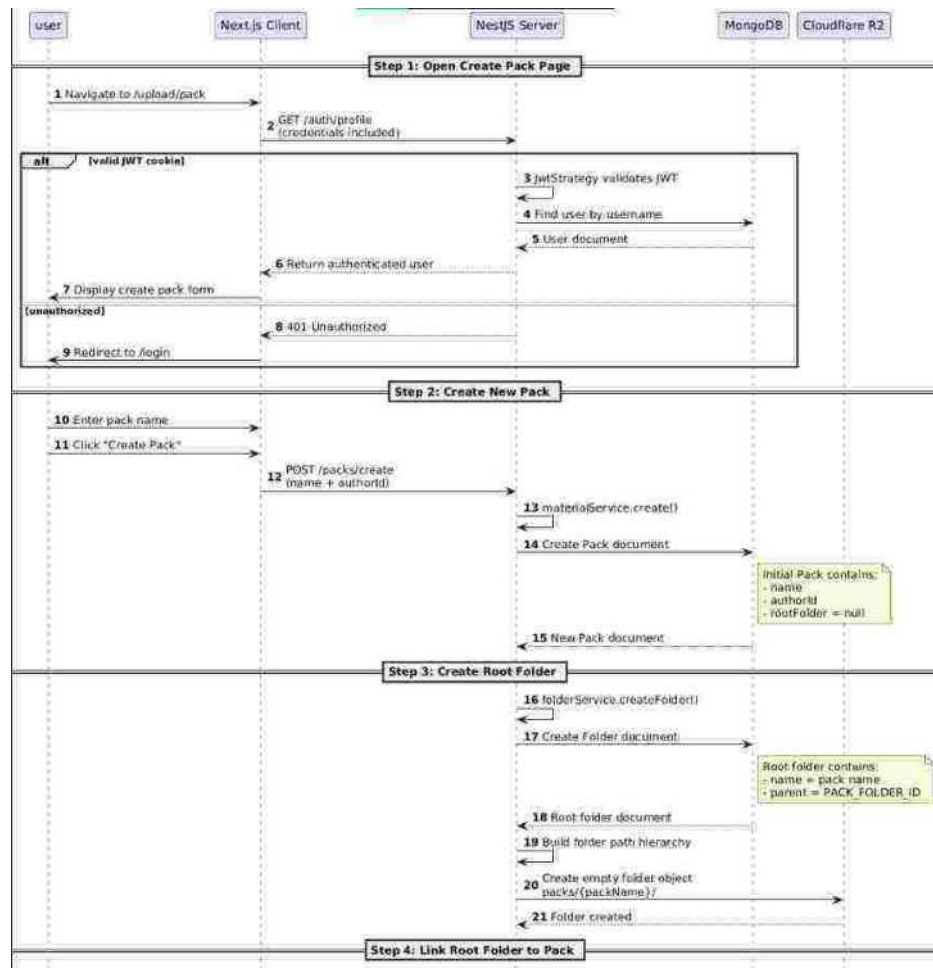


Рисунок 3.10 — Діаграма послідовності дій з наборами, частина 1

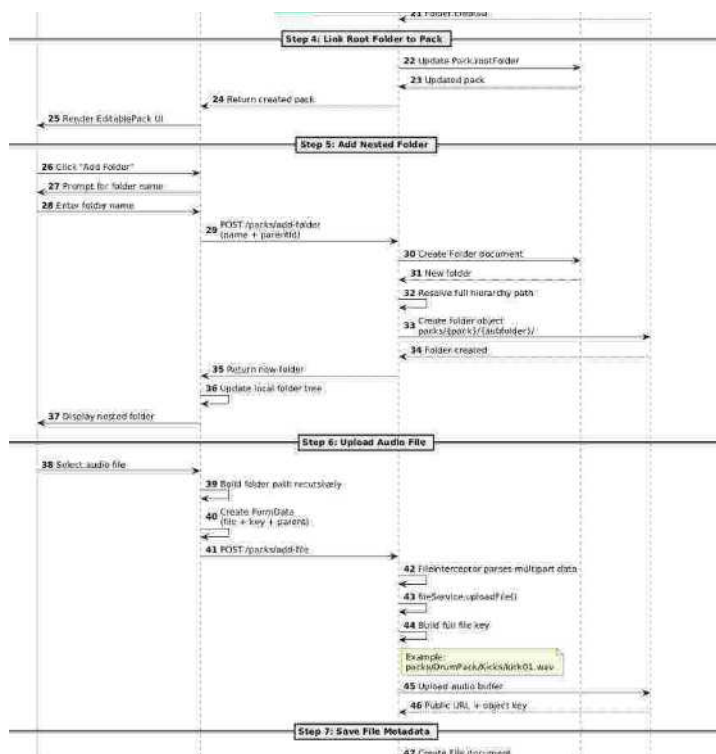


Рисунок 3.11 — Діаграма послідовності дій з наборами, частина 2

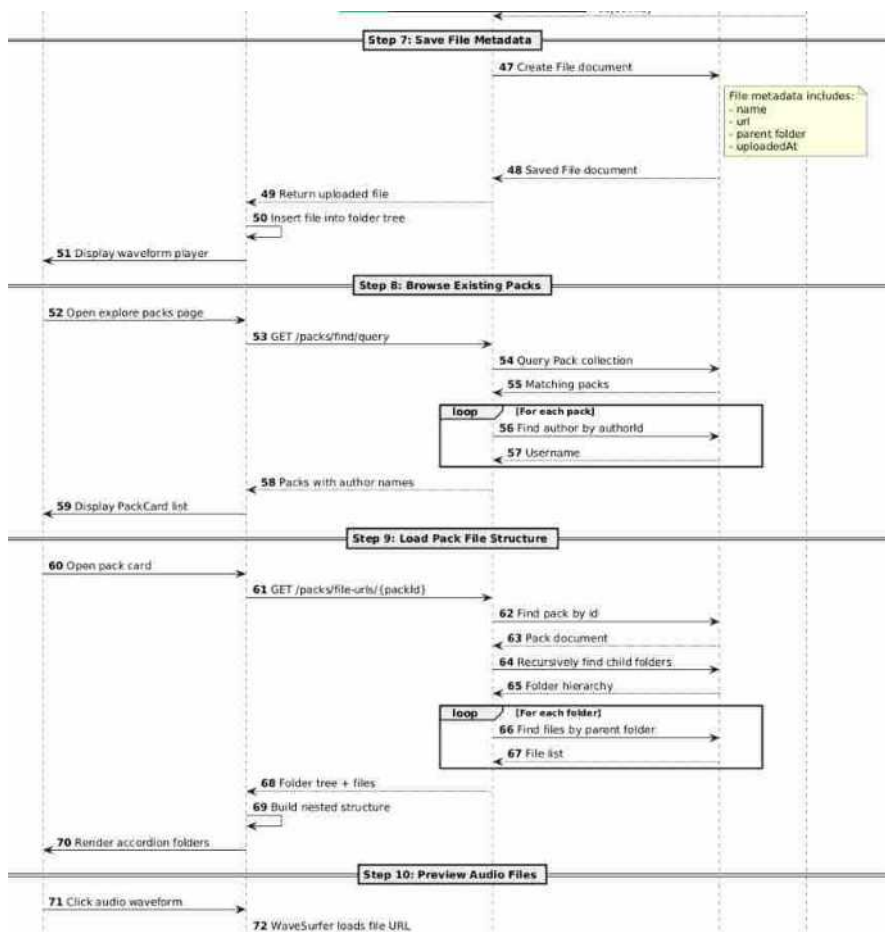


Рисунок 3.12 — Діаграма послідовності дій з наборами, частина 3

Рисунки 3.13-3.15 зображують частини від 4 до 6 діаграми послідовності усіх можливих дій з наборами.

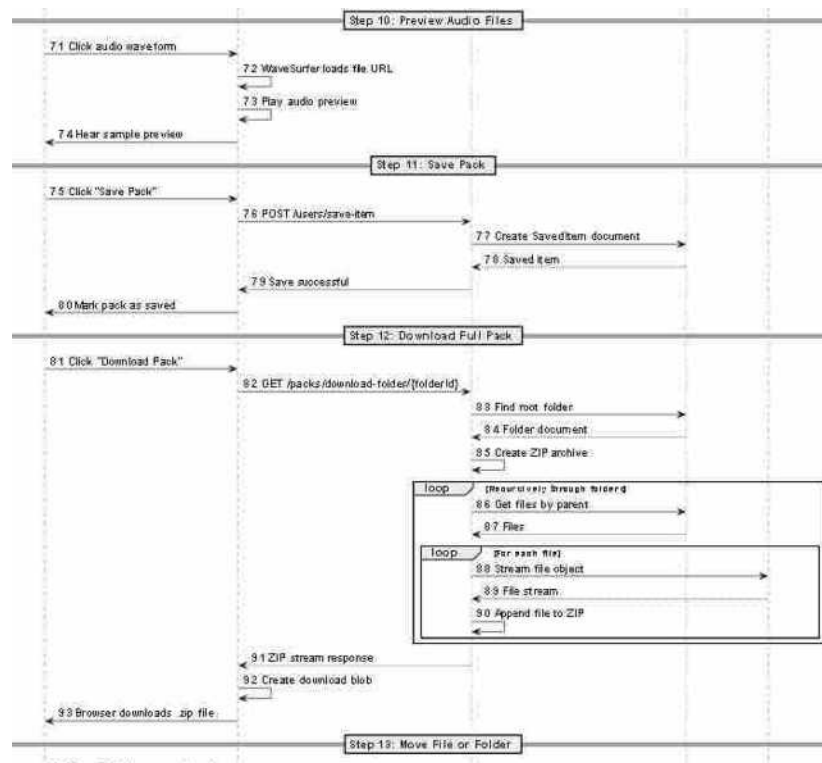


Рисунок 3.13 — Діаграма послідовності дій з наборами, частина 4

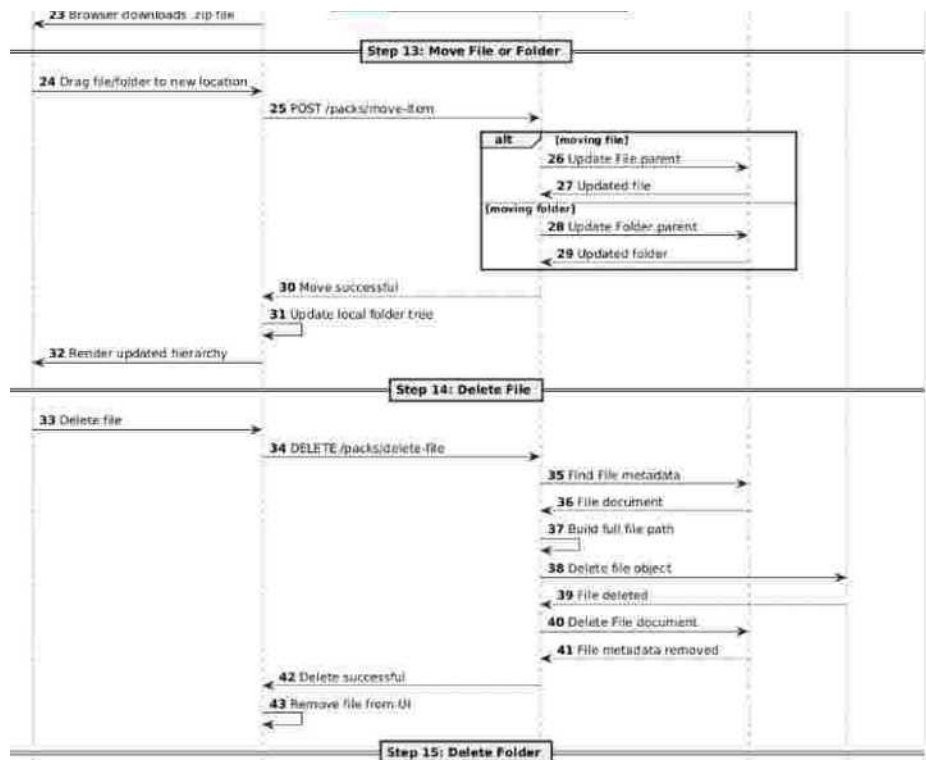


Рисунок 3.14 — діаграма послідовності дій з наборами, частина 5

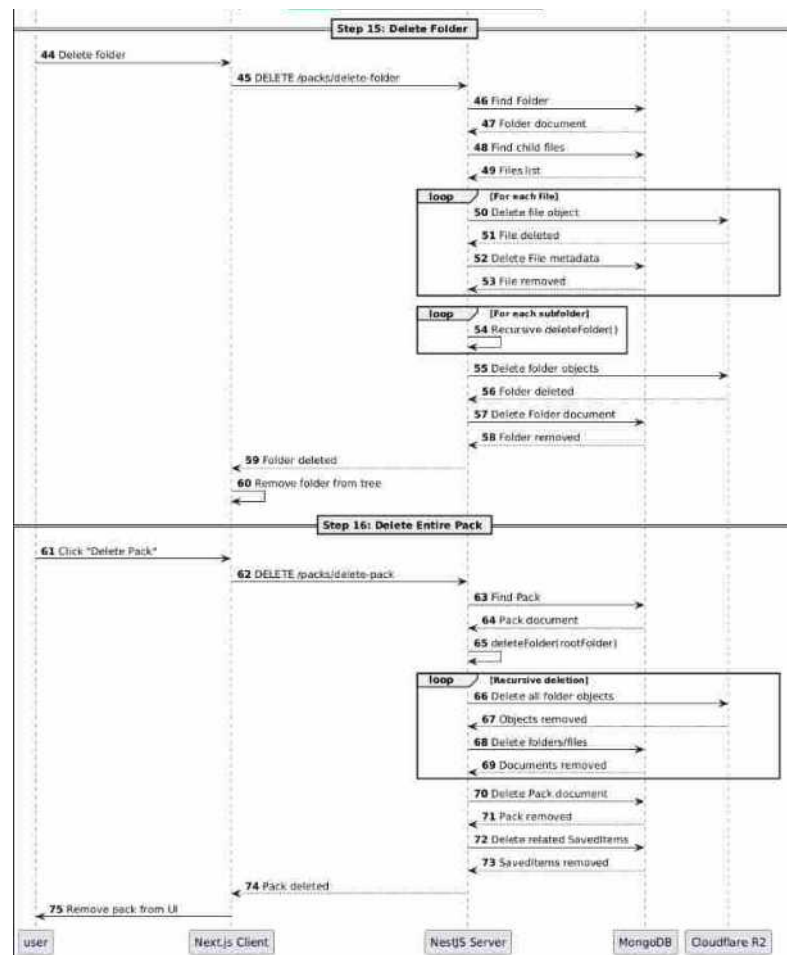


Рисунок 3.15 — Діаграма послідовності дій з наборами, частина 6, остання

3.3 Проєктування бази даних застосунку

Внаслідок аналізу предметної області та висунутих вимог було розроблено структуру бази даних що є достатньою для виконання поставлених цілей та достатньо зрозумілою щоб її можна було назвати простою. У предметній області можна відразу зазначити наступні сутності:

- User, яка репрезентує користувача;
- Sample, для семплів;
- Pack, для наборів семплів;
- Preset, для пресетів.

Також, через функціональні вимоги до проєкту виникає необхідність створення додаткових сутностей:

- File, зберігає назву шлях та зв'язки для файлів семплів, пресетів та змісту наборів;
- Folder, для репрезентації папок з яких складаються набори;
- SavedItem, для репрезентації Many-to-Many відношення між збереженими ресурсами та користувачами;

За результатами розділу 2 було визначено що в даній розробці буде використано СУБД Mongo Atlas, та для взаємодії з нею у серверній частині застосунку буде використано ODB Mongoose [20]. Варто зазначити що MongoDB не є традиційною реляційною базою даних з заздалегідь чітко визначеними таблицями, і здатна зберігати дані різної форми в якості документів. Втім, Mongoose все ж надає розробникам створювати заздалегідь визначені «схеми» даних, які і використовуються для визначення форми даних різних сутностей застосунку. У наведених далі таблицях 3.1 — 3.7 наведено структуру цих схем створених в ході розробки даного застосунку.

Далі, на рисунку 3.17 наведено ER схему спроектованої бази даних, а на рисунках 3.18 — 3.24 наведено скріншоти хмарної бд mongo cloud які демонструють вигляд документів кожної сутності в базі даних.

Таблиця 3.1 наводить схему User та таблиця 3.2 — Sample.

Таблиця 3.1 – Схема User

Назва поля	Тип даних	Обов'язковість	Опис
Username	String	Так	Унікальне ім'я користувача
Email	String	Так	Унікальна адреса пошти користувача
Password	String	Ні	Хеш пароля користувача
Provider	'local' 'google'	Так	Спосіб аутентифікації
Totaldownloads	Number	Так	Кількість завантажених ресурсів
Totaluploads	Number	Так	Кількість доданих до сайту ресурсів

Таблиця 3.2 – Схема Sample

Назва поля	Тип даних	Обов'язковість	Опис
Name	String	Так	Назва семплу
Fileurl	String	Ні	Шлях до файлу семплу в хмарі
Authorid	Objectid	Так	Ідентифікатор користувача який додав семпл до сайту
Bpm	Number	Ні	Beats per minute – темп семплу
Instruments	String	Ні	Список інструментів задіяних в семплі
Genres	String	Ні	Список жанрів якими можна ідентифікувати семпл
Key	String	Ні	Музична тональність семплу
Fileid	Objectid	Так	Ідентифікатор запису про файл семпла в базі даних

Таблиця 3.3 зображує схему даних таблиці Preset, таблиця 3.4 — схему таблиці Pack, таблиця 3.5 схему таблиці File та таблиця 3.6 схему таблиці Folder.

Таблиця 3.3 – Схема Preset

Назва поля	Тип даних	Обов'язковість	Опис
Name	String	Так	Назва пресету
Fileurl	String	Так	Шлях до файлу
Soundfileurl	String	Так	Шлях до аудіо файлу що демонструє пресет
Authorid	Objectid	Так	Ідентифікатор користувача який додав пресет до сайту
Vst	String	Ні	До якого vst-плагіну належить пресет
Genres	String	Ні	Список жанрів якими можна категоризувати пресет

Продовження таблиці 3.3

Назва поля	Тип даних	Обов'язковість	Опис
Types	String	Ні	Список типів звуку (з точки зору саунд-дизайну) якими можна ідентифікувати семпл
Soundfileid	Objectid	Так	Ідентифікатор запису про демонстраційний файл пресета
Fileid	Objectid	Так	Ідентифікатор запису про файл пресета

Таблиця 3.4 – Схема Pack

Назва поля	Тип даних	Обов'язковість	Опис
Name	String	Так	Назва набору
Authorid	String	Так	Ідентифікатор користувача автора набору
Rootfolder	Objectid	Так	Ідентифікатор запису про корньову папку

Таблиця 3.5 – Схема File

Назва поля	Тип даних	Обов'язковість	Опис
Name	String	Так	Ім'я файлу
Url	String	Так	Адреса файлу в хмарі
Parent	Objectid	Ні	Опціональний ідентифікатор папки

Таблиця 3.6 – Схема Folder

Назва поля	Тип даних	Обов'язковість	Опис
Name	String	Так	Ім'я папки
Parent	Objectid	Ні	Опціональний ідентифікатор батьківської папки

Таблиця 3.7 зображує схему даних таблиці SavedItem. Також, на рисунку 3.16 наведено діаграму усіх сутностей в спроектованій базі даних.

Таблиця 3.7 – Схема SavedItem

Назва поля	Тип даних	Обов'язковість	Опис
Userid	Objectid	Так	Ідентифікатор користувача який зберіг ресурс
Itemtype	'sample' 'preset' 'pack'	Так	Тип збереженого ресурсу
Itemid	Objectid	Так	Ідентифікатор збереженого ресурсу
Savedat	Date	Так	Дата збереження

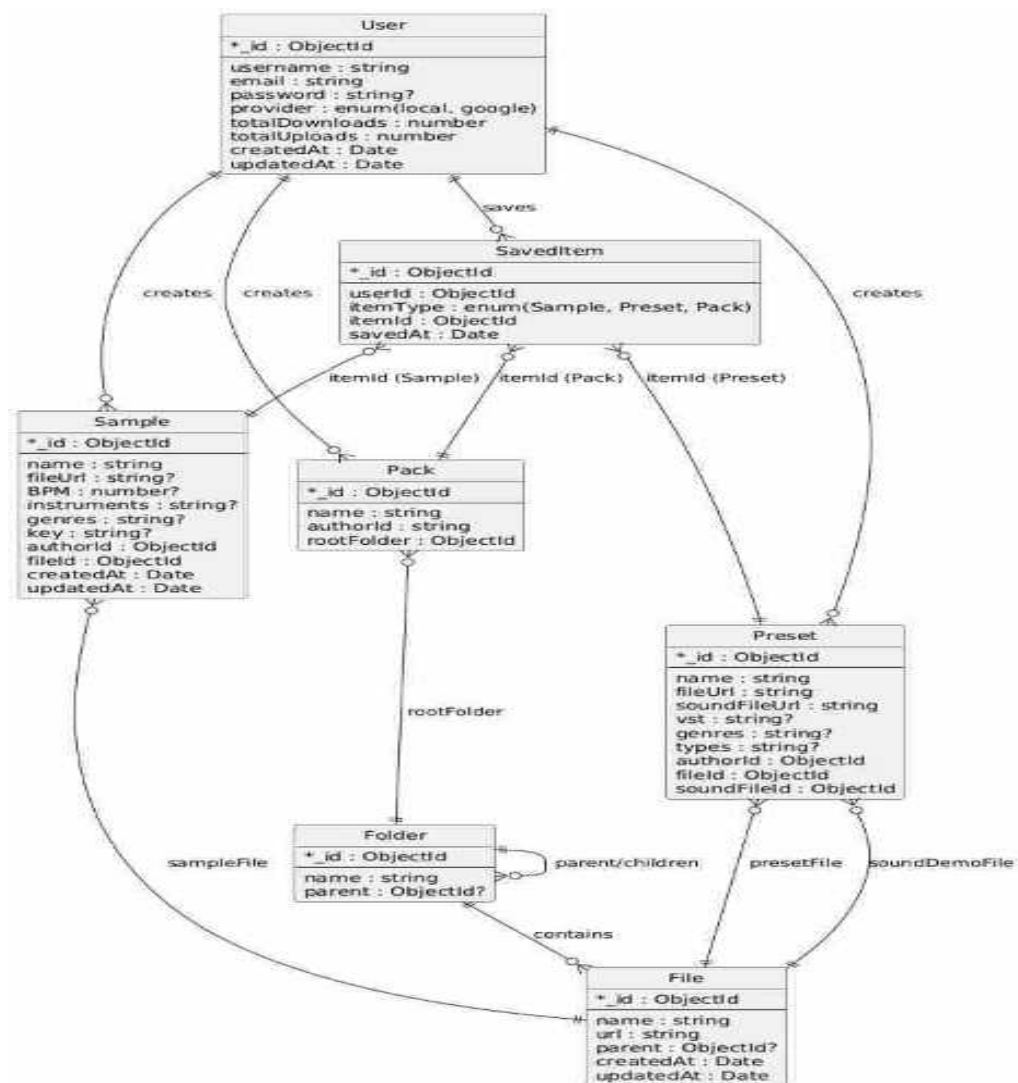


Рисунок 3.16 — Діаграма сутностей бази даних

Рисунки 3.17 та 3.18 зображують вигляд даних таблиць Files та Folders в хмарній базі даних Mongo Atlas.

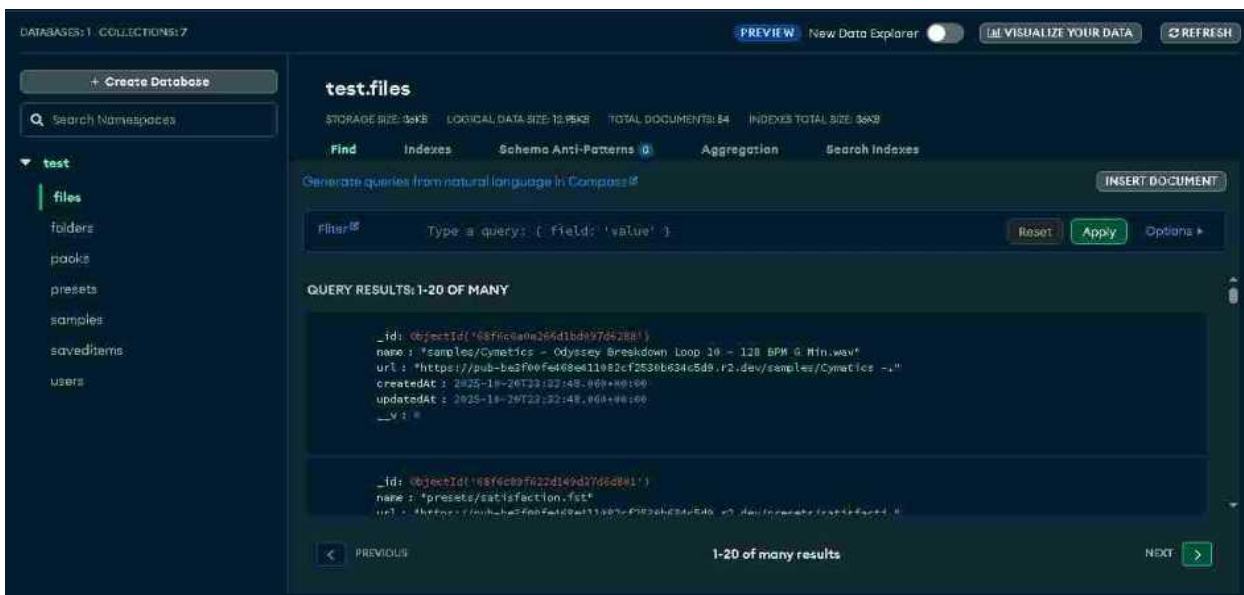


Рисунок 3.17 — Документи схеми Files

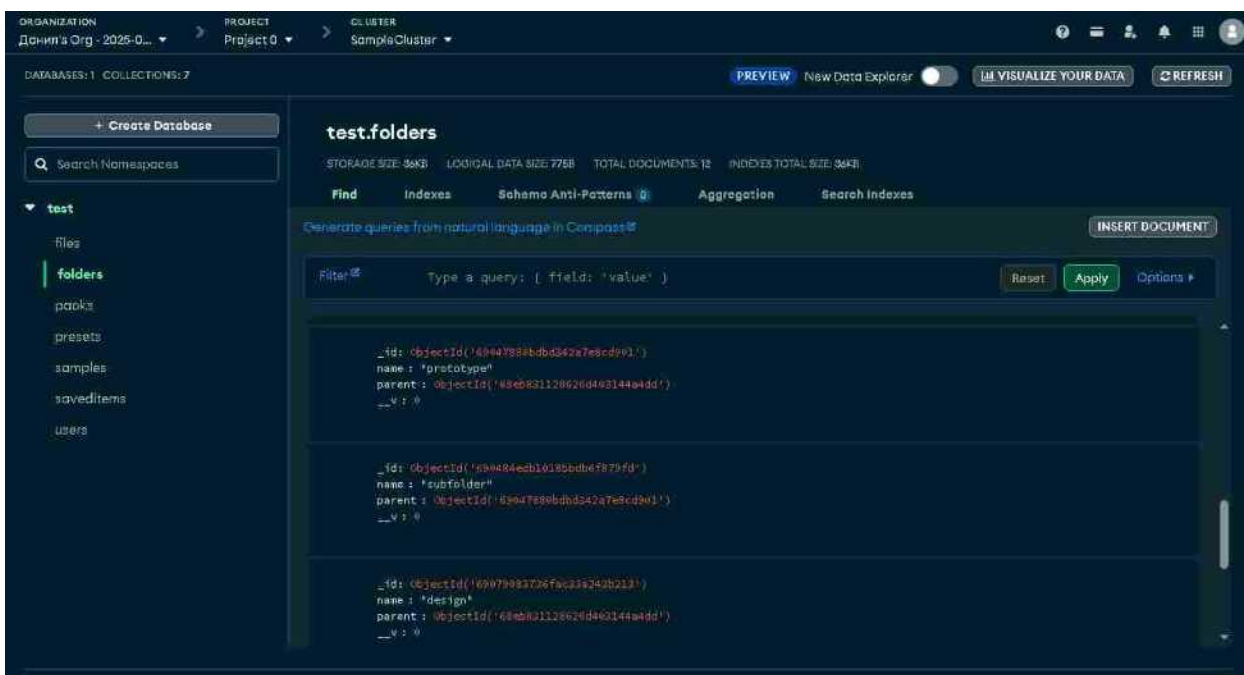


Рисунок 3.18 — Документи схеми Folders

Рисунки 3.19 та 3.20 зображують вигляд даних таблиць Packs та Presets в хмарній базі даних Mongo Atlas.

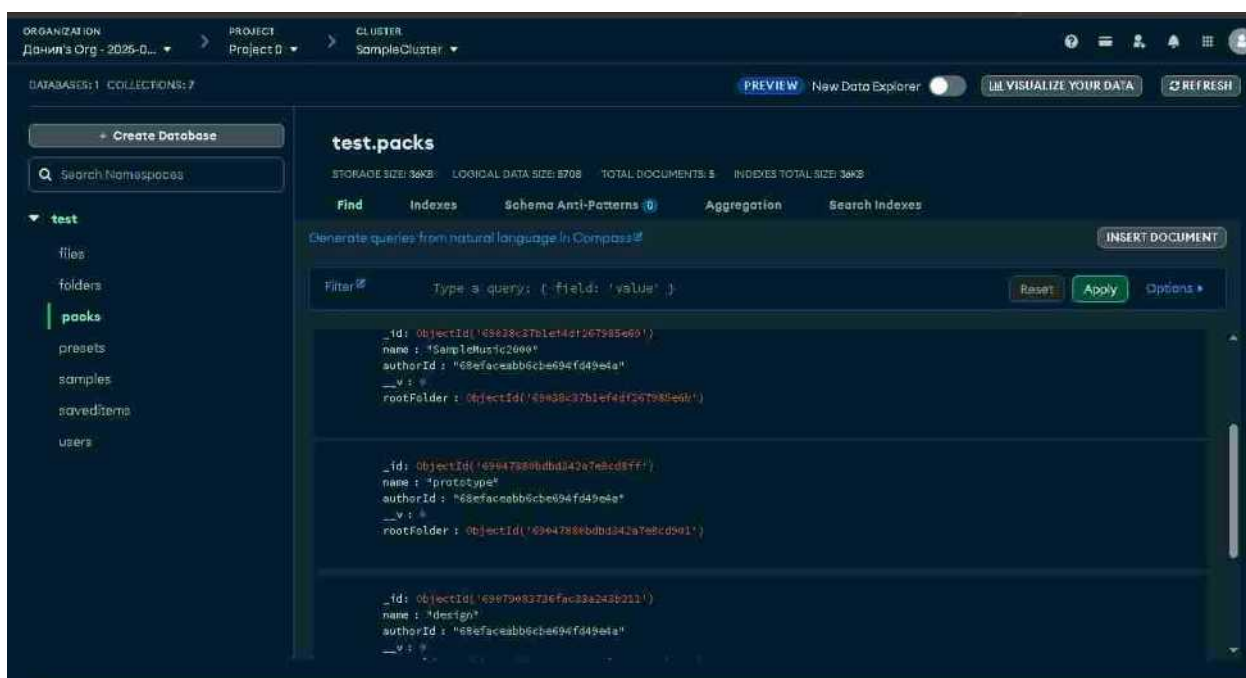


Рисунок 3.19 — Документи схеми Packs

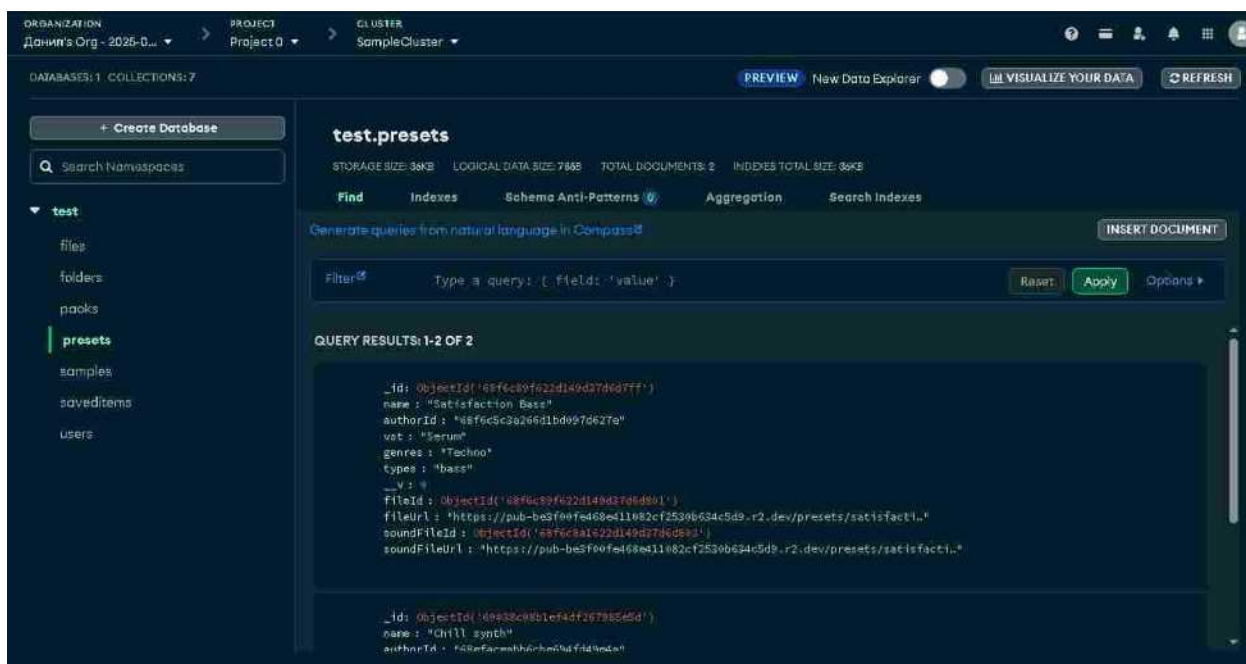


Рисунок 3.20 — Документи схеми Presets

Рисунки 3.21 та 3.22 зображують вигляд даних таблиць Samples та SavedItems в хмарній базі даних Mongo Atlas.

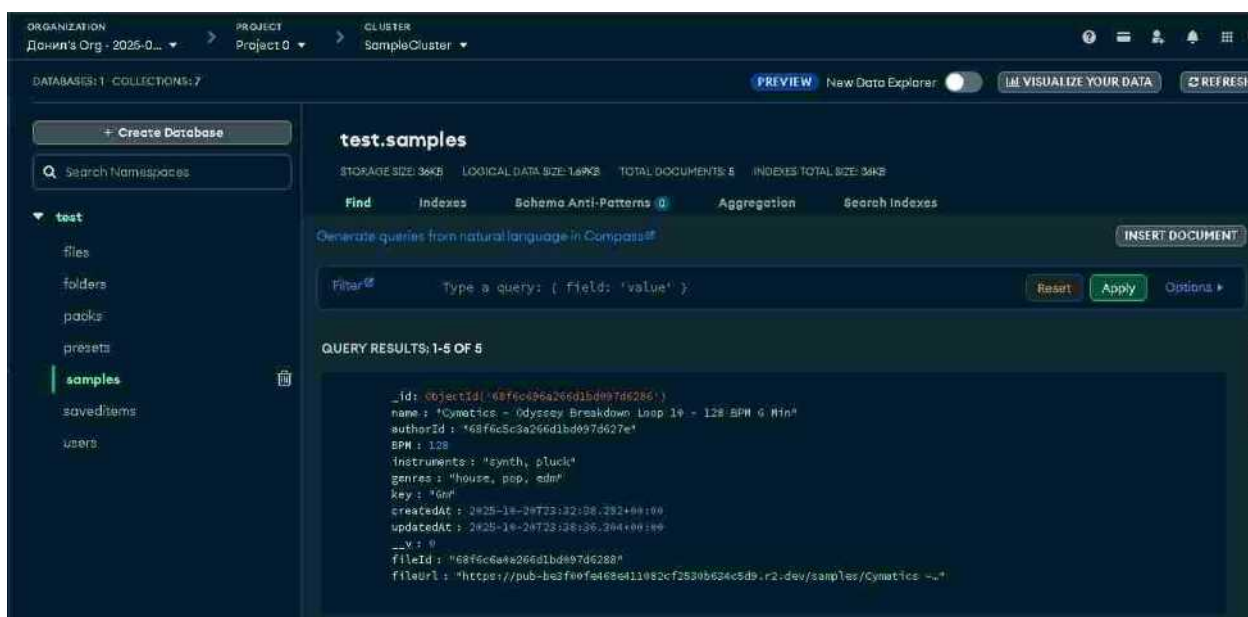


Рисунок 3.21 — Документи схеми Samples

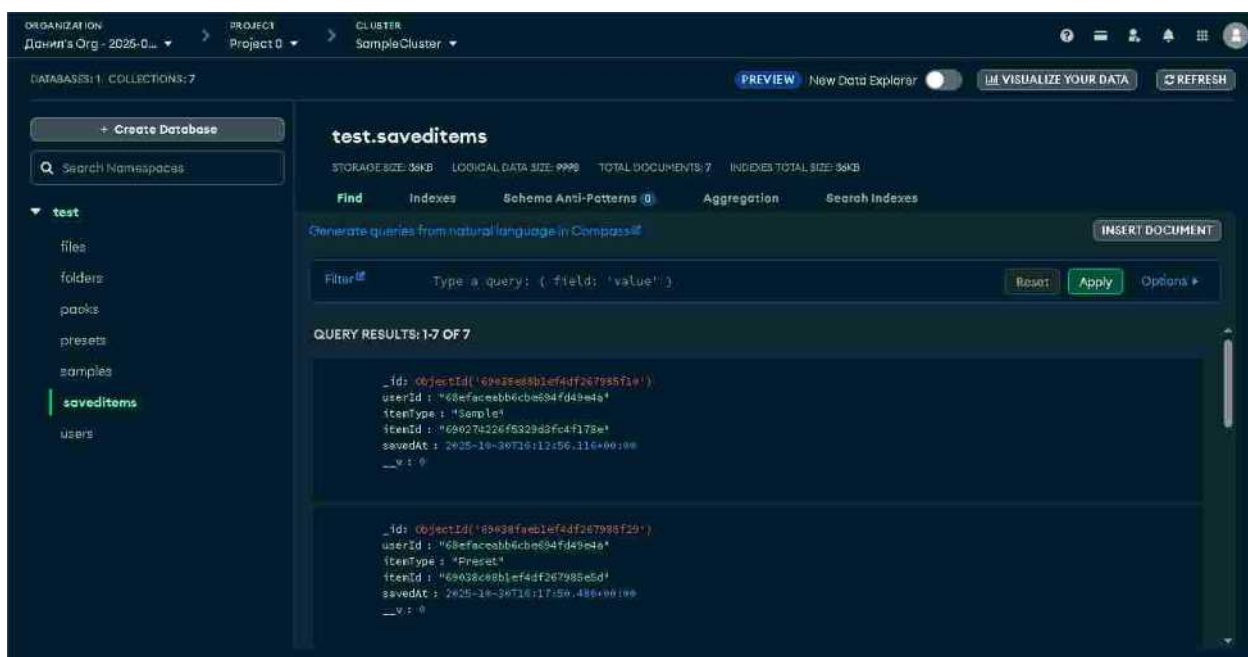


Рисунок 3.22 — Документи схеми SavedItems

Рисунок 3.23 зображує вигляд даних таблиці Users в хмарній базі даних Mongo Atlas.

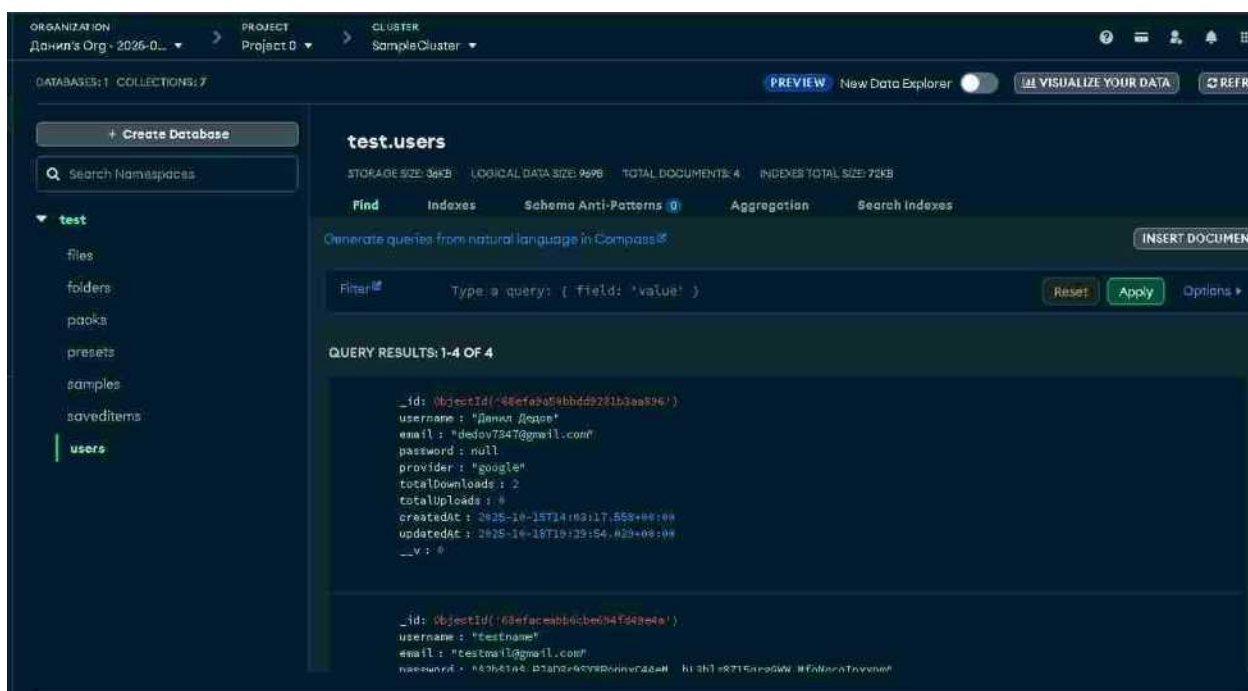


Рисунок 3.23 — Документи схеми Users

3.4 Графічний користувацький інтерфейс

3.4.1 Схема роботи інтерфейсу системи

При вході на сайт, користувач буде потрапляти на сторінку `explore/samples`. Далі наведено елементи інтерфейсу які матиме ця сторінка. Більшість сторінок інтерфейсу застосунку матимуть схожу структуру, тому повний опис сторінки буде наведено лише для `/explore/samples`, а для наступних подібних сторінок буде описано лише відмінності:

- панель навігації згори сторінки яка містить посилання `/login`, `/profile`, `/saved`, `/upload`, `/created`; У разі якщо користувач не ввійшов у систему, лише `login` буде активним посиланням; Аналогічна панель буде знаходитись на кожній сторінці застосунку;

- панель навігації одразу нижче попередньої, з посиланнями на `explore/samples` (активна), `explore/packs`, `explore/presets`. Аналогічна панель навігації буде присутня на усіх сторінках пов'язаних з діями які стосуються усіх трьох видів матеріалів (тобто на сторінках `saved`, `upload`, `uploaded`);

- форма для пошуку фільтрування результатів. Ця форма буде містити поле для пошуку, випадальний список який визначає параметр поля пошуку — назву, ім'я автора або тег, випадальні списки для фільтру за параметрами семплу, тобто жанром, темпом, тональністю, тегами;

- випадальний список для вибору параметра за яким будуть відсортовані результати, та чекбокс для сортування за зростанням/спаданням;

- кнопку «Шукати» та кнопку «Застосувати Фільтр»;

- список результатів, у вигляді таблиці, де кожний рядок це один з результатів, і стовпчики є інформацією про нього, а саме назвою, автором, довжиною, темпом, тональністю, інструментом та жанром. Окрім цього, кожен екземпляр буде мати виділене місце, в якому будуть розташовані кнопка «play», візуалізація звукової хвилі, слайдер гучності, кнопки підвищення/пониження висоти звуку, кнопка «зберегти» та список тегів.

Сторінка `explore/presets` відрізняється від `explore/samples` по суті лише параметрами якими описуються об'єкти. Замість описаних параметрів семплів, пресети мають назву, автора, назву VST плагіну, для якого зроблено пресет, тип звуку (наприклад `bass`, `pad`, `lead`) та жанр.

Сторінка `explore/packs` відрізняється тим, що список наборів буде відображатися не у вигляді таблиці, а як набір інформаційних карток, кожна з яких буде відображати окремий набір, і буде мати назву, автора, та аудіо файли що містяться у наборі, з можливістю почути приклади одразу в застосунку. Кожен набір здатен мати дерево папок всередині, і для їх відображення контейнери наборів в інтерфейсі матимуть елементи інтерфейсу типу «акордеон», натискання на які буде відкривати та закривати окремі папки всередині наборів.

Сторінки розділів `saved` та `created` матимуть схожу до розділу `explore` структуру. Різниця полягатиме в тому, що ці розділи замість кнопок «save» матимуть кнопки «download» та/або «delete». Значну різницю матиме сторінка `saved/packs`, яка буде відображати збережені набори у вигляді дерева директорій та файлів, де є корньова директорія, яка містить директорії які

відображають кожен збережений набір, які в свою чергу містять власні директорії, файли та піддиректорії. Також, головною різницею сторінок типу /created є той факт, що всі інформаційні поля та поля файлів про різні ресурси на цих сторінках будуть доступними для редагування користувачем, і до них будуть додані кнопки «save» для оновлення цих даних та «delete» для перманентного видалення вказаного ресурса з застосунку.

Сторінка upload/sample буде містити форму для вводи даних про семпл, описаних попередньо, а також кнопку для додавання аудіо файлу, та кнопки «додати» (upload). Сторінка upload/preset працює аналогічним чином.

Сторінка upload/pack буде мати два способи роботи. В першому, користувач завантажує папку, яка існує на його комп'ютері, назва папки стає назвою набору, і вміст папки стає його вмістом. В другому, користувач вручну вводить назву набору та вручну створює структуру набору, створюючи папки всередині цієї сторінки та додаючи до них окремі файли.

Сторінка profile буде містити інформацію про обліковий запис користувача, таку як ім'я, кількість доданих та завантажених матеріалів, тощо. Також, ця сторінка буде містити кнопку виході з облікового запису.

Сторінка login буде містити форму для входу посередині екрану, в якій будуть поля для введення ім'я та паролю, кнопки «login», «register» та кнопки для входу через google або github. Сторінка register працюватиме аналогічним чином.

3.4.2 Проєктування користувацького інтерфейсу

В ході цієї роботи було виконано дизайн інтерфейсів, створення вайрфреймів та мокапу за допомогою веб застосунку figma [21]. Також, для однієї сторінки інтерфейсу було створено інтерактивний прототип за допомогою програми ProtoPie [22]. Нижче за текстом наведено посилання на кожен зі створених дизайнів, скріншот всього проєкту у figma (рис. 3.25),

скріншоти кожного окремого дизайну (рис. 3.26-3.39), а також посилання на відео інтерактивного прототипу однієї зі сторінок.

В ході роботи над прототипом дизайну цієї сторінки у figma було створено ряд коментарів нотаток щодо деталей реалізації сторінки інтерфейсу, таких як відступи, кольори, поведінка компонентів інтерфейсу тощо. Оскільки ці коментарі несуть ціль простих нагадувань під час розробки, аніж документації, їх було зроблено англійською мовою для зручності розробки. Ці коментарі доступні для перегляду за посиланням на проєкт в figma, але також їх зміст наведено у наступному списку:

- all headers use Suprapower font, 24 to 32 size;
- top margin 40 for results table;
- bottom margin 24 between form input field rows;
- results table rows have zinc-900 background, but change to 800 on hover;
- most darkened elements all have tailwind's zinc-800 bg-color, including
- header, inputs background etc;
- 12 px left-right margin on entire page;
- all bordered elements have 8px border-radius;
- label in active input field is sliiightly higher than in non-active field that has text;
- with waveform, user can choose start point of playback. When audio playing, wave turns from cyan to blue;
- users can save even the samples created by themselves;
- if a sample was saved, it can be downloaded from this page, as well a saved page;
- if sample was created by user, they can delete it right on this page;
- pitch slider goes back to default when mouse-up;
- waveform rendered through wavesurfer.js [23];
- all usual text uses programme font, 14-16px;
- all headers use Suprapower font, 24 to 32 size;
- all accented elements use tailwind's cyan-400 (#00D3F2) color;

- when input field is empty - the label takes "placeholder" place;
- input field has inputted text - label/placeholder comes up and becomes cyan;
- if input field was auto-filled - border becomes blue;
- if item is created by user - background blue. Here the sliders are an image and just can't change background.

Рисунок 3.24 зображує загальний вигляд проєкту в Figma, рисунок 3.25 зображує детальний макет сторінки Samples. Далі, рисунки 3.26 та 3.27 зображують вайрфрейми сторінок логіну та реєстрації.

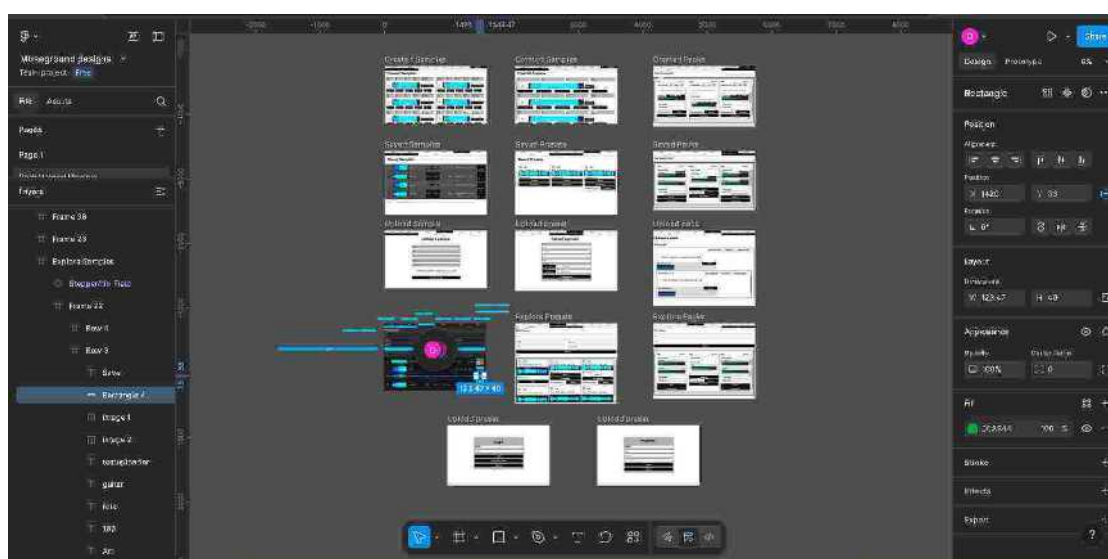


Рисунок 3.24 — Проєкт макету та вайрфреймів у figma



Рисунок 3.25 — Детальний макет сторінки /explore/samples

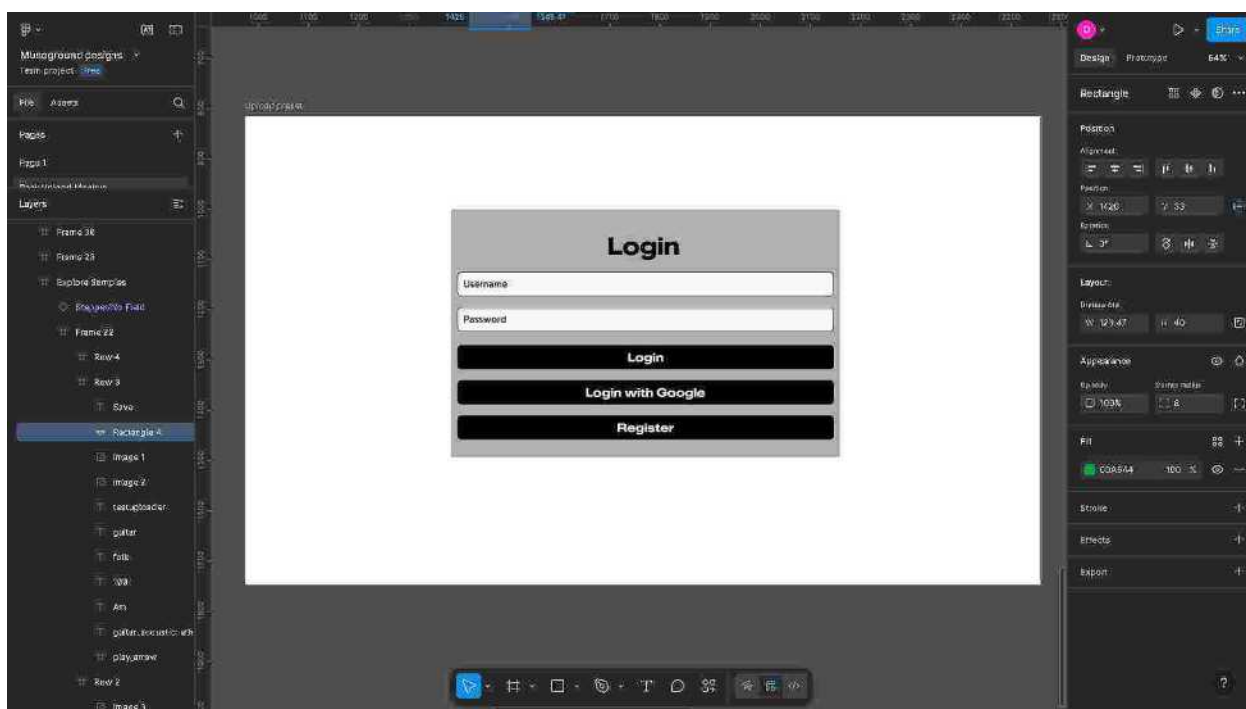


Рисунок 3.26 — Вайрфрейм сторінки логіну

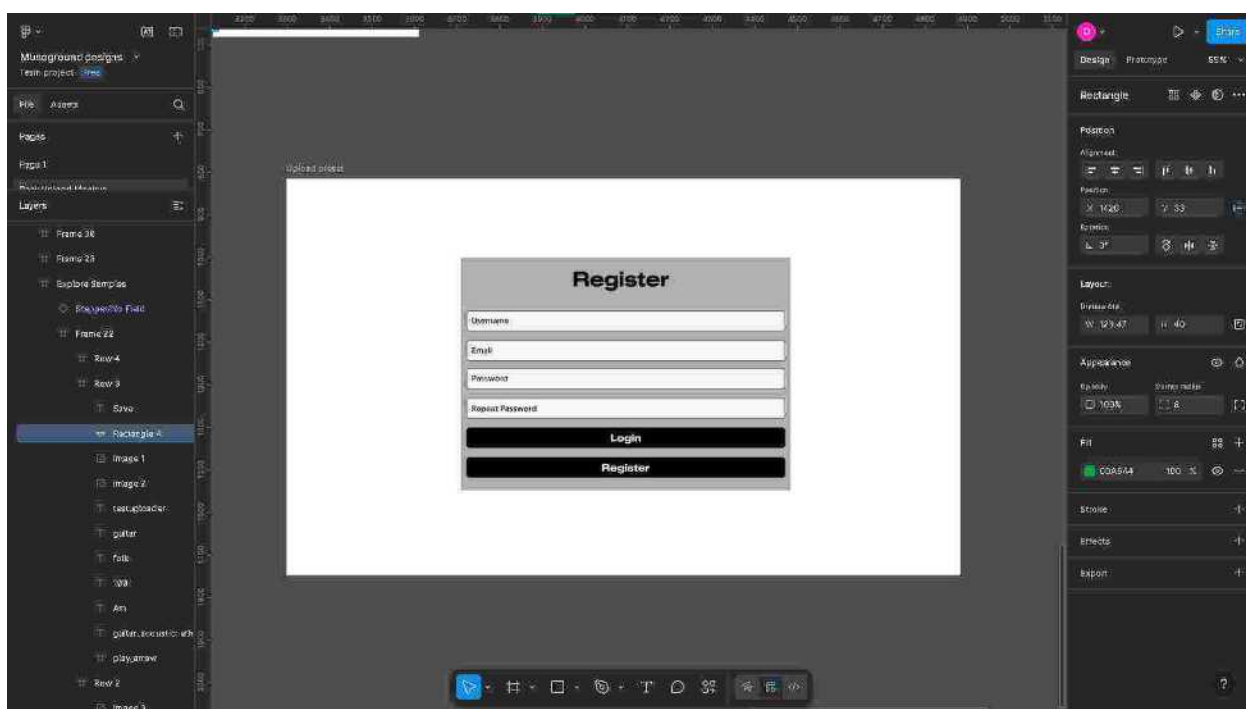


Рисунок 3.27 — Вайрфрейм сторінки реєстрації

Далі, рисунки 3.28 та 3.29 зображують вайрфрейми сторінок explore presets та explore packs.

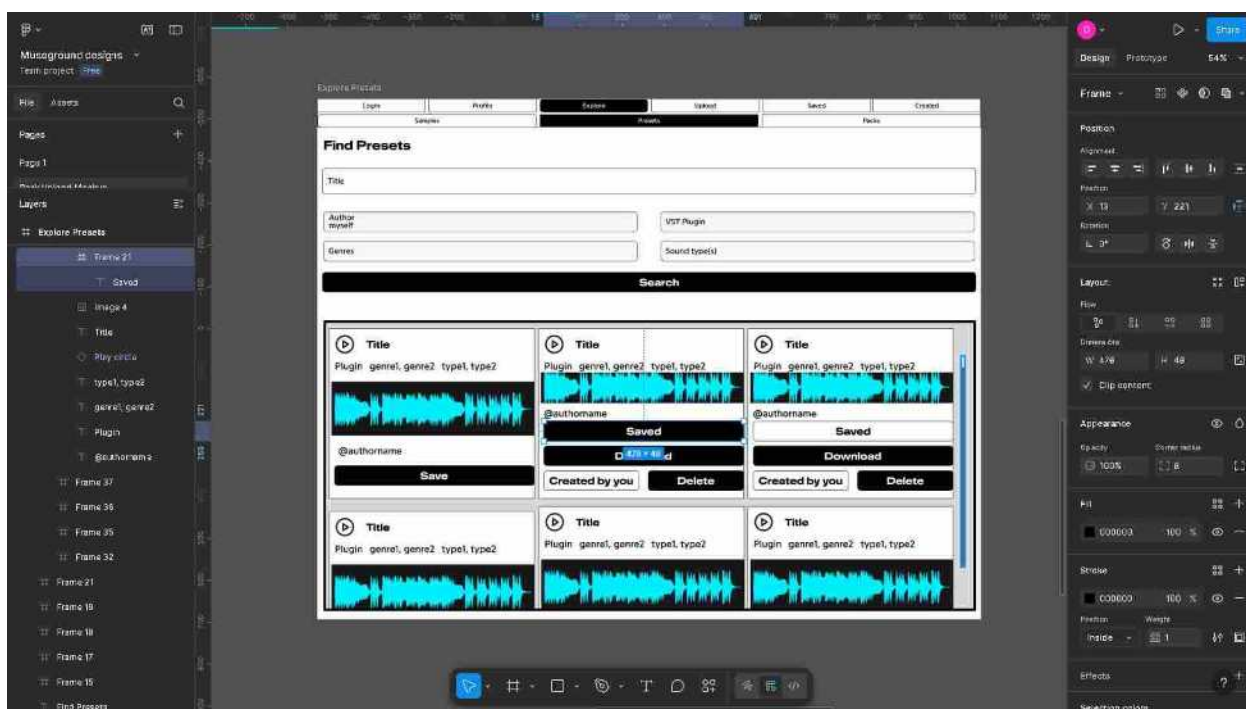


Рисунок 3.28 — Вайрфрейм сторінки explore presets

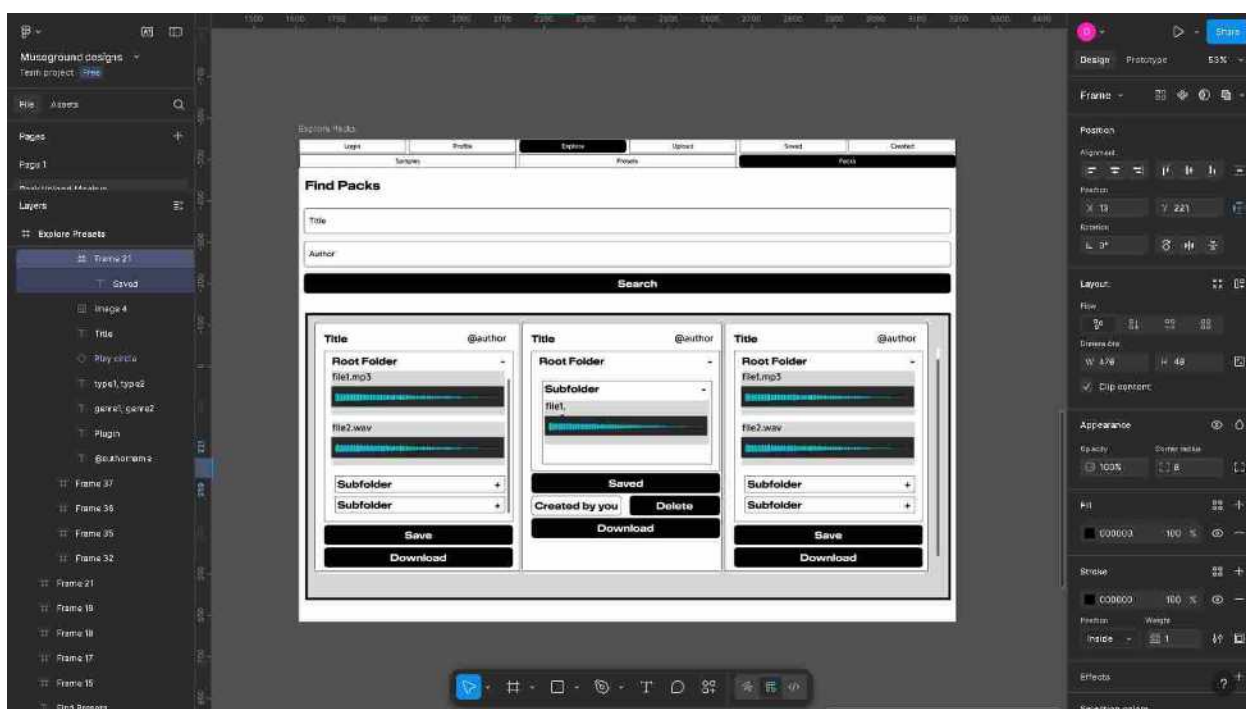


Рисунок 3.29 — Вайрфрейм сторінки explore packs

Далі, рисунки 3.30 та 3.31 зображують вайрфрейми сторінок upload sample та upload preset.

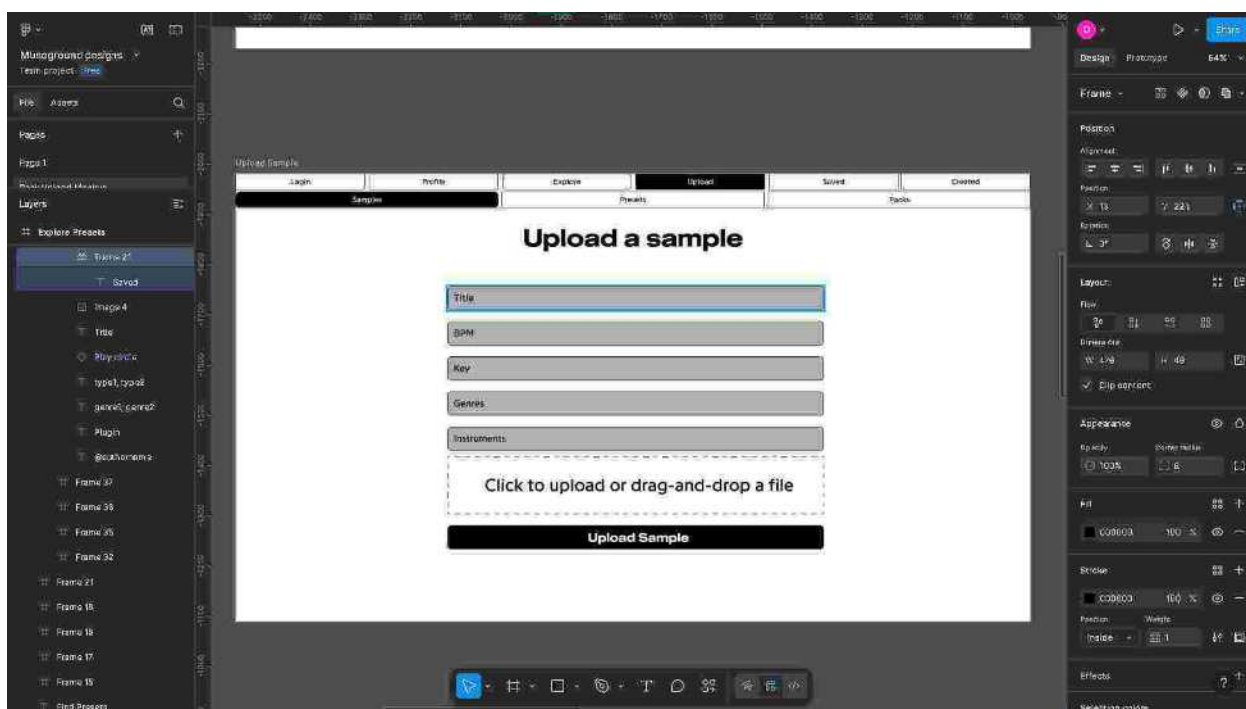


Рисунок 3.30 — Вайрфрейм сторінки upload sample

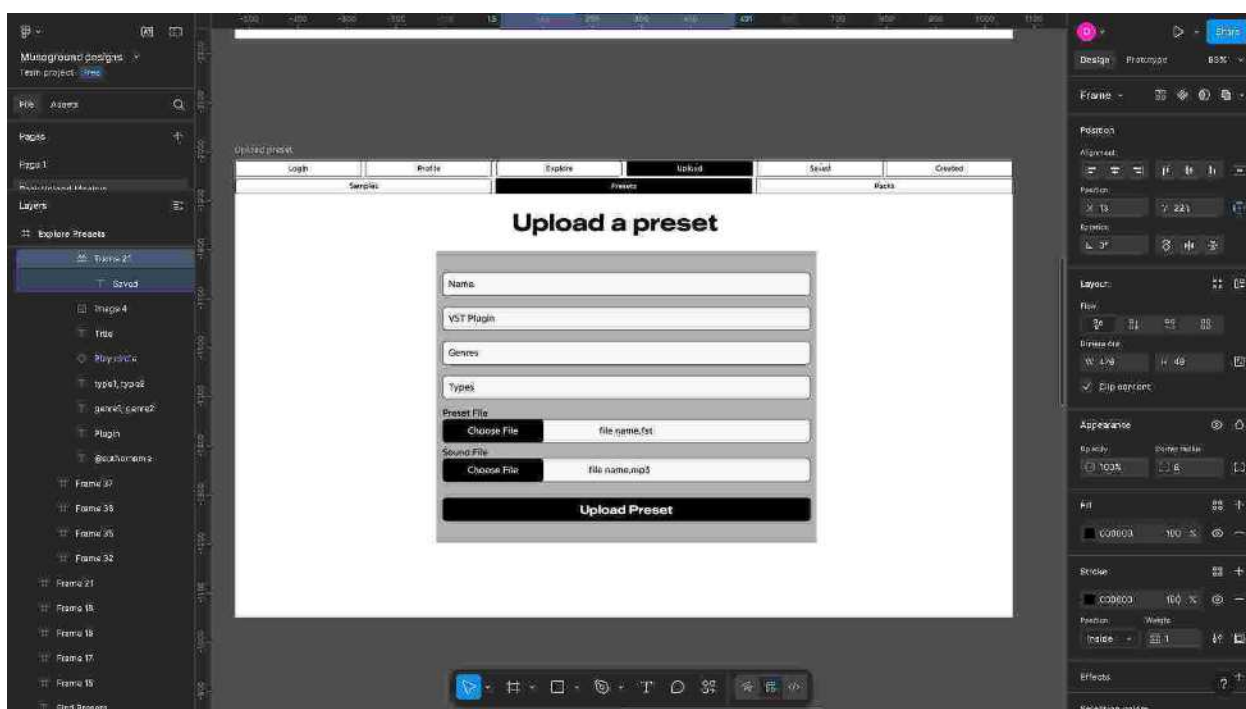


Рисунок 3.31 — Вайрфрейм сторінки upload preset

Далі, рисунки 3.32 та 3.33 зображують вайрфрейми сторінок upload rack та saved samples.

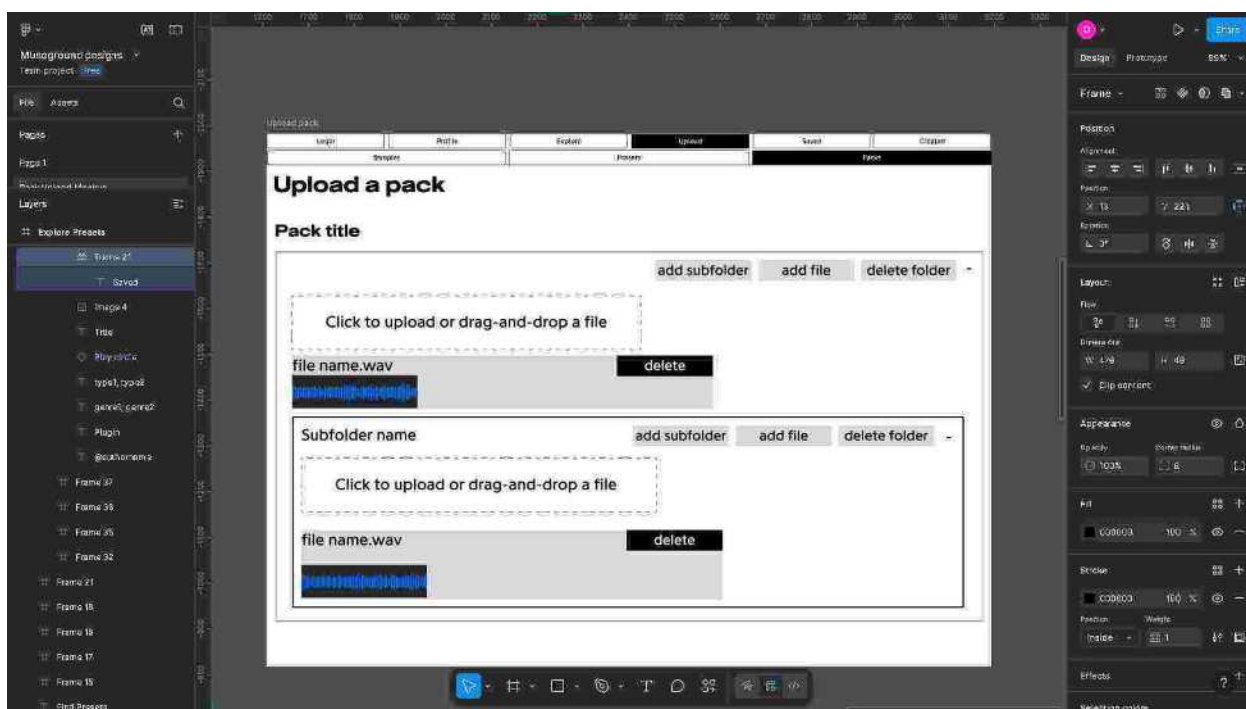


Рисунок 3.32 — Вайрфрейм сторінки upload pack

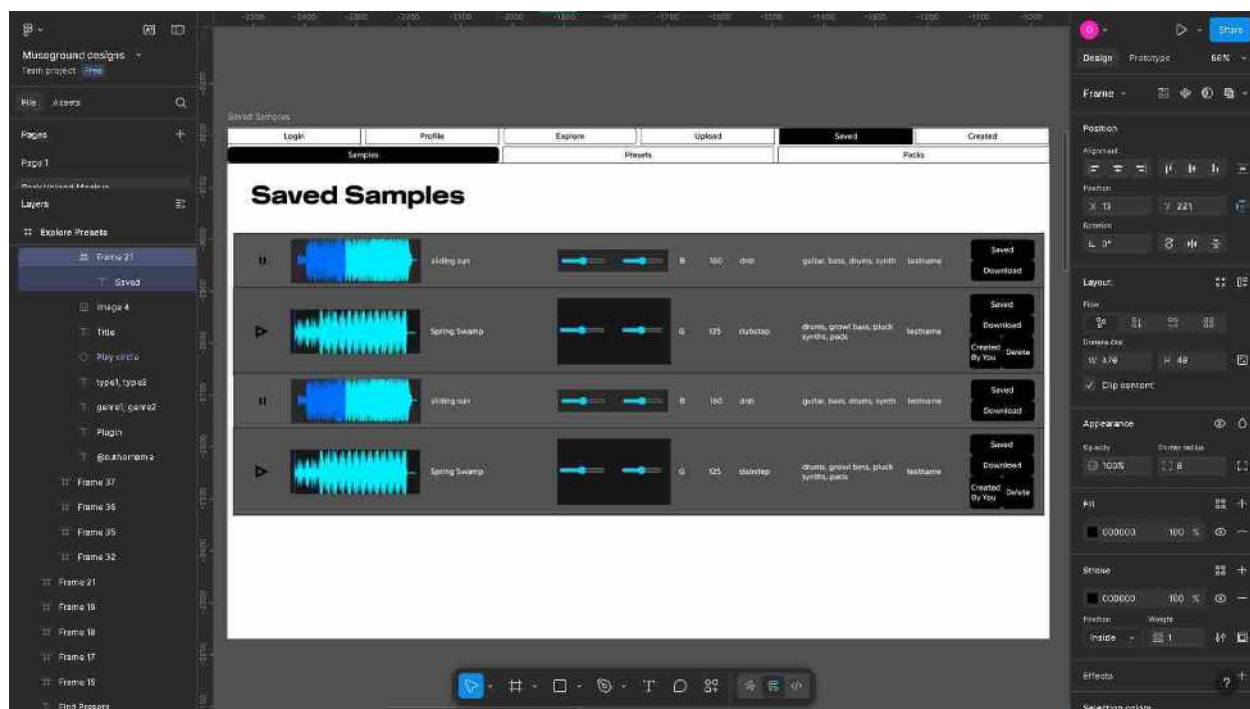


Рисунок 3.33 — Вайрфрейм сторінки saved samples

Далі, рисунки 3.34 та 3.35 зображують вайрфрейми сторінок saved presets та saved packs.



Рисунок 3.34 — Вайрфрейм сторінки saved presets

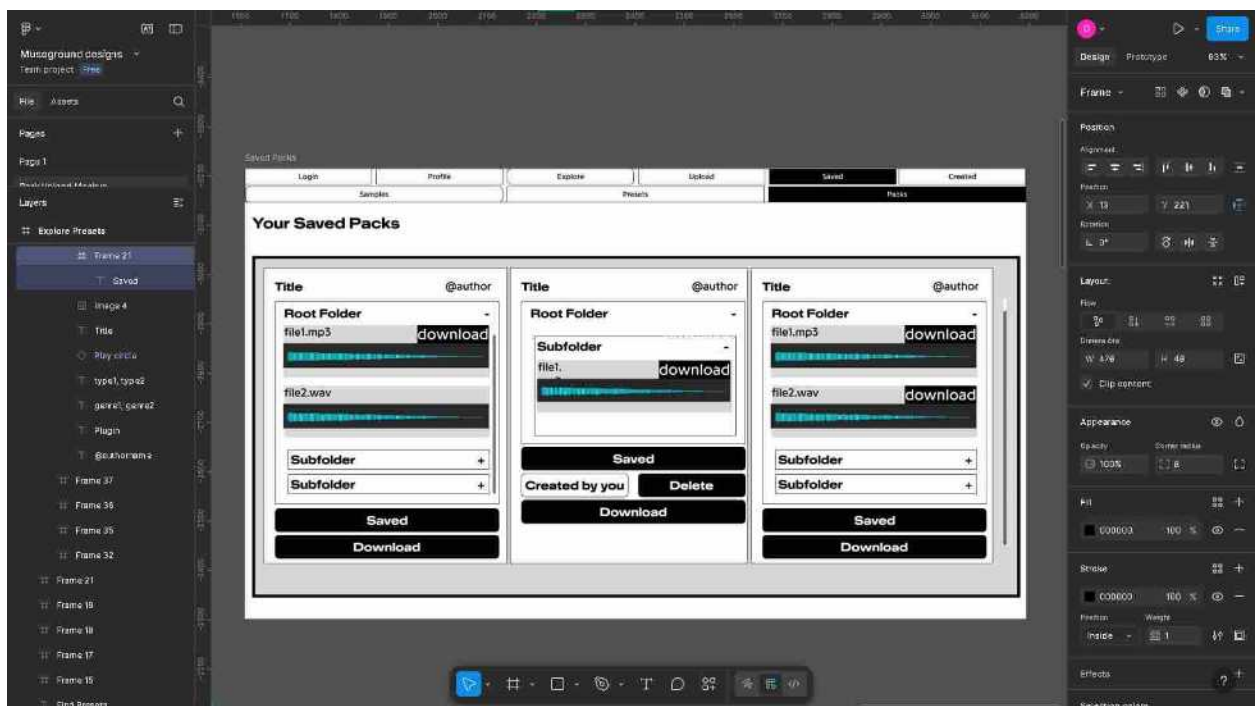


Рисунок 3.35 — Вайрфрейм сторінки saved packs

Далі, рисунки 3.36 та 3.37 зображують вайрфрейми сторінок created samples та created presets.

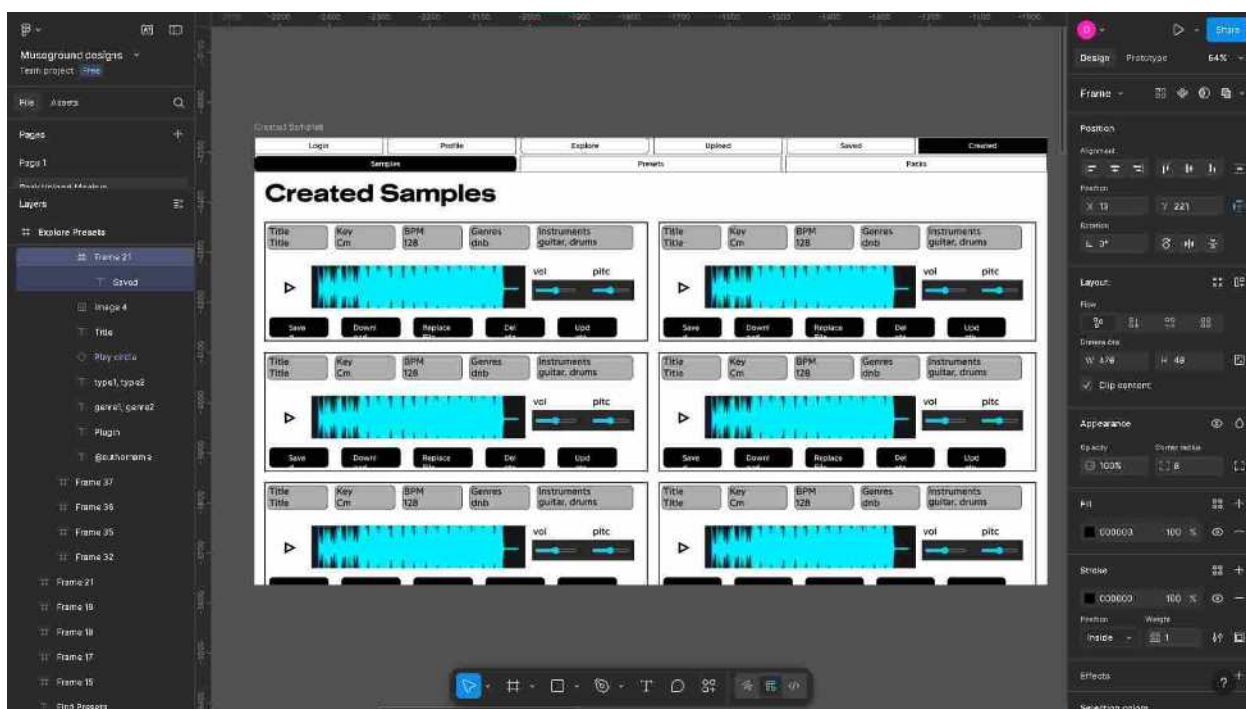


Рисунок 3.36 — Вайрфрейм сторінки created samples

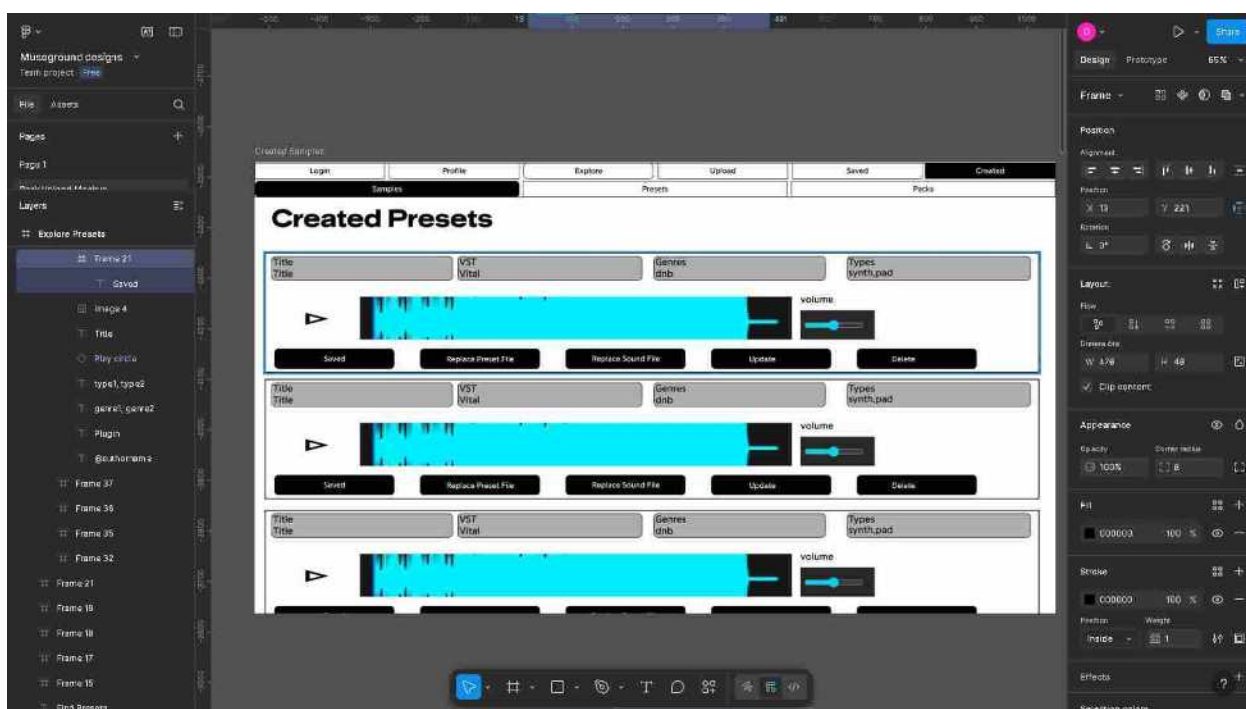


Рисунок 3.37 — Вайрфрейм сторінки created presets

Нарешті, рисунок 3.38 зображує вайрфрейм сторінки created racks.

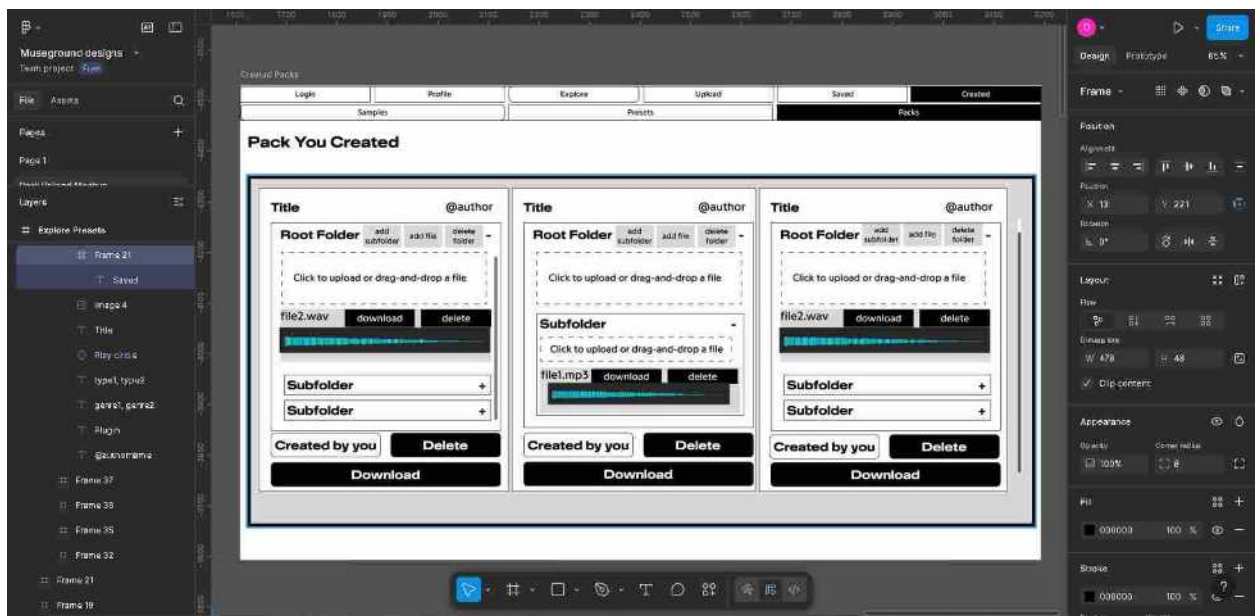


Рисунок 3.38 — Вайрфрейм сторінки created packs

3.5 Розробка серверної частини застосунку

3.5.1 Архітектура фреймворку Nest.js

Визначення цього фреймворку, процитоване з сайту офіційної документації, є наступним:

Nest (NestJS) [24] – це фреймворк для створення ефективних, масштабованих серверних застосунків Node.js. Він використовує прогресивний JavaScript, побудований за допомогою TypeScript та повністю його підтримує (але все ще дозволяє розробникам кодувати на чистому JavaScript) та поєднує елементи ООП (об'єктно-орієнтоване програмування),

FP (функціональне програмування) та FRP (функціонально-реактивне програмування). Nest пропонує готову архітектуру застосунків, яка дозволяє розробникам і командам створювати легко тестовані, масштабовані та легко підтримувані застосунки. Архітектура значною мірою натхненна Angular.»

Головними концепціями в Nest є три види класів: контролери, провайдери та модулі. Контролери – це класи, які відповідають за обробку вхідних http запитів до застосунку, та надсилання відповідей назад до клієнту. Для створення контролера у коді необхідно створити клас, до якого додати

декоратор `@Controller('path')`, де `path` стає частиною `url` шляху до ендпойнтів застосунку. Далі, всередині класу контролера розробник може створювати REST методи, використовуючи інші декоратори, такі як `@Get()`, `@Post`, `@Put`, `@Delete` та інші `http` методи. Також, всередині методів можна використовувати такі декоратори як `@Body()`, `@Res`, `@Req()` та інші.

Наступною концепцією розробки в Nest є провайдери, якими можуть бути багато типових класів, таких як сервіси, репозиторії, фабрики та інші. Ключова ідея провайдера полягає в тому, що його можна впроваджувати як залежність (`dependency injection`), що дозволяє об'єктам формувати різні зв'язки один з одним. Відповідальність за "підключення" цих об'єктів значною мірою бере на себе система рантайму Nest. Найчастіше, провайдерами в застосунках Nest стають саме класи сервіси. Основна ідея сервісів полягає в інкапсуляції логіки застосунку. Якщо контролери оброблюють `http` запити, то сервіси виконують більш комплексні задачі, такі як зберігання та отримання даних з бд та бізнес логіка.

Останньою концепцією з якої складається основа Nest є модулі. Модулі дозволяють поєднувати контролери, провайдери та інші модулі у класи, які добре репрезентують предметну область та взаємодію між сутностями в ній.

Модулі в Nest створюються декоратором `@Module()`, який приймає в якості аргументів 4 властивості (`properties`):

- `providers` - провайдери, екземпляри яких будуть створені за допомогою Nest-інжектора та які можуть бути спільними принаймні в цьому модулі;
- `controllers` - набір контролерів, визначених у цьому модулі, екземпляри яких мають бути створені;
- `imports` - список імпортованих модулів, які експортують провайдерів, необхідних у цьому модулі;
- `exports` - підмножина провайдерів даного модуля, які мають бути доступні в інших модулях, які імпортують цей модуль.

Модуль інкапсулює провайтери за замовчуванням, тобто можна впроваджувати лише провайтери, які є або частиною поточного модуля, або явно експортовані з інших імпортованих модулів.

3.5.2 Структура розробленого серверу

В контексті домену розроблюваного застосунку, сервер створений на основі Nest зображено на діаграмі компонентів наведених на рисунку 3.39. Також, на рисунку 3.40 зображено структуру файлів та папок у такому проєкті.

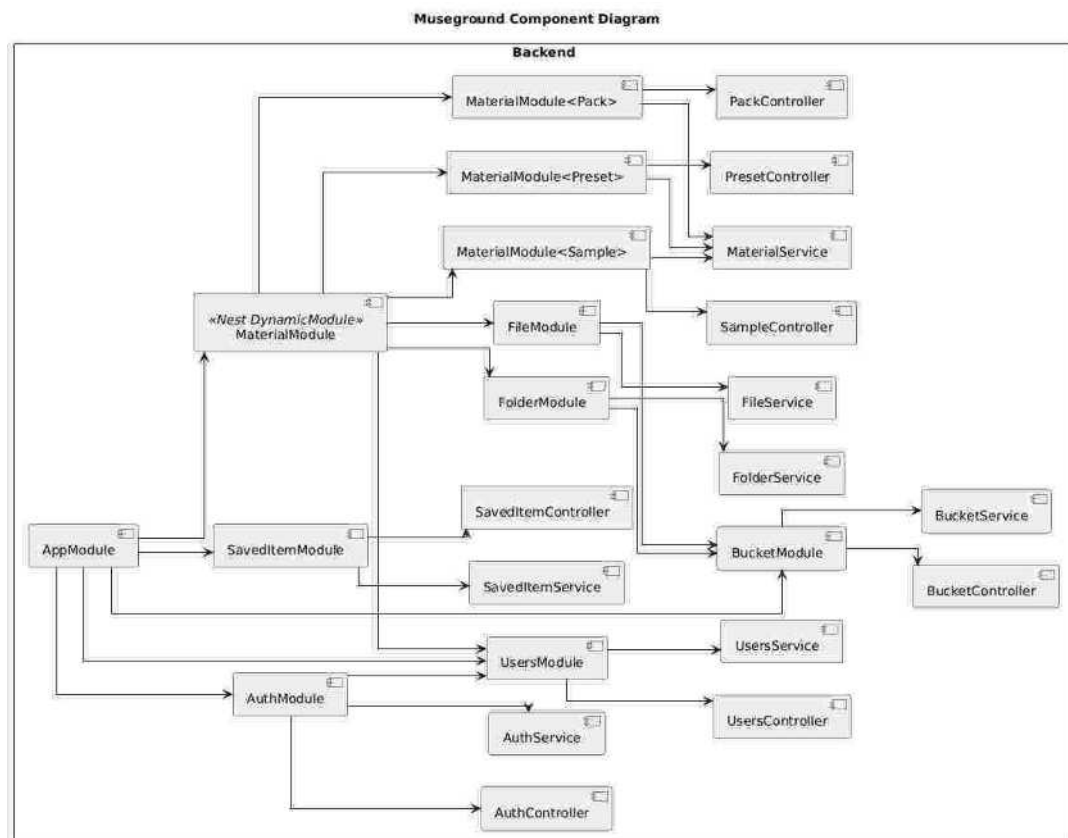


Рисунок 3.39 — Діаграма компонентів серверної частини застосунку



Рисунок 3.40 — Структура файлів проєкту

3.5.2.1 Модулі в розробленому сервері

Створеними в ході розробки застосунку Nest.js модулями є:

- AppModule, який є головним модулем серверної частини вебзастосунку та виконує роль центральної точки конфігурації всієї системи; саме в цьому модулі відбувається підключення до бази даних MongoDB, ініціалізація JWT авторизації, завантаження змінних середовища та об'єднання всіх функціональних підсистем застосунку; через AppModule реєструються модулі користувачів, авторизації, роботи з файлами, збережених матеріалів та динамічні модулі контенту для семплів, пресетів і паків;

- MaterialModule є універсальним динамічним модулем для роботи з музичними матеріалами платформи; його архітектура дозволяє

використовувати одну й ту саму логіку для різних видів ресурсів, тобто для семплів, пресетів та наборів; під час реєстрації модуль автоматично підключає відповідну схему Mongoose, створює сервіс для роботи з матеріалами та прив'язує потрібний контролер API; окрім цього, модуль інтегрується із файловою системою, системою папок та модулем користувачів, що дозволяє працювати як із метаданими контенту, так і з фізичними файлами; такий підхід значно зменшує дублювання коду та спрощує масштабування системи;

– FolderModule реалізує систему ієрархічних папок, яка використовується для організації музичних паків та структурування файлів усередині них. Модуль підтримує створення вкладених директорій, рекурсивний пошук дочірніх елементів, побудову деревовидної структури та видалення папок разом із їхнім вмістом; для цього використовується рекурсивний зв'язок між документами Folder через поле parent; модуль також синхронізує структуру папок із Cloudflare R2, забезпечуючи відповідність між файловою структурою бази даних і хмарного сховища;

– FileModule відповідає за управління файлами, які завантажуються користувачами вебзастосунку; модуль реалізує логіку збереження метаданих файлів у MongoDB та взаємодію з хмарним файловим сховищем Cloudflare R2; через цей модуль здійснюється завантаження, отримання, видалення та обробка аудіофайлів, що використовуються у семплах, пресетах та паках; для роботи модуль використовує схеми File і Folder, а також інтегрується з BucketModule, який забезпечує фізичне збереження файлів у хмарі;

– SavedItemModule реалізує систему збережених користувачем матеріалів; модуль дозволяє користувачам додавати семпли, пресети та паки до персонального списку збережених елементів, а також видаляти їх зі своєї колекції; для цього використовується універсальна схема SavedItem, яка через механізм refPath підтримує динамічні зв'язки з різними типами контенту; модуль також взаємодіє зі схемами користувачів і музичних матеріалів, забезпечуючи єдину систему роботи з персональними збереженнями;

– UsersModule відповідає за управління користувачами платформи та зберігання інформації про них; через цей модуль реалізується створення облікових записів, пошук користувачів, отримання профілів та взаємодія з іншими підсистемами застосунку; дані користувачів використовуються при створенні музичних матеріалів, системі авторизації та збережених елементів; модуль є фундаментальною частиною серверної архітектури, оскільки практично всі інші підсистеми так чи інакше залежать від інформації про користувача;

– AuthModule реалізує систему автентифікації та авторизації користувачів вебзастосунку; модуль підтримує локальну авторизацію через email і пароль, JWT автентифікацію та вхід через Google OAuth; через нього відбувається генерація та перевірка access token, управління сесіями користувачів і захист приватних маршрутів API; модуль інтегрується з UsersModule та JwtModule, забезпечуючи централізовану систему контролю доступу до ресурсів платформи;

API серверної частини вебзастосунку реалізовано у вигляді REST-архітектури на базі NestJS. Сервер забезпечує взаємодію клієнтського застосунку з базою даних MongoDB та хмарним файловим сховищем Cloudflare R2. Основними функціями API є автентифікація користувачів, управління музичними матеріалами, робота з файлами та папками, а також організація й завантаження паків.

3.5.2.2 Маршрути (API) розробленого серверу

API серверної частини вебзастосунку реалізовано у вигляді REST-архітектури на базі NestJS. Кожен маршрут API серверу реалізовано в окремому класі контролері, які є частинами модулів. Сервер забезпечує взаємодію клієнтського застосунку з базою даних MongoDB та хмарним файловим сховищем Cloudflare R2. Основними функціями API є автентифікація користувачів, управління музичними матеріалами, робота з файлами та

папками, а також організація й завантаження паків. Далі наведено списки та опис маршрутів кожного реалізованого в ході розробки контролера застосунку.

Контроллер `AppController` містить лише один маршрут `/get` який повертає тестове повідомлення сервера.

`UserController` забезпечує роботу з користувачами та керування збереженими матеріалами. Через нього виконуються реєстрація користувачів, збереження матеріалів у бібліотеку та отримання списків збережених об'єктів. Його маршрути:

- `POST /users/register` — реєстрація нового користувача;
- `POST /users/save-item` - збереження матеріалу користувачем;
- `POST /users/delete-saved` - видалення матеріалу зі збережених;
- `GET /users/saved-samples/:userId` - отримання збережених семплів користувача;
- `GET /users/saved-presets/:userId` - отримання збережених пресетів користувача;
- `GET /users/saved-packs/:userId` - отримання збережених паків користувача.

Контроллер `AuthController` зреалізує механізми автентифікації та авторизації користувачів. Він підтримує JWT-автентифікацію, Google OAuth та отримання даних авторизованого користувача. Його маршрутами є

- `POST /auth/login-jwt` - вхід користувача через JWT;
- `POST /auth/register-jwt` - реєстрація користувача через JWT;
- `GET /auth/profile` - отримання профілю авторизованого користувача;
- `GET /auth/google` - ініціалізація Google OAuth авторизації;
- `GET /auth/google/callback` - callback після Google OAuth авторизації.

`FolderController` відповідає за створення та базове керування папками у файловій структурі застосунку і має єдиний маршрут `POST /folders/create` для створення нової папки.

FileController забезпечує роботу з файлами та взаємодію з файловим сховищем та містить маршрути:

- POST /files/upload - завантаження файлу;
- POST /files/download - завантаження файлу користувачем;
- POST /files/delete - видалення файлу.

BucketController забезпечує низькорівневу взаємодію з хмарним сховищем R2/S3, включаючи завантаження файлів, доступ до них та генерацію тимчасових посилань та містить маршрути:

- POST /bucket/upload - завантаження файлу у bucket;
- GET /bucket/:key/*path - отримання файлу зі сховища;
- GET /bucket/signed-url/:fileKey - генерація signed URL для файлу.

SampleController керує семплами та реалізує їх завантаження, пошук, редагування, видалення та завантаження користувачами. Його маршрути:

- POST /samples/upload - завантаження нового семплу;
- GET /samples/get/:id - отримання семплу за ID;
- GET /samples/filter/query - пошук і фільтрація семплів;
- GET /samples/download/:id - завантаження семплу;
- DELETE /samples/delete/:id - видалення семплу;
- GET /samples/created-by-user/:userId - отримання семплів користувача;
- PATCH /samples/update/:sampleId - оновлення даних семплу;
- POST /samples/replace-file - заміна аудіофайлу семплу.

PresetController відповідає за роботу з пресетами, включаючи завантаження пресетів і демо-аудіо, пошук, оновлення та видалення. Його маршрути:

- POST /presets/upload - завантаження нового пресету;
- GET /presets - пошук і фільтрація пресетів;
- GET /presets/get/:id - отримання пресету за ID;
- GET /presets/download/:id - завантаження файлу пресету;
- DELETE /presets/delete/:id - видалення пресету;

- GET /presets/created-by-user/:userId - отримання пресетів користувача;
- PATCH /presets/update/:presetId - оновлення даних пресету;
- POST /presets/replace-preset-file - заміна файлів пресету.

PackController реалізує керування паками, файловою структурою всередині паків та операціями над вкладеними файлами й папками. Його маршрутами є:

- POST /packs/create - створення нового набору;
- POST /packs/add-folder - додавання папки до набору;
- POST /packs/add-file - додавання файлу до набору;
- GET /packs/user-created-packs-full/:userId - отримання повної структури наборів користувача;
- DELETE /packs/delete-folder - видалення папки;
- DELETE /packs/delete-file - видалення файлу;
- GET /packs/:id - отримання набору за ID;
- POST /packs/move-item - переміщення файлу або папки;
- GET /packs/find/query - пошук наборів;
- GET /packs/file-urls/:id - отримання файлів набору;
- DELETE /packs/delete-pack - видалення набору;
- GET /packs/download-file/:fileId - завантаження окремого файлу
- GET /packs/download-folder/:folderId - завантаження папки у ZIP-архіві.

3.6 Розробка клієнтської частини застосунку

3.6.1 Архітектура фреймворку Next.js

Next.js це React-фреймворк для створення фулстек веб-застосунків, розробка з яким передбачає використання react компонентів для створення користувацьких інтерфейсів, а Next — для додаткових функцій та оптимізацій. Він також автоматично налаштовує інструменти нижчого рівня, такі як пакетери та компілятори.

Головною ідеєю Next.js є поєднання React-компонентів із можливостями серверного рендерингу, що дозволяє створювати сучасні швидкодіючі вебзастосунки з високою оптимізацією для пошукових систем.

Основні концепції Next:

- компоненти (Components);
- сторінки (Pages);
- маршрутизація (Routing);
- API маршрути (API Routes);
- отримання даних (Data Fetching);
- Layout та Client/Server Components.

Компоненти у Next - це базові будівельні блоки інтерфейсу користувача. Вони створюються за допомогою React і описують окремі елементи або частини сторінок, наприклад кнопки, картки або меню.

Next.js підтримує як клієнтські компоненти, що взаємодіють із користувачем безпосередньо у браузері, так і серверні компоненти, які рендеряться на сервері для підвищення швидкодії.

Маршрутизація у Next побудована на основі файлової системи. Кожен файл, розташований у директорії /app (у нових версіях) або /pages (у старіших), автоматично стає маршрутом. Наприклад, файл `app/blog/page.tsx` відповідає маршруту `/blog`.

Next.js пропонує декілька способів отримання даних, зокрема:

- Server-side Rendering (SSR) - отримання даних під час кожного запиту (`getServerSideProps`);
- Static Site Generation (SSG) - попереднє створення сторінок під час збірки (`getStaticProps`);
- Incremental Static Regeneration (ISR) - оновлення статичних сторінок без повної перебудови;
- у новій архітектурі (Next 13+) - через `async` серверні компоненти або `fetch()` всередині них.

3.6.2 Розроблений клієнт застосунку

Структура файлів проєкту Next, яка реалізує усі необхідні для розроблюваного застосунку сторінки наведена на рисунку 3.41, а на рисунку 3.42 зображена діаграма компонентів яка візуалізує взаємодію компонентів та сторінок цього проєкту. Далі на рисунках 3.43 — 3.57 наведено скріншоти вже розроблених сторінок застосунку. Окрім засобів React та Next.js, для стилів сторінок було використану бібліотеку Tailwind CSS.

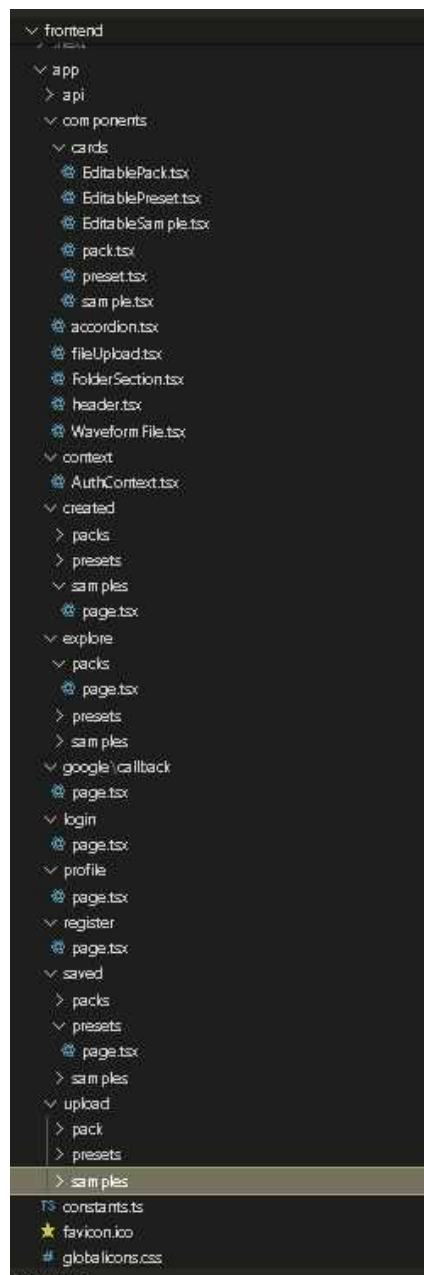


Рисунок 3.41 — Структура файлів клієнта застосунку

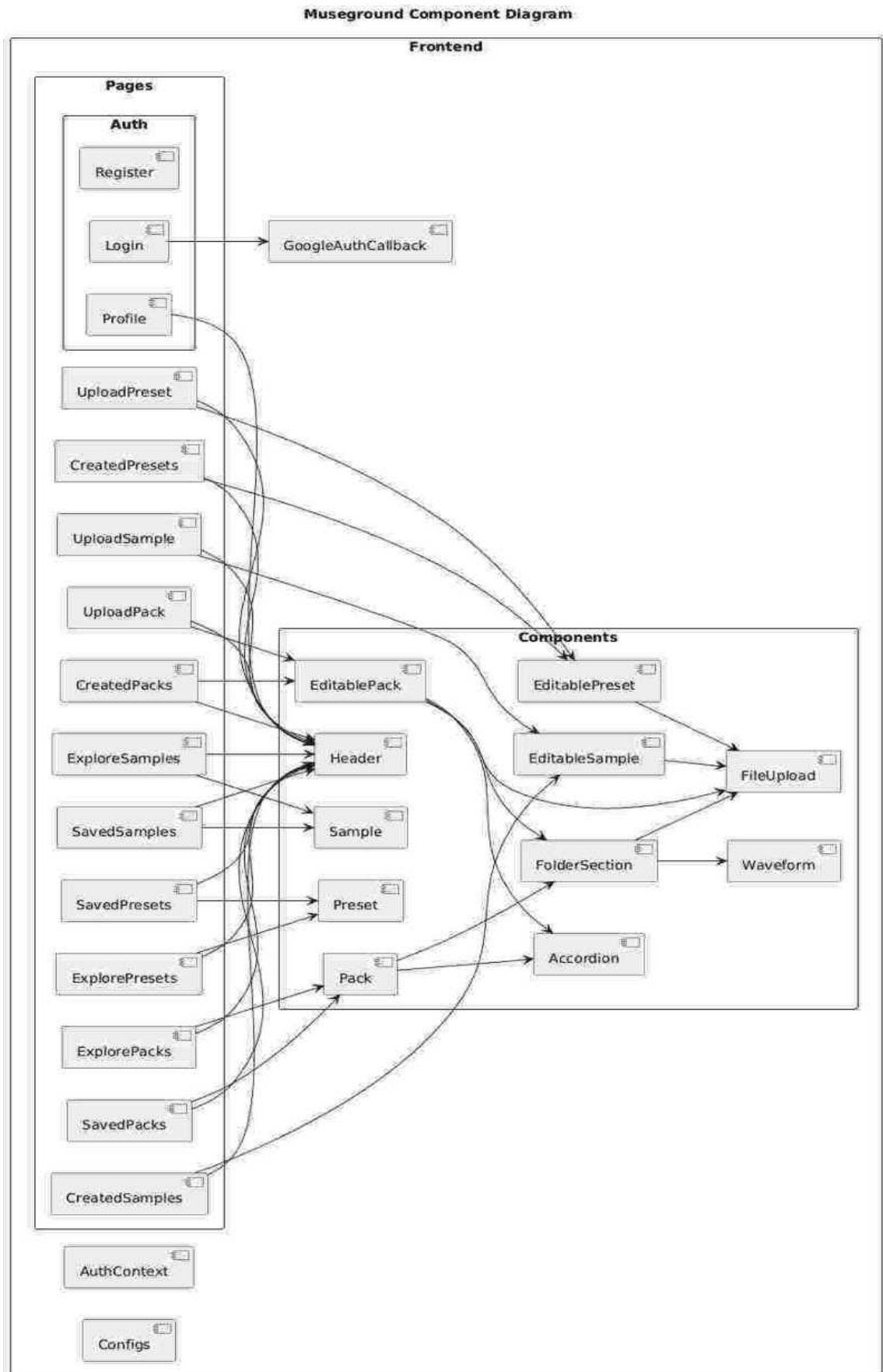


Рисунок 3.42 — Діаграма компонентів клієнтської частини застосунку

В ході розробки для кожного з трьох видів матеріалів було створено однаковий набір компонентів та сторінок:

- <MaterialType> - react компоненти які візуалізують кожен з видів ресурсів;
- Editable<MaterialType> - компоненти які візуалізують ресурси під час їх редагування користувачем, який є їх автором;
- Explore<MaterialType> - сторінки пошуку, фільтрування, перегляду та зберігання типу ресурсів;
- Saved<MaterialType> - сторінки зі списком збережених користувачем ресурсів певного типу;
- Upload<MaterialType> - сторінки для додавання до застосунку нового ресурсу певного типу користувачем;
- Created<MaterialType> - списки доступних для редагування матеріалів, створених користувачем;
- Login – сторінка логіну
- Register – сторінка реєстрації
- Accordion – компонент який інкапсулює інтерфейс списку, який може бути закритим та відкритим;
- FileUpload – компонент який інкапсулює площу для додавання або перетягування файлів;
- FolderSection – компонент який відображає одну папку та її вміст;
- Header – компонент меню навігації сайту;
- WaveformFile – компонент відображення звукової хвилі;

Рисунок 3.43 зображує розроблену сторінку логіну, рисунок 3.44 — сторінку реєстрації та рисунок 3.45 — сторінку профілю.

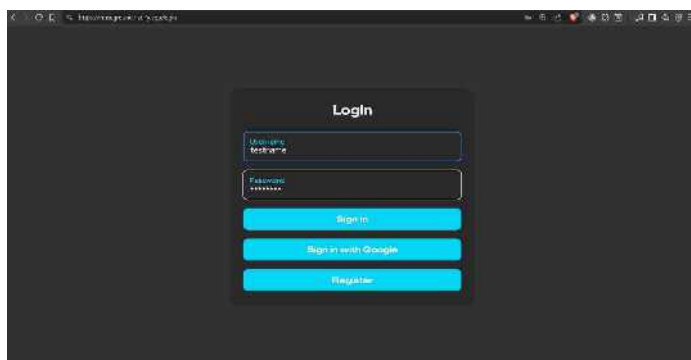


Рисунок 3.43 — Сторінка логіну

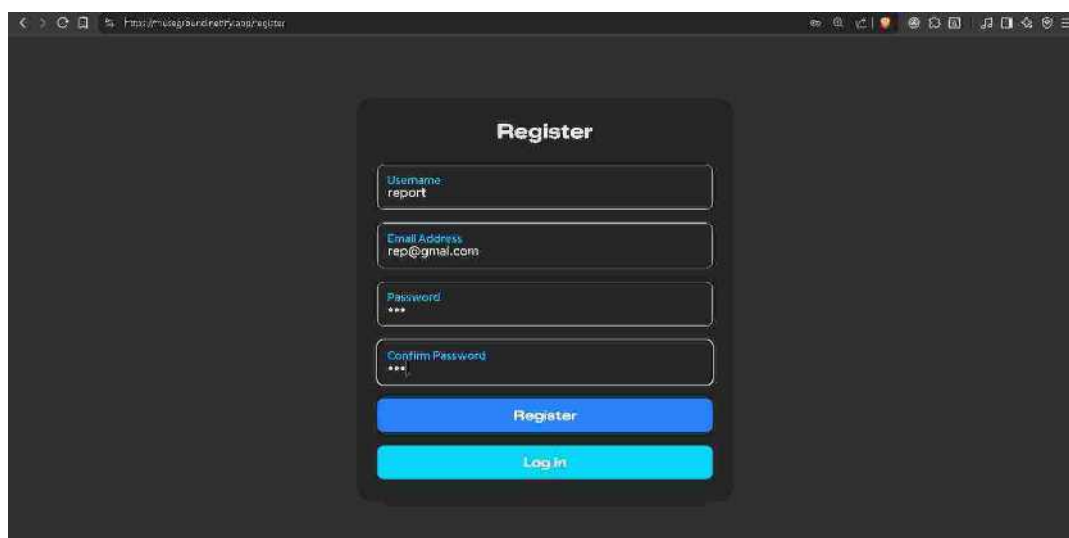


Рисунок 3.44 — реєстрація

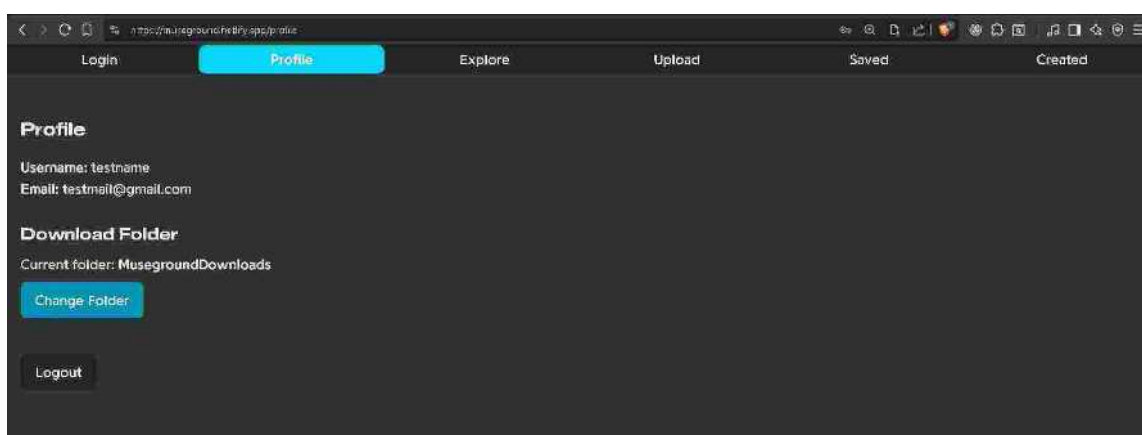


Рисунок 3.45 — Сторінка профілю

Далі, рисунки 3.46 та 3.47 зображують інтерфейс розроблених сторінок explore samples та explore presets.

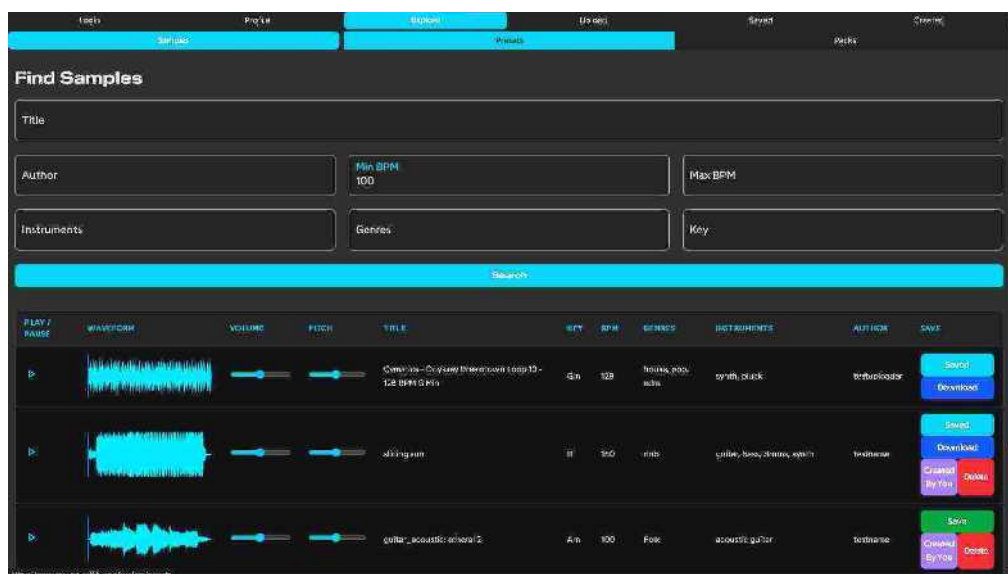


Рисунок 3.46 — Сторінка explore samples

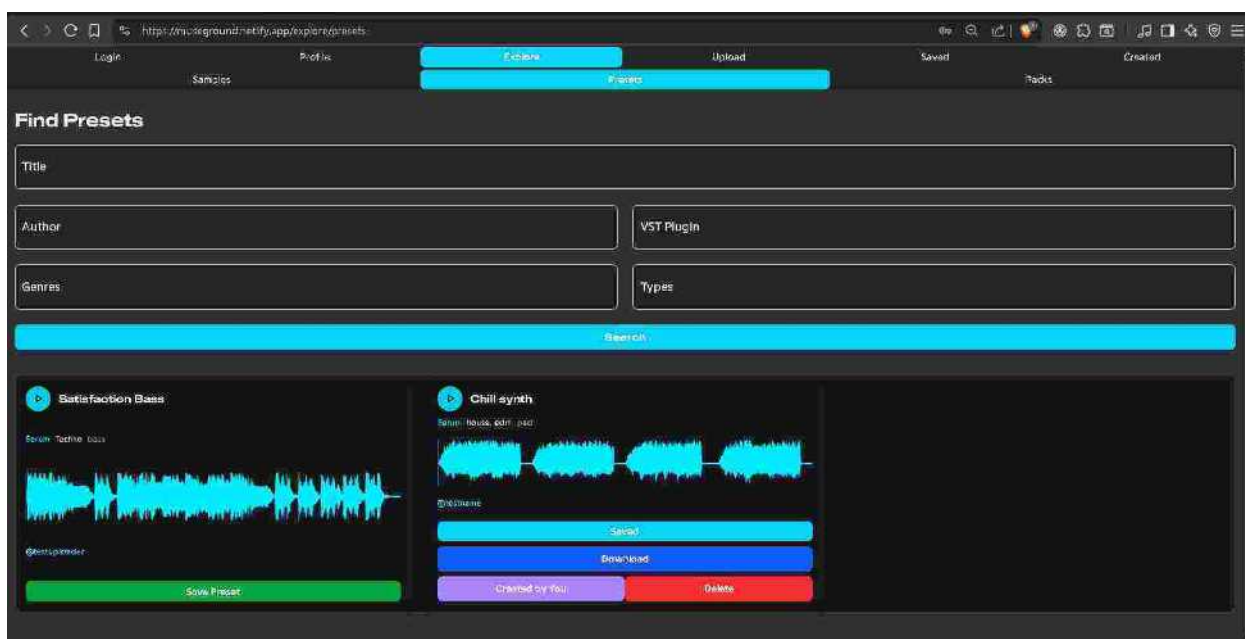


Рисунок 3.47 — Сторінка explore presets

Далі, рисунки 3.48 та 3.49 зображують інтерфейс розроблених сторінок explore packs та upload sample.

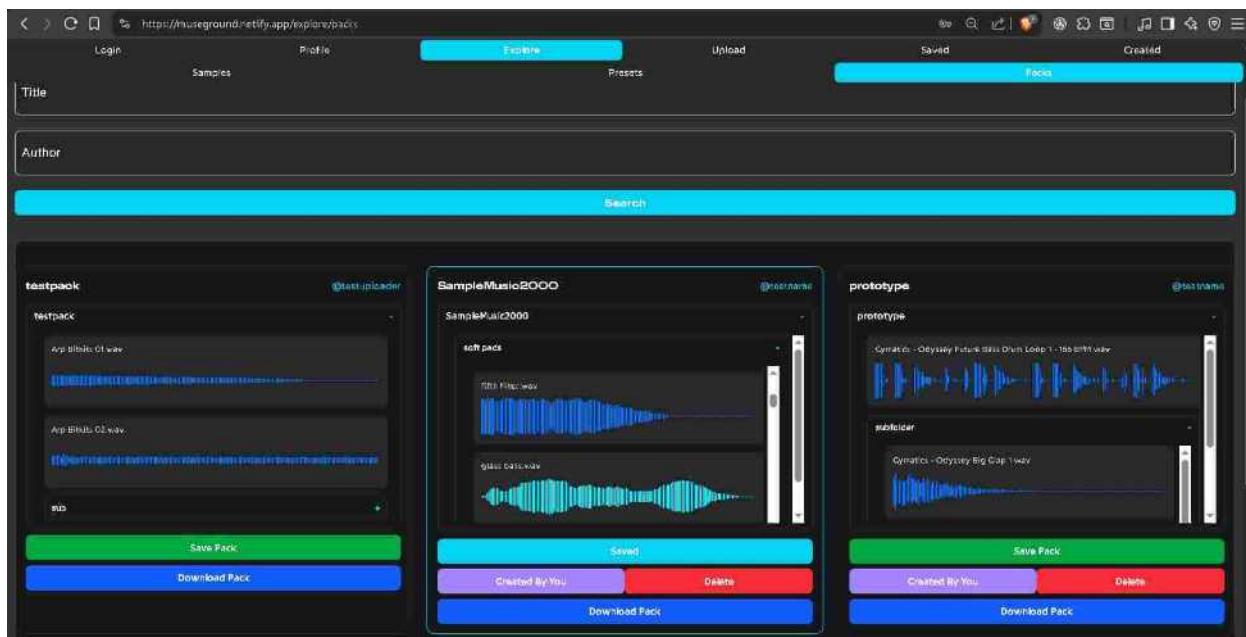


Рисунок 3.48 — Сторінка explore packs

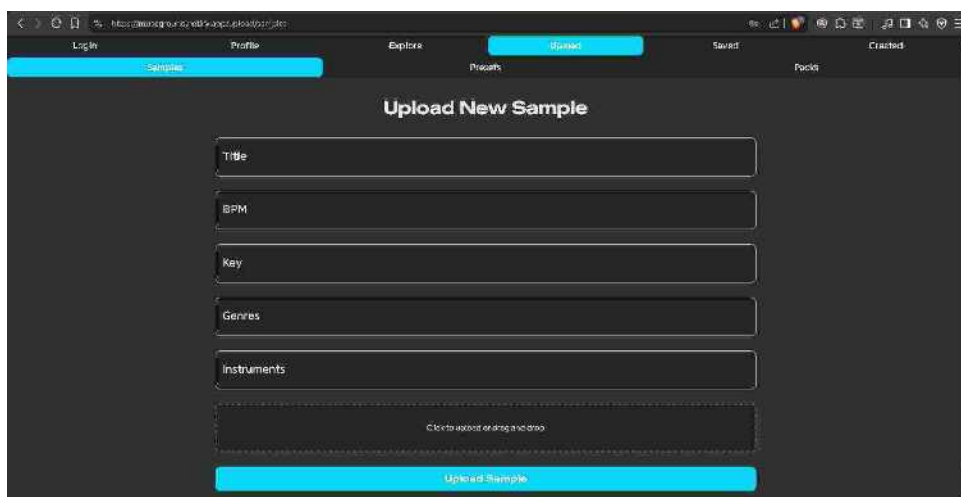


Рисунок 3.49 — Сторінка upload sample

Далі, рисунок 3.50 зображує сторінку upload preset, рисунок 3.51 — upload pack та рисунок 3.52 — saved samples.

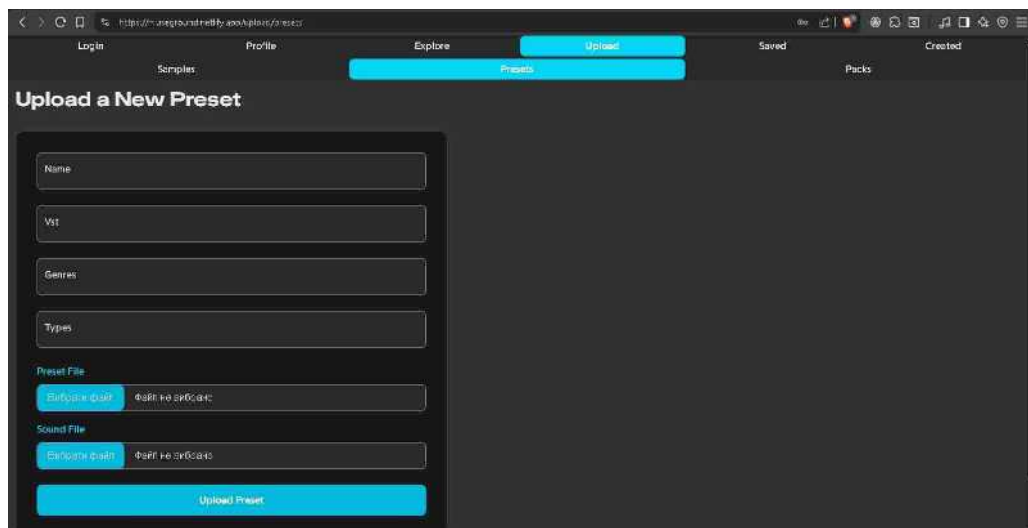


Рисунок 3.50 — Сторінка upload preset

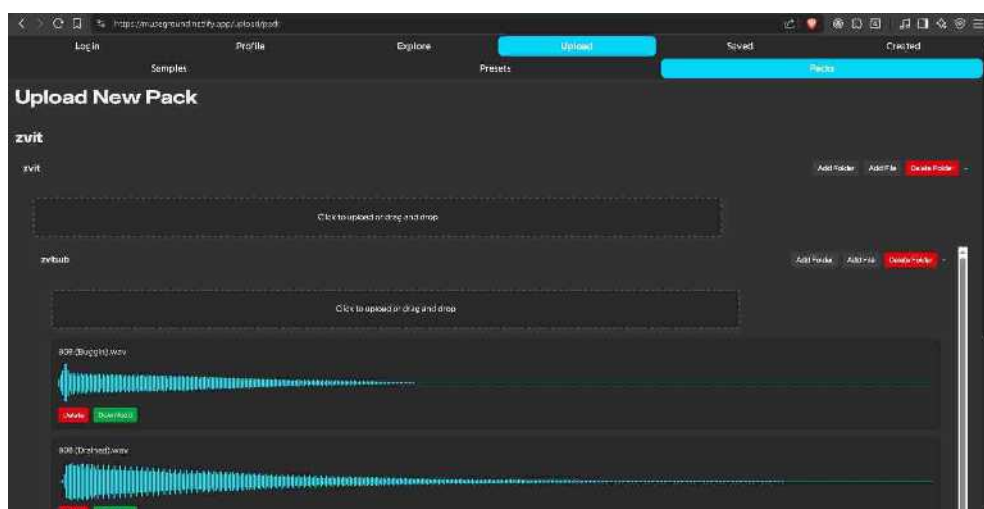


Рисунок 3.51 — Сторінка upload pack

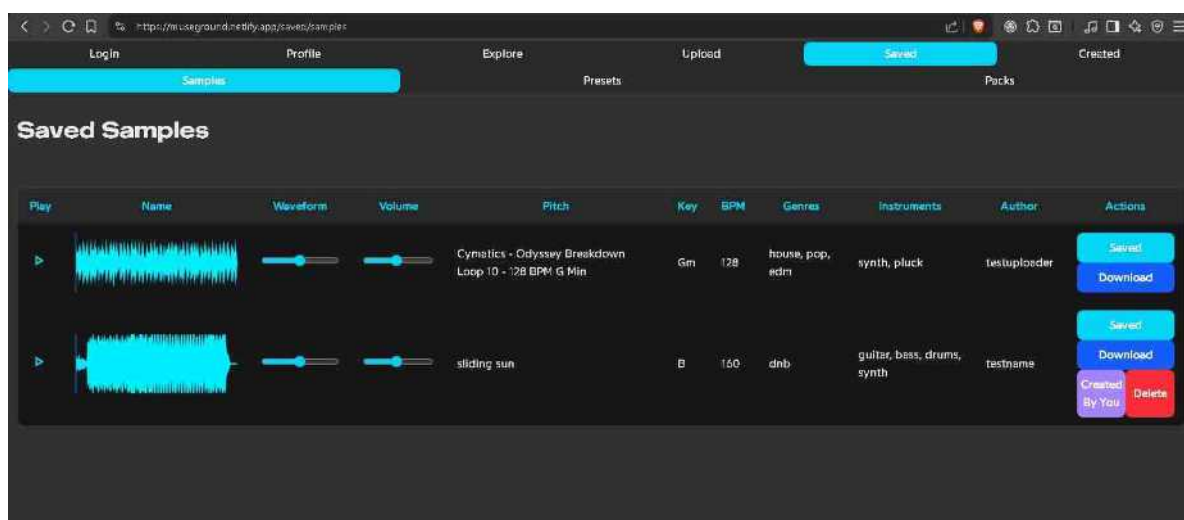


Рисунок 3.52 — Сторінка saved samples

Далі, рисунок 3.53 зображує розроблений інтерфейс сторінки saved presets, рисунок 3.54 — сторінки saved packs, та рисунок 3.55 — сторінки created samples.

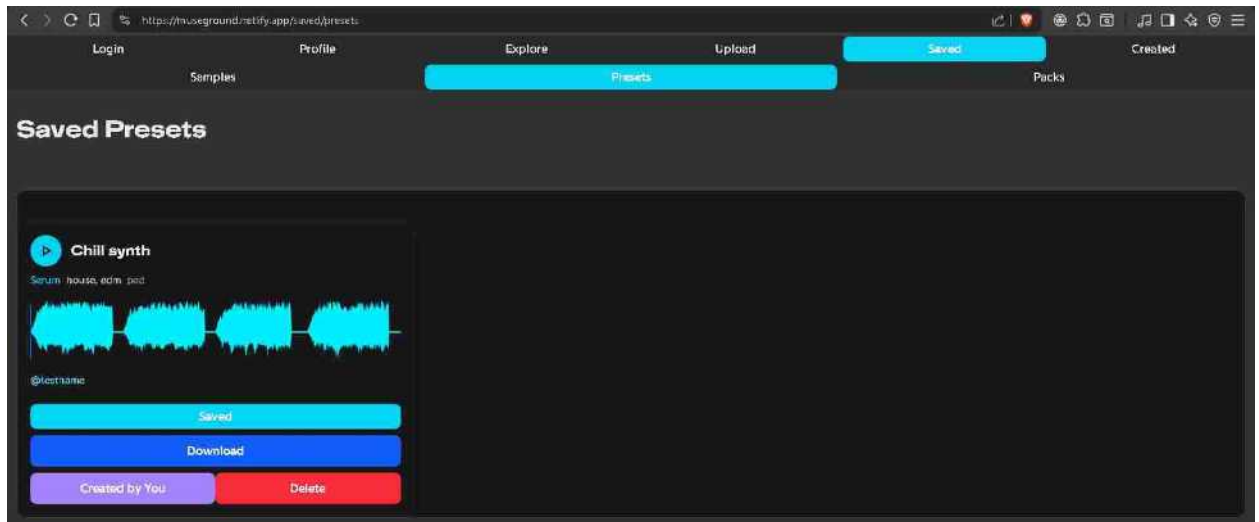


Рисунок 3.53 – Сторінка saved presets

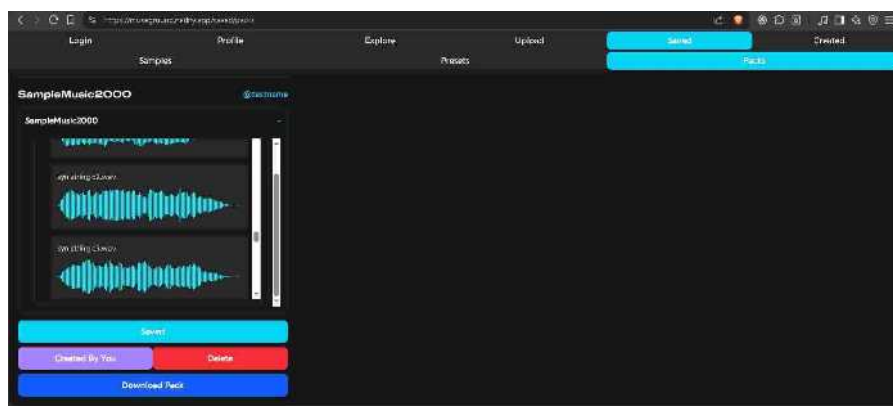


Рисунок 3.54 — Сторінка saved packs



Рисунок 3.55 — Сторінка created samples

Нарешті, рисунки 3.56 та 3.57 зображують інтерфейс розроблених сторінок created presets та created packs.



Рисунок 3.56 — Сторінка created presets

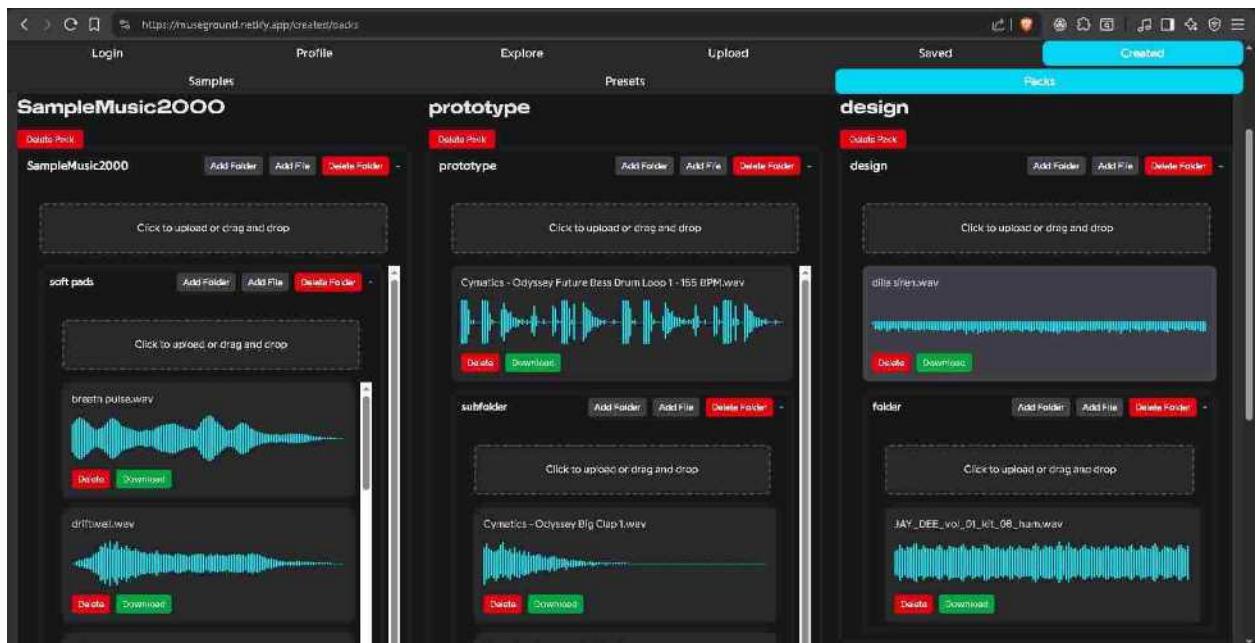


Рисунок 3.57 — Сторінка created packs

3.7 Висновки за розділом 3

У межах розділу 3 було виконано проєктування та розробку вебзастосунку для роботи з цифровими музичними ресурсами. У процесі роботи було сформовано архітектуру системи, спроектовано структуру бази

даних, реалізовано серверну та клієнтську частини застосунку, а також створено графічний користувацький інтерфейс.

Під час проєктування архітектури системи було побудовано UML-діаграми класів, компонентів, прецедентів, діяльності та послідовностей, що дозволило формалізувати структуру застосунку та взаємодію між його основними підсистемами. Було визначено ключові сутності системи, а саме користувачів, семпли, пресети, набори, файли, папки та збережені елементи, а також описано процеси взаємодії між клієнтом, сервером, базою даних і хмарним файловим сховищем Cloudflare R2.

У ході розробки алгоритмів роботи застосунку було описано основні сценарії роботи користувачів із музичними матеріалами, зокрема процеси автентифікації, пошуку, фільтрації, завантаження, редагування, видалення та збереження ресурсів. Для демонстрації послідовності виконання операцій було створено UML-діаграми послідовностей, що відображають взаємодію компонентів системи під час виконання основних функцій застосунку.

Також було спроектовано структуру бази даних MongoDB Atlas та створено схеми сутностей User, Sample, Preset, Pack, File, Folder і SavedItem. Розроблена структура забезпечує гнучке зберігання різноманітних музичних ресурсів, підтримку файлової ієрархії наборів, а також реалізацію системи персональних збережень користувачів.

У межах проєктування графічного інтерфейсу було створено вайрфрейми, макети та інтерактивний прототип сторінок застосунку за допомогою Figma та ProtoPie. Було визначено структуру інтерфейсу для сторінок перегляду, пошуку, завантаження та редагування музичних матеріалів, а також сформовано вимоги до стилізації, навігації та поведінки елементів інтерфейсу.

Для серверної частини застосунку було використано фреймворк NestJS. У процесі розробки реалізовано модульну архітектуру серверу, яка включає модулі користувачів, авторизації, роботи з файлами, папками, збереженими матеріалами та музичними ресурсами. Також було реалізовано REST API для

роботи із семплами, пресетами, наборами, файлами та системою автентифікації користувачів.

Клієнтську частину застосунку було реалізовано за допомогою Next.js та React. У ході розробки створено сторінки для перегляду, пошуку, завантаження, збереження та редагування музичних матеріалів, а також набір багаторазових React-компонентів для побудови інтерфейсу. Розроблена структура клієнтського застосунку забезпечує зручну навігацію, повторне використання компонентів та підтримку масштабування системи в майбутньому.

Отже, у результаті виконання розділу було спроектовано та реалізовано основні компоненти вебзастосунку для роботи з цифровими музичними ресурсами, сформовано архітектурну основу системи та реалізовано функціонал, необхідний для подальшого тестування й експлуатації застосунку.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

Цей розділ є важливою частиною роботи, оскільки він розкриває етапи, пов'язані з впровадженням та використанням розробленого вебзастосунку. У цьому розділі буде здійснено опис роботи, наведено інструкції з використання та проведено тестування вебзастосунку

4.1 Призначення та умови застосування застосунку

Призначенням розробленого застосунку «Museground» є створення онлайн платформи де спільнота користувачів зацікавлена в створенні музики зможе шукати, отримувати та обмінюватись семплами, їх наборами та пресетами для VST-плагінів.

Для доступу до розробленого сайту користувачу достатньо перейти за посилання: <https://museground.netlify.app/login>.

Для розробки ж, розробнику необхідно клонувати репозиторій з платформи github, і далі в терміналі запустити команди `npm install` окремо для директорій `backend` та `frontend`, отримати змінні оточення та запустити `npm run start:dev` на сервері та `npm run dev` на клієнті.

4.2 Опис роботи з програмою

4.2.1 Вхідні та вихідні дані застосунку

Вхідні дані вебзастосунку є різними в залежності від сторінки на якій знаходиться користувач.

На сторінках логіну або реєстрації користувач вводить інформацію про адресу електронної пошти, пароль та ім'я, та в якості результату отримує або перенаправлення на сторінку `/explore/samples` або вивід повідомлення про помилку.

На сторінках типу /explore вхідними даними є дані введені в поля фільтрів пошуку ресурсів, такі як назва, жанри, типи звуку та інші, в залежності від форми даних того чи іншого виду ресурсу. Вихідними даними на цих сторінках є список ресурсів певного виду, де елементи списку відповідають заданим користувачем параметрам пошуку.

На сторінках типу /upload вхідними даними є дані які характеризують ресурс який завантажується, файл(и) з яких складається ресурс та його назву. В разі додавання наборів окрім файлів також можна створювати папки всередині набору. Вихідними даними є повідомлення про помилку або перенаправлення на сторінку /created відповідного ресурсу.

На сторінках типу /created вхідними даними є будь-які дані які користувач змінює стосовно створених ним ресурсів, і вихідними даними є оновлений стан цих ресурсів після редагування.

На сторінках типу /saved вхідними даними є рішення користувача про збереження або видалення зі «збережених» ресурсів того чи іншого виду, і вихідними даними є список ресурсів які користувач додав до «збережених».

4.2.2 Сценарій використання застосунку

При вході на сайт застосунку користувач потрапляє на сторінку логіну, де в разі наявності облікового запису він може ввести облікові дані та увійти в систему. У разі відсутності облікового запису користувач може або перейти на сторінку реєстрації, де після введення імені, пароллю та адреси електронної пошти зможе пройти логін успішно, або використати сторонній провайдер аутентифікації Google.

Після проходження логіну користувач завжди потрапляє на сторінку /explore/samples. Тут йому доступний вільний вибір подальших дій. За замовчуванням ця сторінка демонструє усі наявні на сайті семпли. Кожен з них користувач може зберегти, щоб потім завантажити з розділу «збережене». Також на цій сторінці користувач може ввести значення у поля характеристик

семплів, такі як інструменти, жанри, темп і.т.д, і шукати семпли за цими параметрами. Знайдені семпли також можна сортувати за будь-яким з параметрів, а також відтворювати їх аудіо файли всередині сторінки, змінюючи їх гучність та висоту. Якщо якийсь семпл вже є збереженим, користувач може скасувати це. Якщо семпл був створений користувачем, він зможе його видалити.

Схожий функціонал доступний користувачу на сторінках `/explore/presets` та `/explore/packs`, з урахуванням їх особливостей, таких як вкладена структура папок в наборах.

Іншим видом сторінок доступних користувачу є сторінки `/upload/<вид_ресурсу>`. На них користувач може вводити характеристики ресурсу, додавати файли до них та додавати до сайту в цілому.

На сторінках `/created/<вид_ресурсу>` користувачу надається весь необхідний інтерфейс для редагування ресурсів доданих ним самим до сайту.

На сторінках `/saved/<вид_ресурсу>` користувачу доступний список збережених на інших сторінках ресурсів, та додається можливість локально завантажити їх на комп'ютер користувача.

4.3 Автоматизоване тестування розробленого застосунку

4.3.1 Тестування серверної частини застосунку

Для перевірки коректності роботи серверної частини вебзастосунку «Museground» було проведено автоматизоване тестування за допомогою фреймворку Jest [25], який є стандартним інструментом тестування для застосунків на базі NestJS. Загалом було реалізовано та виконано 10 тестових наборів (test suites), які охоплюють як модульне тестування окремих сервісів і контролерів, так і інтеграційне (e2e) тестування REST API серверу.

У ході виконання тестів було успішно пройдено 34 тестові сценарії без помилок. Результати запуску тестування наведено на рисунку 4.1.

```

PS C:\Documents\Museground-Revised\backend> npm run test

> backend@0.0.1 test
> jest

PASS src/test/file.service.spec.ts (11.698 s)
PASS src/test/bucket.service.spec.ts
PASS src/test/pack.controller.e2e.spec.ts (16.425 s)
PASS src/test/preset.controller.e2e.spec.ts (16.553 s)
PASS src/test/users.service.spec.ts
PASS src/test/folder.service.spec.ts
PASS src/test/auth.service.spec.ts
PASS src/test/auth.controller.spec.ts
PASS src/test/app.controller.spec.ts
PASS src/test/sample.controller.e2e.spec.ts (9.862 s)
A worker process has failed to exit gracefully and has been force exited. This is likely caused by tests leaking due to improper teardown. Try running with --detectOpenHandles to find leaks. Active timers can also cause this, ensure that .unref() was called on them.

Test Suites: 10 passed, 10 total
Tests:       34 passed, 34 total
Snapshots:   0 total
Time:        25.428 s
Ran all test suites.
PS C:\Documents\Museground-Revised\backend>

```

Рисунок 4.1 — Результат запуску автоматизованих тестів для серверу

Для тестування функціональності, пов'язаної з базою даних MongoDB, застосовувався пакет `mongodb-memory-server`, який дозволяє створювати тимчасовий екземпляр MongoDB у пам'яті. Такий підхід забезпечує ізолюваність тестів, відсутність впливу на реальну базу даних та високу швидкість виконання тестування.

Тестування серверної частини охоплювало основні модулі та сервіси застосунку, а також REST API контролери.

Було реалізовано модульні тести для базових компонентів застосунку. Зокрема, тест `AppController` перевіряє коректність роботи головного маршруту серверу та правильність повернення тестового повідомлення «Hello World!». Цей тест підтверджує працездатність базової конфігурації NestJS-застосунку та коректність створення контролера і сервісу.

Для модуля авторизації були створені тести `AuthController` та `AuthService`. У цих тестах перевірялась коректність взаємодії контролера із сервісом авторизації, правильність виклику методів входу користувача, створення JWT-токенів та встановлення cookie у відповідь сервера. Для ізоляції тестів використовувались mock-об'єкти сервісів та конфігураційних параметрів.

Тестування сервісу `UserService` перевіряло роботу з користувачами та збереженими матеріалами. У ході тестування було перевірено:

- створення нового користувача;
- хешування пароля перед збереженням;
- пошук користувача за ім'ям;
- додавання матеріалу до списку збережених;
- отримання списку збережених елементів;
- видалення матеріалу зі списку збережених.

Окремий набір тестів було створено для BucketService, який відповідає за взаємодію з хмарним файловим сховищем Cloudflare R2 через S3 API. Під час тестування перевірялись:

- ініціалізація S3-клієнта;
- завантаження файлів;
- створення папок;
- видалення файлів;
- видалення директорій разом із вкладеними файлами;
- отримання файлів зі сховища;
- генерація публічних URL-адрес.

Для уникнення реальних запитів до Cloudflare R2 використовувалось мокування S3-клієнта AWS SDK.

Тести FileService перевіряли логіку роботи з файлами та їх метаданими. Було протестовано:

- завантаження файлів та збереження інформації про них у MongoDB;
- формування повного шляху до файлів у вкладених директоріях;
- завантаження файлів зі сховища;
- оновлення зв'язків між файлами та папками;
- видалення файлів та їх фізичне видалення зі сховища;
- отримання списку файлів у певній папці.

Тестування FolderService охоплювало функціональність роботи з ієрархічною структурою папок. У тестах перевірялись:

- створення кореневих та вкладених папок;

- рекурсивна побудова деревоподібної структури;
- отримання дочірніх папок і файлів;
- взаємодія із файловим сховищем під час створення директорій.

Особливу увагу було приділено end-to-end тестуванню REST API контролерів. Такі тести перевіряють повний цикл роботи серверу: від HTTP-запиту до взаємодії з базою даних та повернення відповіді клієнту.

У тестах `SampleController` було перевірено:

- завантаження нового семплу;
- отримання семплу за ідентифікатором;
- оновлення даних семплу;
- заміну аудіо файлу;
- видалення семплу разом із файлом.

Для `PresetController` були протестовані:

- створення нового пресету;
- завантаження файлів пресету та демо-аудіо;
- отримання пресету;
- оновлення його параметрів;
- заміна файлів;
- видалення пресету.

У тестах `PackController` перевірялась найбільш складна функціональність серверу, пов'язана з ієрархічною структурою наборів. Було протестованно:

- створення набору;
- створення вкладених папок;
- додавання файлів до різних рівнів структури;
- отримання повної структури паку;
- отримання URL файлів;
- завантаження окремих файлів;

- завантаження папок у ZIP-архівах;
- видалення набору разом із вкладеними файлами та папками.

Для ізоляції тестування та зменшення залежності від зовнішніх сервісів у e2e тестах активно використовувались mock-реалізації сервісів `FileService`, `FolderService` та `MaterialService`.

Отримані результати тестування підтверджують коректність роботи серверної частини вебзастосунку, правильність реалізації REST API, взаємодії з MongoDB та Cloudflare R2, а також стабільність основних бізнес-процесів системи.

4.3.2 Автоматизоване тестування клієнтської частини застосунку

Для тестування верстки застосунку було використано тестувальний фреймворк Playwright [26]. Цей фреймворк надає інструменти для зручного крос-браузерного end-to-end тестування вебзастосунків, дозволяє перевіряти навігацію, взаємодії зі сторінками, видимість, текст, значення змінних та структуру сторінок за різних умов. Окрім цього, playwright також надає розробникам можливість «генерації» тестів командою `codegen`. При використанні цієї опції, розробник вказує адресу сторінки для тестування, і вона відкривається в окремо створеному вікні браузера (chromium за замовчуванням), разом з вікном коду тесту, а також інтерфейсом для вказування селекторів та перевірок (рис. 4.2). Під час використання цієї команди, playwright веде запис дій користувача, а розробник може створювати перевірки видимості, тексту, значення або структури елементів інтерфейсу.

В ході даної роботи, для кожної з розроблених сторінок інтерфейсу було створено тести. Спочатку для кожної сторінки було проведено тести доступні у середовищі `codegen`, а після — доповнено тестами створеними вручну в коді.

Також, в ході розробки тестів було виявлено проблему того, що код тестів не міг коректно працювати через те, що більшість сторінок застосунку

виконують перевірку на аутентифікацію користувача, і переводять його на сторінку логіну при невдалій перевірці.

Для вирішення цієї проблеми було створено скрипт, який через засоби playwright використовує сторінку логіну, вводить дані тестового користувача, та отримує cookie, який заноситься в файл `storageState.json`, і пізніше використовується у тестах для всіх інших сторінок через `test.use({ storageState: "storageState.json" });`

Для використання вже створених тестів використовується команда `prx playwright test`, яка автоматично знаходить усі створені тести, та виконує них у трьох різних середовищах — `chromium`, `firefox`, `webkit`. Опціонально також можна додати тестування у мобільних версіях браузерів, але для цього проєкту було вирішено зосереджуватися лише на «десктопних» браузерах.

Тести перевіряють:

- коректність завантаження сторінок;
- наявність та видимість елементів інтерфейсу;
- правильність роботи форм;
- можливість взаємодії з кнопками, таблицями, слайдерами та аудіоплеєрами;
- функціонування навігації між сторінками;
- роботу авторизації користувача;
- відображення та редагування ресурсів.

Першим етапом тестування стала перевірка сторінок авторизації та реєстрації користувача. Для сторінки `/login` тест перевіряє:

- наявність заголовка “Login”;
- поля введення імені користувача та паролю;
- кнопок “Sign In”, “Sign in with Google” та “Register”;
- можливість введення тексту у форму;
- коректне відображення всіх елементів інтерфейсу.

Аналогічно, для сторінки `/register` перевіряється:

- наявність полів Username, Email Address, Password та Confirm Password;

- кнопок “Register” та “Log In”;

- можливість введення даних у форму;

- коректність структури інтерфейсу сторінки.

Для тестування функціоналу авторизованого користувача було створено окремий сценарій авторизації, який виконує вхід у систему та зберігає стан браузера. Це дозволило повторно використовувати авторизовану сесію під час запуску інших тестів без необхідності повторного логіну.

Значна частина тестів була присвячена сторінкам типу explore. Зокрема, тест сторінки /explore/samples перевіряє:

- наявність навігаційного меню;

- наявність форми пошуку та фільтрації;

- коректність таблиці результатів;

- наявність кнопок відтворення, збереження, видалення та завантаження;

- відображення звукової хвилі;

- роботу слайдерів гучності та висоти звуку;

- функціонування кнопок “Play/Pause”;

- можливість введення даних у поля фільтрації.

Також тестуються стилі інтерфейсу, зокрема перевіряється використання необхідних шрифтів для заголовків та таблиць.

Для сторінки /explore/presets було створено тести, що перевіряють:

- поля пошуку за назвою, автором, VST-плагіном, жанром та типом звуку;

- можливість прослуховування демонстраційного аудіо;

- наявність кнопок збереження, видалення та завантаження;

- коректне відображення інформаційних карток пресетів.

Сторінка /explore/packs тестується на:

- коректність пошуку наборів;
- відображення інформаційних карток;
- наявність кнопок “Save Pack” та “Download Pack”;
- правильне відображення вкладених директорій та файлів.

Окремо було реалізовано тести для сторінок saved, які перевіряють:

- правильність відображення збережених ресурсів;
- роботу кнопок завантаження;
- відображення таблиць та інформаційних карток;
- доступність аудіопрогравачів та хвильових форм.

Для сторінок created тестування є більш комплексним, оскільки ці сторінки підтримують редагування матеріалів. Тести перевіряють:

- оновлення ВРМ, жанру, інструментів та інших параметрів;
- роботу кнопок “Update”, “Replace File”, “Delete”;
- взаємодію зі слайдерами;
- можливість прослуховування аудіо;
- редагування структури наборів та папок.

Наприклад, для сторінки /created/packs перевіряється:

- відкриття та закриття папок через Accordion-компонент;
- додавання та видалення файлів;
- завантаження наборів;
- відображення хвильових форм аудіофайлів;
- робота кнопок “Add Folder”, “Add File” та “Delete Folder”.

Для сторінки /profile тестується:

- коректне відображення імені та email користувача;
- відображення статистичної інформації профілю;
- доступність елементів інтерфейсу сторінки профілю.

Таким чином, створений набір тестів забезпечує автоматичну перевірку основного функціоналу інтерфейсу системи та дозволяє виявляти помилки

після внесення змін до застосунку. Використання Playwright дозволило реалізувати комплексне end-to-end тестування, яке охоплює як перевірку окремих компонентів інтерфейсу, так і повні сценарії взаємодії користувача із системою.

На рисунку 4.2 зображено інтерфейс playwright для вибору селекторів.

В ході тестування було виявлено помилки, які виводяться як в терміналі (рис. 4.3), так і у вікні результатів тестування (рис. 4.4), які було виправлено та пізніше отримано успішні результати тестування (рис. 4.5).

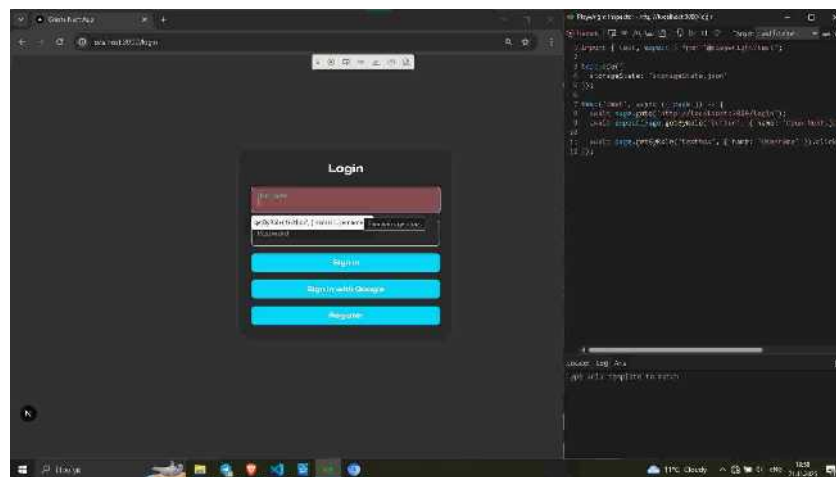


Рисунок 4.2 — Інтерфейсом Playwright для вказування селекторів та перевірок

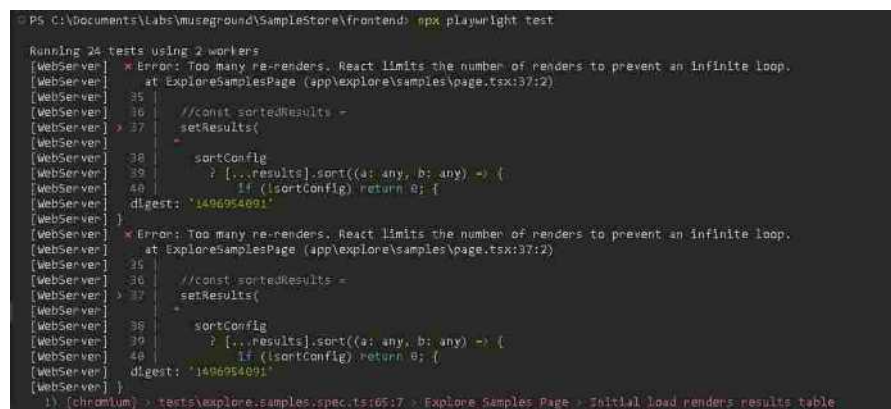


Рисунок 4.3 — Виведення помилок в терміналі

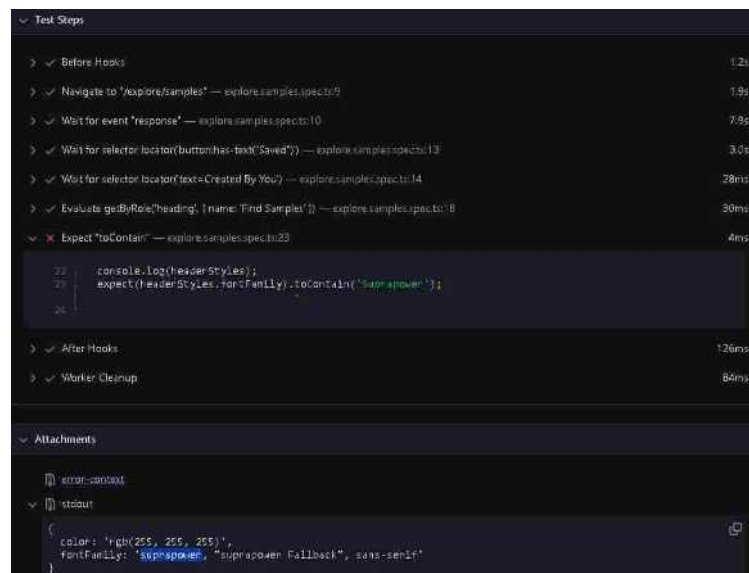


Рисунок 4.4 — Виведення помилок у вікні результатів тестування

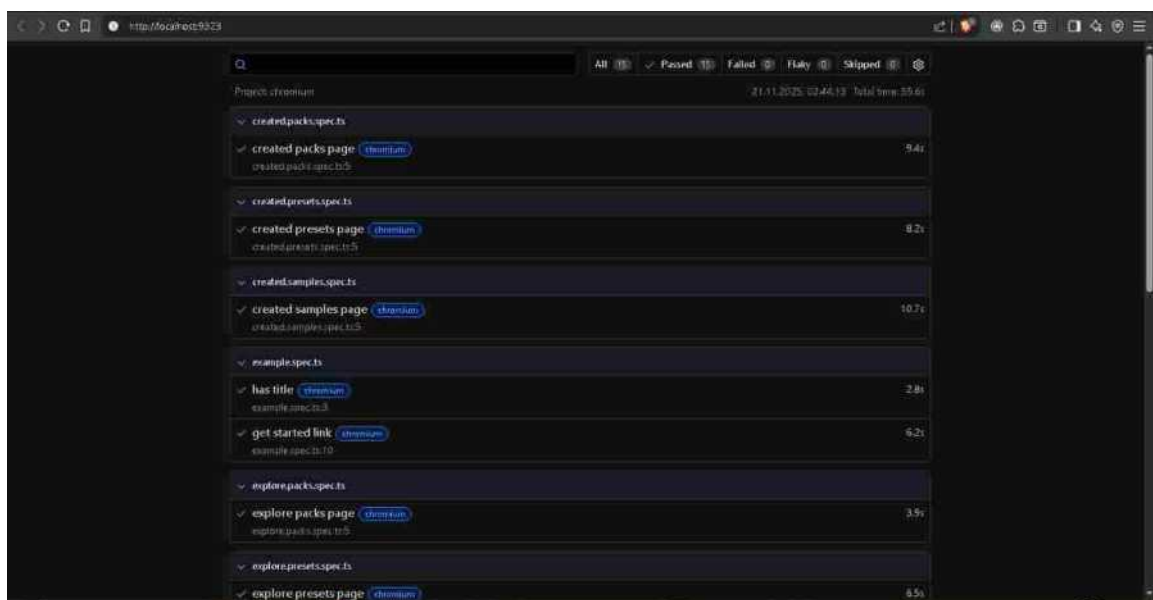


Рисунок 4.5 — Успішно пройдені тести після виправлення помилок

4.4 Висновки за розділом 4

У даному розділі було розглянуто процес експлуатації та тестування вебзастосунку «Museground». Було визначено призначення системи, умови її використання та описано основні сценарії взаємодії користувача із застосунком. Проведений аналіз показав, що розроблений вебзастосунок

забезпечує зручний інтерфейс для пошуку, завантаження, збереження та редагування музичних ресурсів, зокрема семплів, пресетів та наборів файлів.

У ході роботи було детально описано вхідні та вихідні дані застосунку, а також основні функціональні можливості різних сторінок системи. Реалізований інтерфейс дозволяє користувачам ефективно взаємодіяти із ресурсами, виконувати їх фільтрацію, відтворення аудіо, редагування та локальне завантаження.

Особливу увагу було приділено автоматизованому тестуванню серверної та клієнтської частин застосунку. Для серверної частини використано фреймворк Jest, за допомогою якого було реалізовано модульні та інтеграційні тести основних сервісів, контролерів та REST API. Результати тестування підтвердили коректність роботи бізнес-логіки, взаємодії з базою даних MongoDB та хмарним файловим сховищем Cloudflare R2.

Для тестування клієнтської частини застосунку було застосовано фреймворк Playwright, який дозволив перевірити роботу інтерфейсу у реальному браузері. Реалізовані end-to-end тести охопили ключові сценарії роботи користувача, включаючи авторизацію, навігацію, пошук, редагування та взаємодію з аудіофункціоналом системи. Використання автоматизованого тестування дозволило підвищити стабільність застосунку та спростити процес виявлення можливих помилок після внесення змін до програмного коду.

Отже, результати експлуатації та тестування підтверджують працездатність, стабільність та коректність функціонування вебзастосунку «Museground». Розроблена система успішно реалізує поставлені функціональні вимоги та може бути використана як повноцінна онлайн-платформа для обміну музичними ресурсами між користувачами.

ВИСНОВКИ

У ході виконання дипломної кваліфікаційної роботи, об'єктом дослідження якої є процес розповсюдження цифрових ресурсів для створення музики, а предметом — вебзастосунок для обміну та управління музичними ресурсами, було розроблено вебзастосунок «Museground», який надає користувачам зручні інструменти для пошуку, завантаження, збереження, редагування та обміну музичними матеріалами.

Актуальність даного дослідження обумовлена стрімким розвитком музичної індустрії та постійним збільшенням кількості цифрових ресурсів, які використовуються музичними продюсерами, звукорежисерами та виконавцями у процесі створення музики. Незважаючи на наявність великої кількості онлайн-платформ для розповсюдження музичних матеріалів, більшість із них спеціалізуються лише на окремих видах ресурсів або обмежуються контентом певних компаній. Це створює потребу у створенні універсальної системи, яка б забезпечувала єдине середовище для роботи із семплами, наборами семплів та пресетами для VST-плагінів.

У процесі виконання роботи було проведено аналіз предметної області та досліджено існуючі вебзастосунки для розповсюдження музичних ресурсів, зокрема Tracklib, Splice, Looperman, Cymatics, Plugin Boutique та PresetShare. Аналіз дозволив визначити їх основні переваги та недоліки, а також сформулювати функціональні та нефункціональні вимоги до розроблюваної системи.

У межах роботи було виконано поставлені завдання:

- проведено аналіз існуючих платформ для розповсюдження цифрових музичних ресурсів;
- визначено функціональні вимоги та межі розробки вебзастосунку;
- обґрунтовано вибір технологічного стеку для реалізації системи;
- спроектовано архітектуру застосунку, структуру бази даних та користувацький інтерфейс;

- реалізовано серверну та клієнтську частини вебзастосунку;
- створено REST API для роботи з музичними ресурсами та автентифікації користувачів;
- реалізовано систему роботи з файлами та хмарним файловим сховищем;
- проведено автоматизоване тестування серверної та клієнтської частин системи.

Під час розробки було використано сучасний технологічний стек. Клієнтську частину застосунку реалізовано за допомогою Next.js, React та TypeScript, що забезпечило високу продуктивність, компонентну архітектуру та зручність масштабування інтерфейсу. Серверну частину реалізовано на базі NestJS та Node.js із використанням TypeScript, що дозволило створити структуровану модульну архітектуру серверу. Для зберігання даних використано MongoDB Atlas, а для роботи з файлами — хмарне сховище Cloudflare R2. Розгортання клієнтської та серверної частин було реалізовано за допомогою платформ Vercel та Render відповідно.

У результаті виконання роботи було створено функціональний вебзастосунок «Museground», який забезпечує:

- реєстрацію та автентифікацію користувачів;
- пошук та фільтрацію музичних ресурсів за різними параметрами;
- завантаження, редагування та видалення семплів, пресетів і наборів;
- підтримку вкладеної файлової структури для наборів;
- прослуховування аудіофайлів безпосередньо у браузері;
- систему збереження ресурсів;
- локальне завантаження файлів користувачами.

Для перевірки працездатності системи було проведено комплексне автоматизоване тестування. Серверна частина тестувалася за допомогою Jest із використанням модульних та інтеграційних тестів, а клієнтська — за допомогою Playwright, який дозволив реалізувати end-to-end тестування

інтерфейсу у реальному браузері. Результати тестування підтвердили коректність роботи бізнес-логіки, REST API, взаємодії з базою даних та файловим сховищем, а також стабільність роботи користувацького інтерфейсу.

Практична цінність роботи полягає у створенні універсальної онлайн-платформи для обміну цифровими музичними ресурсами, яка може бути використана музичними продюсерами, саунд-дизайнерами та іншими учасниками музичної індустрії. Розроблений вебзастосунок може слугувати основою для подальшого розвитку системи, додавання соціальних функцій, рекомендаційних алгоритмів, інтеграції з DAW та розширення підтримки нових типів музичних ресурсів.

Отже, у результаті виконання дипломної кваліфікаційної роботи було досягнуто поставленої мети — розроблено та протестовано вебзастосунок для розповсюдження та обміну цифровими ресурсами для створення музики, який забезпечує зручний, стабільний та масштабований інструмент для роботи з музичним контентом у межах єдиної онлайн-платформи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Pro Tools Documentation. URL: <https://www.avid.com/pro-tools> (date of access: 11.05.2026).
2. Ableton Live Reference Manual. URL: <https://www.ableton.com/en/manual/welcome-to-live/> (date of access: 11.05.2026).
3. FL Studio User Manual. URL: <https://www.image-line.com/fl-studio-learning/fl-studio-online-manual/> (date of access: 11.05.2026).
4. REAPER User Guide. URL: <https://www.reaper.fm/userguide.php> (date of access: 11.05.2026).
5. Steinberg VST Plug-In SDK 3.7 Documentation. URL: https://steinbergmedia.github.io/vst3_doc/ (date of access: 11.05.2026).
6. Tracklib. URL: <https://www.tracklib.com/> (date of access: 11.05.2026).
7. Splice. URL: <https://splice.com/> (date of access: 11.05.2026).
8. Looperman. URL: <https://www.looperman.com/> (date of access: 11.05.2026).
9. Cymatics. URL: <https://cymatics.fm/> (date of access: 11.05.2026).
10. Plugin Boutique. Режим доступу: <https://www.pluginboutique.com/> (date of access: 20.05.2026).
11. Presetshare. URL: <https://presetshare.com/> (date of access: 11.05.2026).
12. TypeScript Handbook. URL: <https://www.typescriptlang.org/docs/> (date of access: 11.05.2026).
13. Node.js Documentation. URL: <https://nodejs.org/en/docs> (date of access: 11.05.2026).
14. Visual Studio Code Documentation. URL: <https://code.visualstudio.com/docs> (date of access: 11.05.2026).
15. React Documentation. URL: <https://react.dev/> (date of access: 11.05.2026).

16. Next.js Documentation. URL: <https://nextjs.org/docs> (date of access: 11.05.2026).
17. MongoDB MongoDB Atlas Documentation. URL: <https://www.mongodb.com/docs/atlas/> (date of access: 11.05.2026).
18. Cloudflare R2 Object Storage Documentation. URL: <https://developers.cloudflare.com/r2/> (date of access: 11.05.2026).
19. PlantUML Official Documentation. URL: <https://plantuml.com/> (date of access: 11.05.2026).
20. Mongoose Documentation. URL: <https://mongoosejs.com/docs/guide.html> (date of access: 11.05.2026).
21. Figma Help Center. URL: <https://help.figma.com/> (date of access: 11.05.2026).
22. ProtoPie Documentation. URL: <https://www.protopie.io/learn/docs> (date of access: 11.05.2026).
23. Wavesurfer.js. library. URL: <https://wavesurfer.xyz/>.
24. NestJS Documentation. URL: <https://docs.nestjs.com/> (date of access: 11.05.2026).
25. Jest Documentation. URL: <https://jestjs.io/docs/getting-started> (date of access: 11.05.2026).
26. Playwright Documentation. URL: <https://playwright.dev/docs/intro> (date of access: 11.05.2026).

ДОДАТОК А
Технічне завдання

A.1 Вступ

A.1.1 Найменування

Найменування застосунку -«Museground – платформа для обміну цифровими ресурсами для створення музики»

A.1.2 Коротка характеристика предметної області

Музична індустрія є постійно зростаючим та еволюціонуючим ринком. З кожним роком, кількість нових виконавців та загальна кількість музики лише збільшуються. Разом зі збільшенням людей зацікавлених у цій індустрії збільшується і кількість ресурсів та продуктів які використовуються авторами для як для самої роботи так і для її полегшення та пришвидшення. Створення та розповсюдження таких ресурсів є значною частиною економіки індустрії музики, але станом на сьогодні більшість онлайн платформ для розповсюдження таких ресурсів є обмеженими або у видах ресурсів які вони надають або у їх джерелі, надаючи лише продукти певної компанії.

Видами ресурсів які розглядаються в даній роботі є:

- пресети для VST плагінів — готові набори налаштувань які дозволяють зберігати та швидко отримувати готові звуки для інструментів, відповідні до ситуації налаштування ефектів;

- семпли — окремі звуки чи відрізки аудіо будь-якого походження, звуки перкусійних (ударних) інструментів або короткі вирізки з інших композицій;

- набори (sample racks) – набори семплів. Найчастіше використовуються як набори різних частин однієї перкусійної системи (drum kits).

A.2 Підстави для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Вебзастосунок обміну цифровими ресурсами для створення музики»,

затверджене наказом національного університету «Запорізька політехніка» № 139 від 07 квітня 2026 р.

А.3 Призначення розробки

Призначенням розробленого застосунку «Museground» є створення онлайн платформи де спільнота користувачів зацікавлена в створенні музики зможе шукати, отримувати та обмінюватись семплами, їх наборами та пресетами для VST-плагінів.

А.4 Вимоги до програми

А.4.1 Вимоги до функціональних характеристик

Авторизація та реєстрація:

- створення облікового запису прив'язаного до самого застосунку;
- створення облікового запису через використання сторонніх провайдерів, таких як google, github, discord тощо;
- вхід до застосунку через дані введені при реєстрації, якщо реєстрацію було проведено через сам застосунок;
- вхід до застосунку через сторонні провайдери.

Додавання матеріалів:

- зареєстрований користувач має мати змогу додавати семпли, набори та пресети;
- при додаванні семплу, користувач може вказати назву, темп, використаний(і) інструмент(и), жанр, темп, тональність, короткий опис, теги; при цьому лише назва є обов'язковою для введення; довжина, розмір файлу та дата створення файлу зчитуватимуться застосунком з метаданих аудіофайлу;
- при додаванні набору, користувач завантажує цілу папку, яка містить аудіофайли та декілька рівнів вкладених папок; альтернативно, користувач

може створювати папки на сторінці застосунку, обираючи які файли додавати до якої папки, створюючи структуру набору;

- при додаванні пресету, користувач має вказати назву, VST плагін до якого належить пресет, і опціонально жанр, «інструмент», короткий опис.

Пошук матеріалів:

- користувач може шукати (explore) семпли, набори та пресети;
- користувач може фільтрувати матеріали, шукаючи їх за назвою та/або будь-яким набором параметрів які мають відповідні матеріали, наприклад жанром;

- користувач може сортувати отримані результати, які за замовченням йдуть в порядку від найбільшої до найменшої кількості завантажень, при чому сортувати можливо за будь-якими з наявних параметрів; Якщо ж певний матеріал відповідає заданим фільтрам, але не містить частини інформації за якою відбувається сортування, такі матеріали будуть розміщені в кінці списку;

- користувач може завантажити будь-який знайдений матеріал; для цього, обраний у розділі explore матеріал можна буде «зберегти», що перенесе його у розділ «збережене», звідки користувач вже може його завантажити.

Профіль користувача:

- користувач має ім'я, зображення профілю, кількість завантажених, кількість доданих матеріалів, дати створення облікового запису, останніх завантажень та додавань, списки завантажених та доданих матеріалів;

- додані раніше матеріали можна видалити з застосунку;

- користувач має змогу перейти до сторінки «збережене», обрати там розділ за типом матеріалу, і отримати список збережених матеріалів, які він може переглядати, фільтрувати, шукати, сортувати, видаляти та завантажувати;

- користувач має змогу перейти до сторінки «мої матеріали», де йому буде надано аналогічний список матеріалів, які створив він сам.

A.4.2 Вимоги до користувацького інтерфейсу

У цьому розділі наведено опис вимог до складу сторінок користувацького інтерфейсу розроблюваного застосунку та основні вимоги до їх стилізації.

Основними стилістичними рішеннями, які визначено на даному етапі розробки, є:

- використання темної теми як основної;
- кольором фону сторінок є #333333;
- кольором тексту є #C9C9C9;
- кольором виділених елементів та акцентування є #00F0FF;
- якщо якийсь елемент сторінки потрібно акцентувати, наприклад при наведенні курсором, колір тексту змінюється на #2C2C2C а колір фону елемету змінюється на #00F0FF;

- усі заголовки використовують шрифт Suprapower;
- увесь звичайний текст використовує шрифт Programme.

Список сторінок який планується розробити:

- explore/samples;
- explore/packs;
- explore/presets;
- saved/samples;
- saved/packs;
- saved/presets;
- upload/sample;
- upload/pack;
- upload/preset;
- profile;
- saved;
- uploaded;
- login;

– register.

При вході на сайт, користувач буде потрапляти на сторінку explore/samples. Далі наведено елементи інтерфейсу які матиме ця сторінка.

Більшість сторінок інтерфейсу застосунку матимуть схожу структуру, тому повний опис сторінки буде наведено лише для explore/samples, а для наступних подібних сторінок буде описано лише відмінності.

Компонентами графічного інтерфейсу на сторінці explore/samples є:

– панель навігації згори сторінки яка містить посилання login, profile, saved, upload new, uploaded; У разі якщо користувач не ввійшов у систему, лише login буде активним посиланням; Аналогічна панель буде знаходитись на кожній сторінці застосунку;

– панель навігації одразу нижче попередньої, з посиланнями на explore/samples (активна), explore/packs, explore/presets. Аналогічна панель навігації буде присутня на усіх сторінках пов'язаних з діями які стосуються усіх трьох видів матеріалів (тобто на сторінках saved, upload, uploaded);

– форма для пошуку фільтрування результатів. Ця форма буде містити поле для пошуку, випадаючий який визначає параметр поля пошуку — назву, ім'я автора або тег, випадаючі списки для фільтру за параметрами семплу, тобто жанром, темпом, тональністю, тегами;

– випадаючий список для вибору параметра за яким будуть відсортовані результати, та чекбокс для сортування за зростанням/спаданням;

– кнопку «Шукати» та кнопку «Застосувати Фільтр»;

– список результатів, у вигляді таблиці, де кожний рядок це один з результатів, і стовпчики є інформацією про нього, а саме назвою, автором, довжиною, темпом, тональністю, інструментом та жанром. Окрім цього, кожен екземпляр буде мати виділене місце, в якому будуть розташовані кнопка «play», візуалізація звукової хвилі, слайдер гучності, кнопки підвищення/пониження висоти звуку, кнопка «зберегти» та список тегів.

Сторінка `explore/presets` відрізняється від `explore/samples` по суті лише параметрами якими описуються об'єкти. Замість описаних параметрів семплів, пресети мають назву, автора, назву VST плагіну, для якого зроблено пресет, тип звуку (наприклад `bass`, `pad`, `lead`) та жанр.

Сторінка `explore/packs` відрізняється тим, що список наборів буде відображатися не у вигляді таблиці, а як набір інформаційних карток, кожна з яких буде відображати окремий набір, і буде мати зображення, назву, автора, та декілька (~5) прикладів аудіофайлів що містяться у наборі, з можливістю почути приклади одразу в застосунку.

Сторінки розділів `saved` та `uploaded` матимуть схожу до розділу `explore` структуру. Різниця полягатиме в тому, що ці розділи замість кнопок «`save`» матимуть кнопки «`download`» та/або «`delete`».

Найбільшу різницю матиме сторінка `saved/packs`, яка буде відображати збережені набори у вигляді дерева директорій та файлів, де є корньова директорія, яка містить директорії які відображають кожен збережений набір, які в свою чергу містять власні директорії, файли та піддиректорії.

Сторінка `upload/sample` буде містити форму для вводи даних про семпл, описаних попередньо, а також кнопку для додавання аудіофайлу, та кнопки «додати» (`upload`). Сторінка `upload/preset` працює аналогічним чином.

Сторінка `upload/pack` буде мати два «режима» роботи. В першому, користувач завантажує папку, яка усніє на його комп'ютері, назва папки стає назвою набору, і вміст папки стає його вмістом. В другому, користувач вручну вводить назву набору та вручну створює структуру набору, створюючи папки всередині цієї сторінки та додаючи до них окремі файли.

Сторінка `profile` буде містити інформацію про обліковий запис користувача, таку як ім'я, кількість доданих та завантажених матеріалів, тощо. Також, ця сторінка буде містити кнопку виході з облікового запису.

Сторінка `login` буде містити форму для входу посередині екрану, в якій будуть поля для введення ім'я та паролю, кнопки «`login`», «`register`» та кнопки

для входу через google або github. Сторінка register працюватиме аналогічним чином.

A.4.3 Вимоги до надійності та експлуатації

У розроблюваному застосунку існує 4 чинники, які можуть суттєво вплинути на швидкість роботи і погіршити досвід користувача. Ними є швидкість виконання запитів до бази даних, швидкість додавання файлів, швидкість завантаження файлів з застосунку на комп'ютер та швидкість рендеру сторінки, яка може погіршуватись на сторінках, які відображають багато аудіофайлів, або використовують візуалізацію звукових хвиль файлів.

У ході розробки необхідно виділити час на оптимізацію цих процесів, застосувати засоби оптимізації запитів до бази даних, рендеру елементів сторінок, швидкість обробки файлів та комунікації сервера з клієнтом. Щодо експлуатації, застосунок розроблюється для використання у браузері на основі chromium на екранах розміру десктопних комп'ютерів.

A.4.4 Вимоги до складу й параметрів технічних засобів

Для коректної роботи вебзастосунку “Museground” необхідно таке апаратне забезпечення: не менше 2 Гб оперативної пам'яті, а також 5 Гб вільного простору на накопичувачі будь-якого типу;

ДОДАТОК Б
Текст програми

Б.1 Текст файла backend/main.ts

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from '../app.module';
import { ValidationPipe } from '@nestjs/common';
import cookieParser from 'cookie-parser';
import {
  Injectable,
  CallHandler,
  ExecutionContext,
  NestInterceptor,
} from '@nestjs/common';
import { Observable, tap } from 'rxjs';

@Injectable()
export class TraceInterceptor implements NestInterceptor {
  intercept(context: ExecutionContext, next: CallHandler): Observable<any> {
    const handler = context.getHandler().name;
    const className = context.getClass().name;

    console.log(`--> Entering: ${className}.${handler}()`);

    return next.handle().pipe(
      tap(() => {
        console.log(`<-- Exiting: ${className}.${handler}()`);
      }),
    );
  }
}

async function bootstrap(): Promise<void> {
  const app = await NestFactory.create(AppModule);

  // Properly typed middleware usage
  // eslint-disable-next-line @typescript-eslint/no-unsafe-call
  app.use(cookieParser());

  app.useGlobalPipes(
    new ValidationPipe({
      whitelist: true,
      transform: true,
    }),
  );
  app.useGlobalInterceptors(new TraceInterceptor());

  app.enableCors({
    origin: ['http://localhost:3000', 'https://museground.netlify.app'],
    methods: 'GET,HEAD,PUT,PATCH,POST,DELETE',
    credentials: true, // important for cookies/authorization headers
    allowedHeaders: [
      'Content-Type',
      'Authorization',
      'X-Requested-With',
      'Accept',
      'Origin',
    ],
    exposedHeaders: [
      'Set-Cookie', // allow frontend to see cookies in response
    ],
  });
  const port = process.env.PORT ? Number(process.env.PORT) : 3001;
```

```

    await app.listen(port);
    console.log(`API listening on ${port}`);
  }

void bootstrap();

```

Б.2 Текст файла backend/app.module.ts

```

import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { MongooseModule } from '@nestjs/mongoose';
import { UsersModule } from './users/users.module';
import { AuthModule } from './auth/auth.module';
import { JwtModule } from '@nestjs/jwt';
import { MaterialModule } from './material/material.module';
import { SampleSchema } from './schemas/sample.schema';
import { PresetSchema } from './schemas/preset.schema';
import { PackSchema } from './schemas/pack.schema';
import { BucketModule } from './r2bucket/bucket.module';
import { SavedItemModule } from './savedItems/savedItem.module';
//import { DevtoolsModule } from '@nestjs/devtools-integration';

@Module({
  imports: [
    ConfigModule.forRoot({ isGlobal: true }),
    MongooseModule.forRoot(process.env.MONGODB_URI as string),
    UsersModule,
    SavedItemModule,
    AuthModule,
    JwtModule.registerAsync({
      imports: [ConfigModule],
      inject: [ConfigService],
      useFactory: (configService: ConfigService) => ({
        secret: configService.get<string>('JWT_SECRET'),
        signOptions: { expiresIn: '120000s' },
      }),
      global: true,
    }),
    BucketModule,
    MaterialModule.register({ modelName: 'Sample', schema: SampleSchema }),
    MaterialModule.register({ modelName: 'Preset', schema: PresetSchema }),
    MaterialModule.register({ modelName: 'Pack', schema: PackSchema }),
  ],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}

```

Б.3 Текст файла backend/users/users.controller.ts

```

import { Body, Controller, Get, Param, Post } from '@nestjs/common';
import { UsersService } from './users.service';

```

```

import { CreateUserDTO } from '../DTO/createUser.dto';
import { SaveItemDTO } from '../DTO/saveItem.dto';

@Controller('users')
export class UsersController {
  constructor(private userService: UsersService) {}

  @Post('register')
  createUser(@Body() createUserDTO: CreateUserDTO) {
    //console.log(JSON.stringify(createUserDTO));
    return this.userService.createUser(createUserDTO);
  }

  @Post('save-item')
  async saveItem(@Body() saveItemDto: SaveItemDTO) {
    return await this.userService.saveItem(saveItemDto);
  }

  @Post('delete-saved')
  async deleteSavedItem(@Body() saveItemDto: SaveItemDTO) {
    //console.log(`deleting saved: ${JSON.stringify(saveItemDto)}`);
    return await this.userService.deleteSavedItem(saveItemDto);
  }

  @Get('saved-samples/:userId')
  async getSavedSamples(@Param('userId') userId: string) {
    return await this.userService.getSavedItems(userId, 'Sample');
  }

  @Get('saved-presets/:userId')
  async getSavedPresets(@Param('userId') userId: string) {
    return await this.userService.getSavedItems(userId, 'Preset');
  }

  @Get('saved-packs/:userId')
  async getSavedPacks(@Param('userId') userId: string) {
    return await this.userService.getSavedItems(userId, 'Pack');
  }
}

```

Б.4 Текст файла backend/users/users.service.ts

```

import { BadRequestException, Injectable } from '@nestjs/common';
import { InjectModel } from '@nestjs/mongoose';
import { Model } from 'mongoose';
import { User, UserDocument } from '../schemas/user.schema';
import { CreateUserDTO } from '../DTO/createUser.dto';
import { SaveItemDTO } from '../DTO/saveItem.dto';
import * as bcrypt from 'bcrypt';
import { SavedItem, SavedItemDocument } from '../schemas/savedItem.schema';

@Injectable()
export class UsersService {
  constructor(
    @InjectModel(User.name) private userModel: Model<UserDocument>,
    @InjectModel(SavedItem.name)
    private savedItemModel: Model<SavedItemDocument>,
  ) {}

```

```

async createUser(createUserDTO: CreateUserDTO): Promise<UserDocument> {
  //console.log('creating a new user\n');
  if (createUserDTO.password) {
    createUserDTO.password = await bcrypt.hash(createUserDTO.password, 10);
  }

  const newUser = new this.userModel({
    ...createUserDTO,
    provider: createUserDTO.provider ?? 'local',
  });

  return newUser.save();
}

async findOne(username: string): Promise<UserDocument | null> {
  return this.userModel.findOne({ username }).exec();
}

async findById(id: string): Promise<UserDocument | null> {
  return this.userModel.findById(id).exec();
}

async findByName(username: string): Promise<UserDocument | null> {
  return this.userModel.findOne({ username }).exec();
}

async findByEmail(email: string): Promise<UserDocument | null> {
  return this.userModel.findOne({ email }).exec();
}

async saveItem(saveItemDto: SaveItemDTO) {
  //TODO: add check if saved item is existing
  await new this.savedItemModel(saveItemDto).save();
  const user = await this.userModel.findById(saveItemDto.userId).exec();
  if (user) {
    user.totalDownloads += 1;
    return user.save();
  } else throw new BadRequestException("Couldn't Save Item");
}

async getSavedItems(userId: string, type: string) {
  return await this.savedItemModel
    .find({ userId: userId, itemType: type })
    .exec();
}

async deleteSavedItem(saveItemDto: SaveItemDTO) {
  return await this.savedItemModel
    .deleteOne({
      userId: saveItemDto.userId,
      itemType: saveItemDto.itemType,
      itemId: saveItemDto.itemId,
    })
    .exec();
}
}

```

Б.5 Текст файлу backend/material.module.ts

```
@Module({})
export class MaterialModule {
  static register<T extends Document>(
    options: MaterialModuleOptions,
  ): DynamicModule {
    const controllerMap: Record<string, Type<any>> = {
      Sample: SampleController,
      Preset: PresetController,
      Pack: PackController,
    };

    const modelToken = getModelToken(options.modelName);
    const savedItemModelToken = getModelToken(SavedItem.name);

    const materialProvider: Provider = {
      provide: MaterialService,
      useFactory: (model: Model<T>, savedItemModel: Model<SavedItemDocument>) =>
        new MaterialService<T>(options, model, savedItemModel),
      inject: [modelToken, savedItemModelToken],
    };

    return {
      module: MaterialModule,
      imports: [
        MongooseModule.forFeature([
          { name: options.modelName, schema: options.schema as Schema },
          { name: SavedItem.name, schema: SavedItemSchema },
        ]),
        FileModule,
        FolderModule,
        UsersModule,
      ],
      providers: [
        { provide: MATERIAL_OPTIONS, useValue: options },
        materialProvider,
      ],
      controllers: [controllerMap[options.modelName]],
      exports: [MaterialService],
    };
  }
}
```

Б.6 Текст файлу frontend/login/page.tsx

```
"use client";
import { useState } from "react";
import { useRouter } from "next/navigation";
import { BACKEND_URL } from "../../constants";
import { useAuth } from "../../context/AuthContext";
export default function Login() {
  const { refetchUser } = useAuth();
  const [error, setError] = useState<string>("");
  const [formData, setFormData] = useState({ username: "", password: "" });
  const router = useRouter();
```

```

const handleFormDatachange = (e: React.ChangeEvent<HTMLInputElement>) => {
  setFormData({ ...formData, [e.target.name]: e.target.value });
};

const handleLogin = async (e: React.FormEvent<HTMLFormElement>) => {
  e.preventDefault();
  setError("");

  try {
    //alert(`${BACKEND_URL}/auth/login-jwt`);
    //alert(JSON.stringify(formData));
    const res = await fetch(`${BACKEND_URL}/auth/login-jwt`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      credentials: "include",
      body: JSON.stringify(formData),
    });

    const data = await res.json();
    //alert(JSON.stringify(data));
    if (res.ok) {
      if (res.ok) {
        await refetchUser(); // ensures AuthProvider has user info
        router.push("/explore/samples");
      }
    } else {
      setError(data?.error ?? "Login failed");
    }
  } catch (err) {
    console.error("Unexpected Error", err);
    setError("Could not connect to server");
  }
};

const handleGoogleLogin = () => {
  window.location.href = `${BACKEND_URL}/auth/google`;
};

return (
  <div className="flex items-center justify-center min-h-screen px-4">
    <div className="w-full max-w-md">
      <div className="bg-zinc-800 shadow-xl rounded-2xl p-6">
        {error && (
          <div className="mb-4 text-sm text-red-500 bg-red-100 p-2 rounded">
            {error}
          </div>
        )}

        <h1 className="text-2xl font-bold text-center mb-6 text-white-shadow font-suprapower">
          Login
        </h1>

        <form onSubmit={handleLogin} className="space-y-4">
          <div className="relative">
            <input
              type="text"
              id="username"
              name="username"
              required
              value={formData.username}
              placeholder=" "
              onChange={handleFormDatachange}

```

```

        className="peer w-full rounded-lg border border-gray-300 px-3
pt-5 pb-2 text-white-shadow placeholder-transparent focus:border-blue-500
focus:ring focus:ring-blue-200"
      />
      <label
        htmlFor="username"
        className="absolute left-3 top-2 text-cyan-400 text-sm
transition-all peer-placeholder-shown:top-4 peer-placeholder-shown:text-gray-400
peer-placeholder-shown:text-base peer-focus:top-2 peer-focus:text-sm peer-
focus:text-cyan-400"
      >
        Username
      </label>
    </div>

    <div className="relative">
      <input
        type="password"
        id="password"
        name="password"
        required
        placeholder=" "
        value={formData.password}
        onChange={handleFormDatachange}
        className="peer w-full rounded-lg border border-gray-300 px-3
pt-5 pb-2 text-white-shadow placeholder-transparent focus:border-blue-500
focus:ring focus:ring-blue-200"
      />
      <label
        htmlFor="password"
        className="absolute left-3 top-2 text-cyan-400 text-sm
transition-all peer-placeholder-shown:top-4 peer-placeholder-shown:text-gray-400
peer-placeholder-shown:text-base peer-focus:top-2 peer-focus:text-sm peer-
focus:text-cyan-400"
      >
        Password
      </label>
    </div>

    <button
      type="submit"
      className="w-full py-2 rounded-lg bg-cyan-400 text-white font-
suprapower font-semibold hover:bg-blue-500 transition"
    >
      Sign In
    </button>
  </form>

  <div className="mt-4">
    <button
      onClick={handleGoogleLogin}
      className="w-full py-2 rounded-lg bg-cyan-400 text-white font-
suprapower font-semibold hover:bg-blue-500 transition"
    >
      Sign in with Google
    </button>
  </div>

  <div className="mt-4">
    <button
      type="button"
      onClick={() => (window.location.href = "/register")}

```

```

        className="w-full py-2 rounded-lg bg-cyan-400 text-white font-
suprapower font-semibold hover:bg-blue-500 transition"
      >
        Register
      </button>
    </div>
  </div>
</div>
</div>
);
}

```

Б.7 Текст файлу frontend/explore/samples/page.tsx

```

"use client";
import SampleCard from "@app/components/cards/sample";
import Header from "@app/components/header";
import { useState, useEffect, useMemo } from "react";
import { BACKEND_URL } from "@app/constants";

export default function ExploreSamplesPage() {
  const [sampleFilterFormData, setSampleFilterFormData] = useState({
    name: "",
    author: "",
    minBPM: "",
    maxBPM: "",
    instruments: "",
    genres: "",
    key: "",
  });

  const [results, setResults] = useState<unknown[]>([]);
  const [error, setError] = useState("");
  const [sortConfig, setSortConfig] = useState<{
    key: string;
    direction: "asc" | "desc";
  } | null>(null);

  const handleFormDataChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    const { name, value, type } = e.target;

    setSampleFilterFormData({
      ...sampleFilterFormData,
      [name]: type === "number" && value !== "" ? Number(value) : value,
    });
  };

  const sortedResults = useMemo(() => {
    if (!sortConfig) return results;

    const { key, direction } = sortConfig;
    return [...results].sort((a: any, b: any) => {
      const dir = direction === "asc" ? 1 : -1;
      return a[key] > b[key] ? dir : -dir;
    });
  }, [results, sortConfig]);

  const handleSort = (key: string) => {
    setSortConfig((prev) => {
      if (prev && prev.key === key) {

```

```

        // toggle asc/desc
        return { key, direction: prev.direction === "asc" ? "desc" : "asc" };
    }
    return { key, direction: "asc" };
});
};

const handleFilterQuery = async (e?: React.FormEvent) => {
    if (e) e.preventDefault();

    const filteredData: Record<string, string> = {};
    Object.entries(sampleFilterFormData).forEach(([key, value]) => {
        if (
            (typeof value === "string" && value.trim() !== "") ||
            (typeof value === "number" && !isNaN(value) && value !== 0)
        ) {
            filteredData[key] = value.toString();
        }
    });

    const queryString = new URLSearchParams(filteredData).toString();
    const url = `${BACKEND_URL}/samples/filter/query?${queryString}`;
    try {
        const res = await fetch(url);
        if (!res.ok) throw new Error("Failed to fetch samples");
        const data = await res.json();

        setResults(Array.isArray(data) ? data : (data.filterResult ?? []));
    } catch (err) {
        console.error(err);
        setError("Error fetching filtered samples");
    }
};

const handleDeleteSampleFromList = (id: string) => {
    setResults((prev) => prev.filter((sample: any) => sample._id !== id));
};

useEffect(() => {
    handleFilterQuery();
}, []);

return (
    <>
        <Header activeTop="Explore" activeBottom="Samples" />
        <main className="bg-[#333333] py-20 px-3">
            <h1 className="py-5 text-3xl">Find Samples</h1>
            <form onSubmit={handleFilterQuery} className="relative space-y-6">
                <div key="name" className="relative">
                    <input
                        type="text"
                        id="name"
                        name="name"
                        value={sampleFilterFormData.name}
                        placeholder=" "
                        onChange={handleFormDataChange}
                        className="bg-zinc-800 w-full text-xl size-18
peer rounded-lg border border-gray-300 px-3 pt-5 text-shadow-white
placeholder-transparent focus:border-cyan-400 focus:ring focus:ring-blue-
200"
                    />
                    <label
                        htmlFor="name"
                        className="absolute left-3 top-2 text-cyan-400 text-xl transition-
all
peer-placeholder-shown:top-5.5 peer-placeholder-shown:text-gray-200

```

```

peer-placeholder-shown:text-xl peer-focus:top-1 peer-focus:text-xl
peer-focus:text-sky-500"
    >
      Title
    </label>
  </div>

  <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 gap-6">
    [[
      { label: "Author", name: "author", type: "text" },
      { label: "Min BPM", name: "minBPM", type: "number" },
      { label: "Max BPM", name: "maxBPM", type: "number" },
      { label: "Instruments", name: "instruments", type: "text" },
      { label: "Genres", name: "genres", type: "text" },
      { label: "Key", name: "key", type: "text" },
    ].map((field) => (
      <div key={field.name} className="relative">
        <input
          type={field.type}
          id={field.name}
          name={field.name}
          value={
            sampleFilterFormData[
              field.name as keyof typeof sampleFilterFormData
            ]
          }
          placeholder=""
          onChange={handleFormDataChange}
          className="bg-zinc-800 w-full text-xl size-18
peer rounded-lg border border-gray-300 px-3 pt-5 text-shadow-white
placeholder-transparent focus:border-cyan-400 focus:ring focus:ring-blue-
200"
        />
        <label
          htmlFor={field.name}
          className="absolute left-3 top-2 text-cyan-400 border-
[#3b82f6] text-xl transition-all
peer-placeholder-shown:top-5.5 peer-placeholder-shown:text-gray-200
peer-placeholder-shown:text-xl peer-focus:top-1 peer-focus:text-xl
peer-focus:text-blue-600"
        >
          {field.label}
        </label>
      </div>
    )))
  </div>
  <button
    type="submit"
    className="w-full py-2 rounded-lg bg-cyan-400 text-white font-
suprapower font-semibold transition hover:text-[#343a40] hover:bg-sky-500"
  >
    Search
  </button>
</form>

{error && <p className="mt-4 text-red-600">{error}</p>}

{results.length > 0 && (
  <div className="mt-10 overflow-x-auto rounded-xl border border-gray-
700 bg-zinc-900 shadow-lg">
    <table className="w-full text-left text-gray-200">
      <thead className="bg-zinc-800 text-cyan-400 uppercase text-sm
tracking-wide">

```

```

    <tr>
      <th className="px-4 py-3">Play / Pause</th>
      <th className="px-4 py-3">Waveform</th>
      <th className="px-4 py-3">Volume</th>
      <th className="px-4 py-3">Pitch</th>
      {[
        { label: "Title", key: "name" },
        { label: "Key", key: "key" },
        { label: "BPM", key: "BPM" },
        { label: "Genres", key: "genres" },
        { label: "Instruments", key: "instruments" },
        { label: "Author", key: "authorName" },
      ]}.map(({ label, key }) => (
        <th
          key={key}
          onClick={() => handleSort(key)}
          className="px-4 py-3 cursor-pointer select-none
hover:text-white"
        >
          {label}
          {sortConfig?.key === key && (
            <span className="ml-1">
              {sortConfig.direction === "asc" ? "▲" : "▼"}
            </span>
          )}
        </th>
      ))}
      <th className="px-4 py-3">Save</th>
    </tr>
  </thead>
  <tbody className="divide-y divide-gray-700">
    {sortedResults.map((sample: any) => (
      <SampleCard
        key={sample._id}
        sample={sample}
        onDelete={handleDeleteSampleFromList}
      />
    ))}
  </tbody>
</table>
</div>
  )}
</main>
</>
);
}

```

ДОДАТОК В
Слайди презентації



Рисунок В.1 — Титульний слайд



Рисунок В.2 - Об'єкт та предмет дослідження, мета роботи

ПОРІВНЯННЯ ЗАСТОСУНКІВ ДЛЯ РОЗПОВСЮДЖЕННЯ МУЗИЧНИХ РЕСУРСІВ

3

Характеристика	Tracklib	Splice	Looperman	Cymatics	pluginboutique	presetshare
Цінова політика	підписка	підписка	безкоштовно	Окремі продукти, деякі безкоштовні	Окремі продукти, деякі безкоштовні	безкоштовно
Простота використання	Висока	Висока	Низька	Середня	Низька	Середня
Інтегрованість з DAW	Висока	Дуже висока	Низька	Низька	Середня	Низька
Тип продукції	Ліцензовані треки	Семпли, плагіни, пресети	Семпли	Набори, плагіни	Плагіни, пресети	Пресети, семпли
Розмір бібліотек	Високий	Високий	Низький	Середній	Високий	Середній
Підтримка	Висока	Висока	Маленька	Маленька	Середня	Низька
Якість продуктів	Висока	Висока	Низька	Середня	Середня	Низька

Рисунок В.3 - Порівняння застосунків для розповсюдження музичних ресурсів

ПОРІВНЯННЯ МОВ ПРОГРАМУВАННЯ

4

ДЛЯ КЛІЄНТУ ЗАСТОСУНКА

ДЛЯ СЕРВЕРНОЇ ЧАСТИНИ ЗАСТОСУНКА

Характеристика	Javascript	Typescript
Типізація	Динамічна, слабка	Статична, сильна
Виявлення помилок	Переважно під час виконання	При компіляції
Безпечність коду	нижча	вища
Зручність підтримки великих проєктів	обмежена	висока
Підтримка IDE	часткова	повна
Зручність рефакторингу	низька	вища
Сумісність з браузерами	повна	Повна після компіляції
Підтримка сучасних фреймворків	висока	висока
Підтримка інструментів	базова	розширена

Характеристика	Typescript	Java	C#	Go
Швидкість розробки	Висока	Середня	Середня	Висока
Простота синтаксису	Висока	Нижча	Середня	Висока
Можливість використання на клієнтській стороні	так	ні	ні	ні
Підтримка сучасної веброзробки	висока	висока	висока	середня
Зручність підтримки невеликих проєктів	висока	середня	середня	висока
Продуктивність	середня	висока	висока	висока
Популярність у сучасних вебзастосунках	Дуже висока	висока	висока	висока
Екосистема прип	повна	відсутня	відсутня	обмежена

Рисунок В.4 — Порівняння мов програмування



Рисунок В.5 — Вибір середовища розробки



Рисунок В.6 —Порівняння фреймворків для розробки

7

ВИБІР СИСТЕМИ
УПРАВЛІННЯ БАЗОЮ
ДАНИХ

Характеристика	MongoDB Atlas	Firebase Firestore/ Realtime Database	Supabase (PostgreSQL)	MySQL
Модель даних	Документи, гнучка схема	Документи/ колекції, realtime- синхронізація	Реляційна Postgres- модель, Realtime, Auth, Storage	Реляційні таблиці
Гнучкість структури	Висока	Висока	Середня	низька
Підтримка валідації	Schema validation та mongoose validation	Є правила доступу та структури даних, але модель жорсткіша	RLS та політика SQL	Constraints та реляційні обмеження
Зручність для typescript/node.js	Висока через Mongoose TypeScript support	Потрібна окрема інтеграція SDK	висока, але орієнтована на Postgres- екосистему	Залежить від ORM/драйвера
Відповідність проекту	висока	середня	середня	середня

Рисунок В.7 — Вибір Системи управління базою даних

8

ВИБІР ХМАРНОГО
СХОВИЩА ДАНИХ

Характеристика	Cloudflare R2	Firebase Storage	Supabase Storage
Тип сховища	об'єктне S3- сумісне сховище	хмарне сховище в Google Cloud Storage bucket	сховище файлів, інтегроване з Postgres
Сумісність з S3	так	ні	ні
Вартість передачі даних	безкоштовна	залежить від правил Google Cloud	залежить від тарифної моделі Supabase
Масштабованість	висока, глобальна мережа Cloudflare	висока, Google Cloud інфраструктура	висока, у межах платформи Supabase
Зручність інтеграції з node.js	висока через S3 API	висока через SDK Firebase	висока через інтеграцію з Supabase- екосистемою
Відповідність проекту	висока	середня	середня

Рисунок В.8 — Вибір хмарного сховища даних

АРХІТЕКТУРА КЛАСІВ ЗАСТОСУНКУ

9

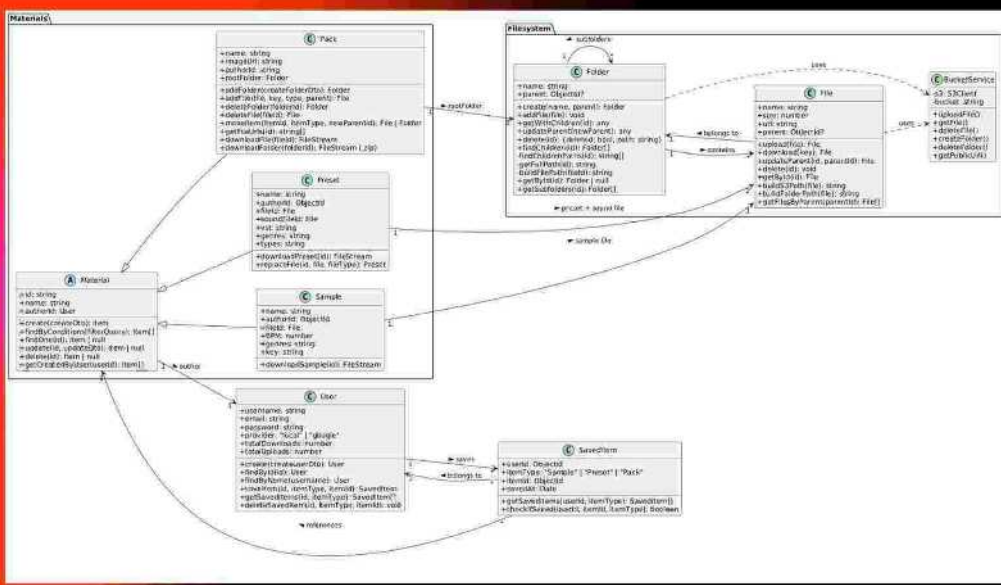


Рисунок В.9 — Архітектура класів застосунку

ДІАГРАМА СУТНОСТЕЙ БАЗИ ДАНИХ

10

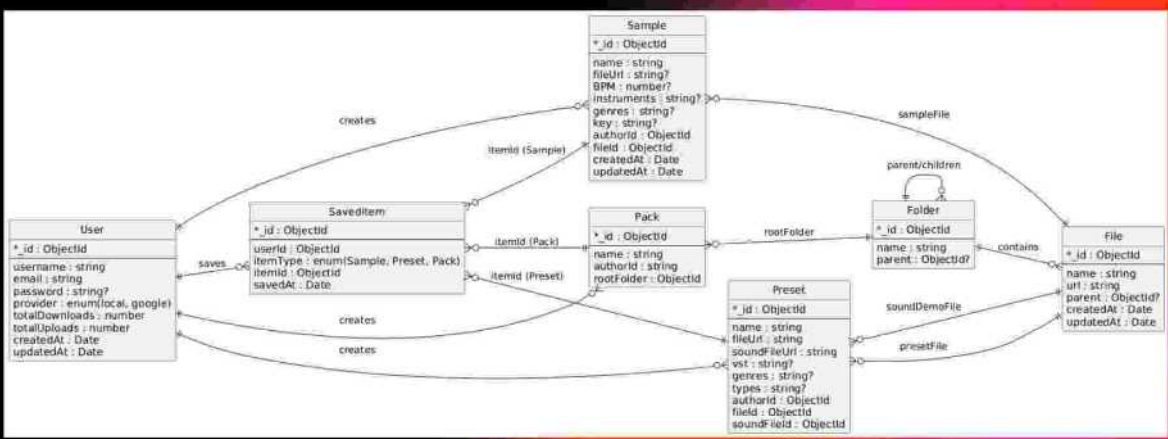


Рисунок В.10 — Діаграма сутностей бази даних



Рисунок В.11 — Абстрактна діаграма прецедентів застосунку

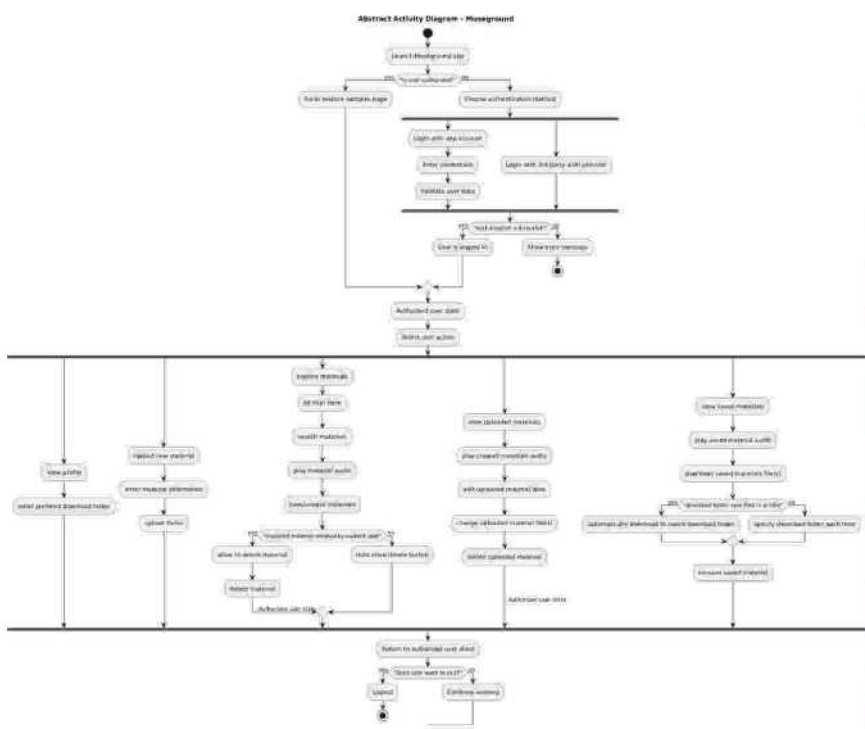


Рисунок В.12 — Абстрактна діаграма діяльності системи

ДІАГРАМА КОМПОНЕНТІВ СЕРВЕРУ ЗАСТОСУНКА

13

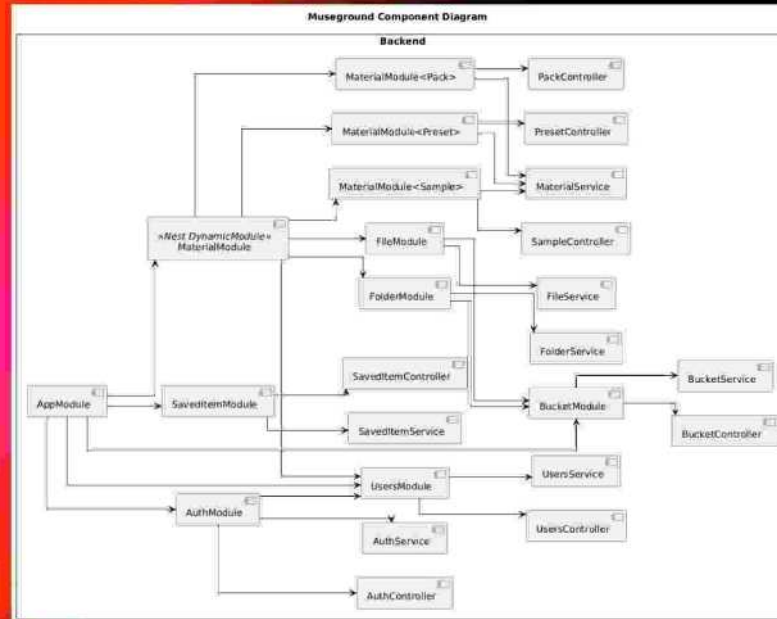


Рисунок В.13 — Діаграма компонентів серверу застосунка

ДІАГРАМА КОМПОНЕНТІВ КЛІЄНТУ ЗАСТОСУНКА

СТРУКТУРА ФАЙЛІВ ПРОЄКТУ

14

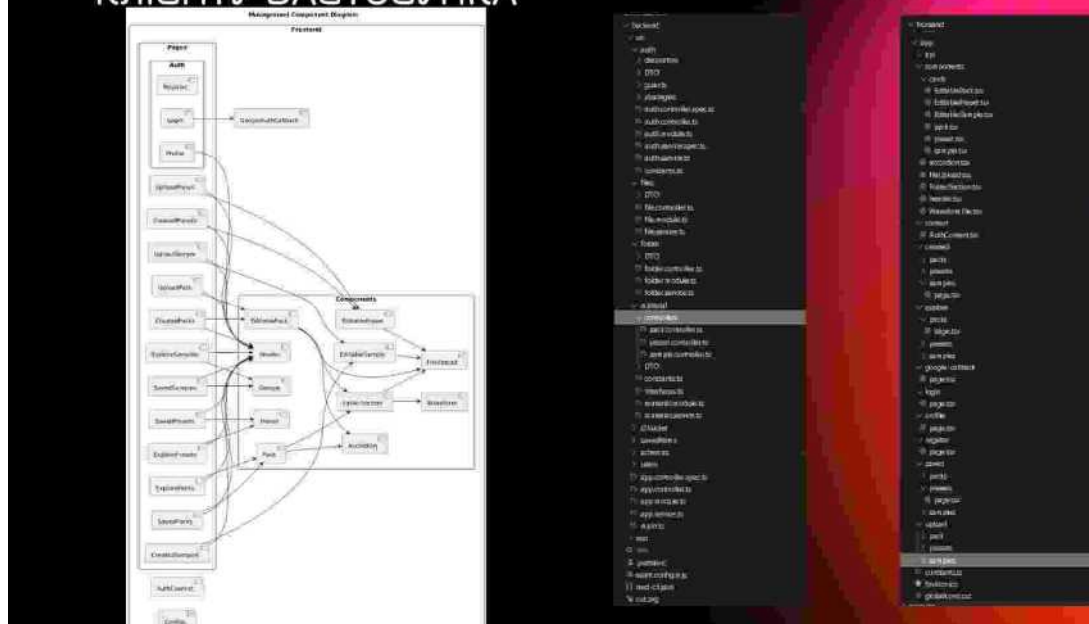


Рисунок В.14 — Діаграма компонентів клієнту застосунка та структура файлів проекту

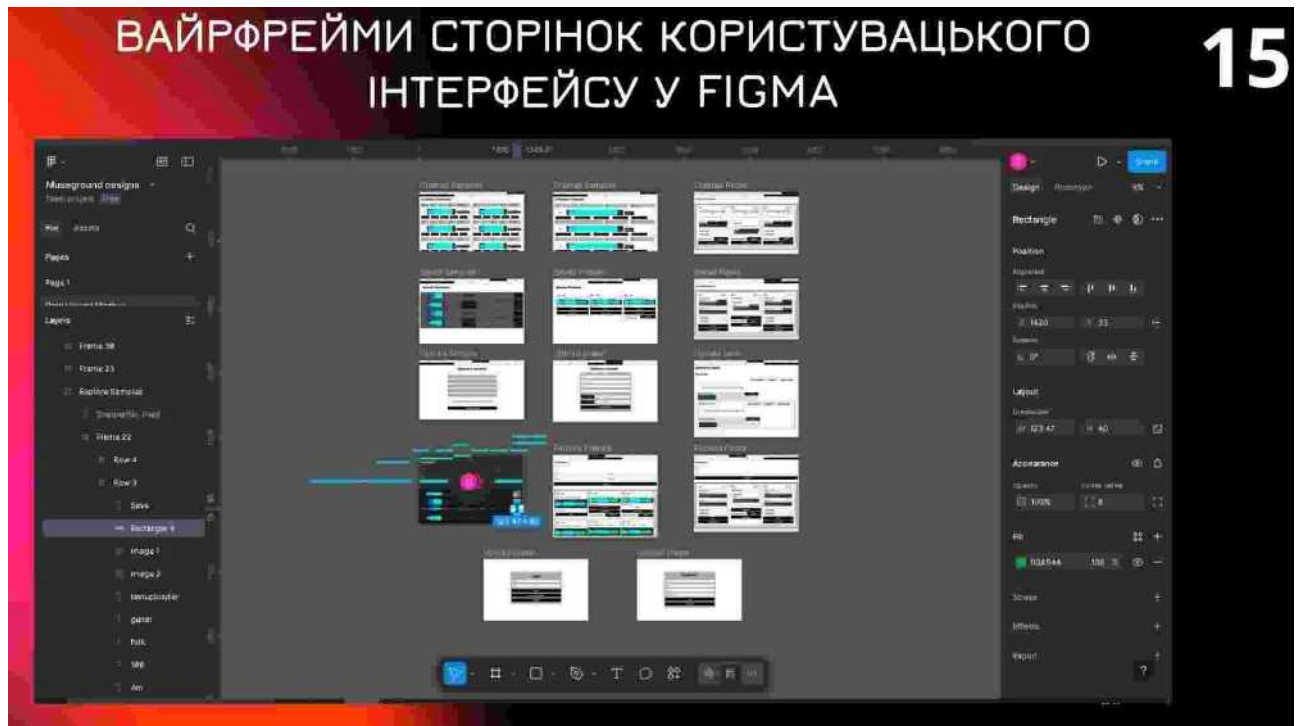


Рисунок В.15 — Вайрфрейми сторінок користувацького інтерфейсу і Figma

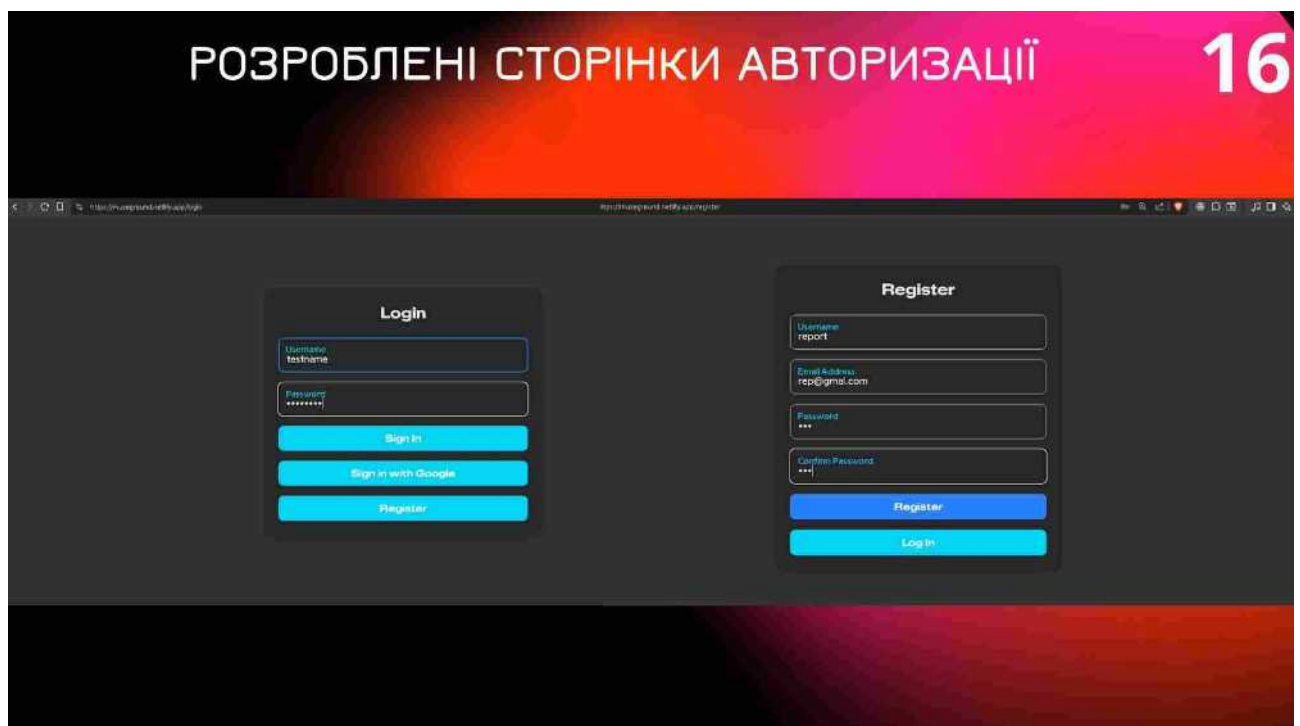


Рисунок В.16 — Розроблені сторінки авторизації

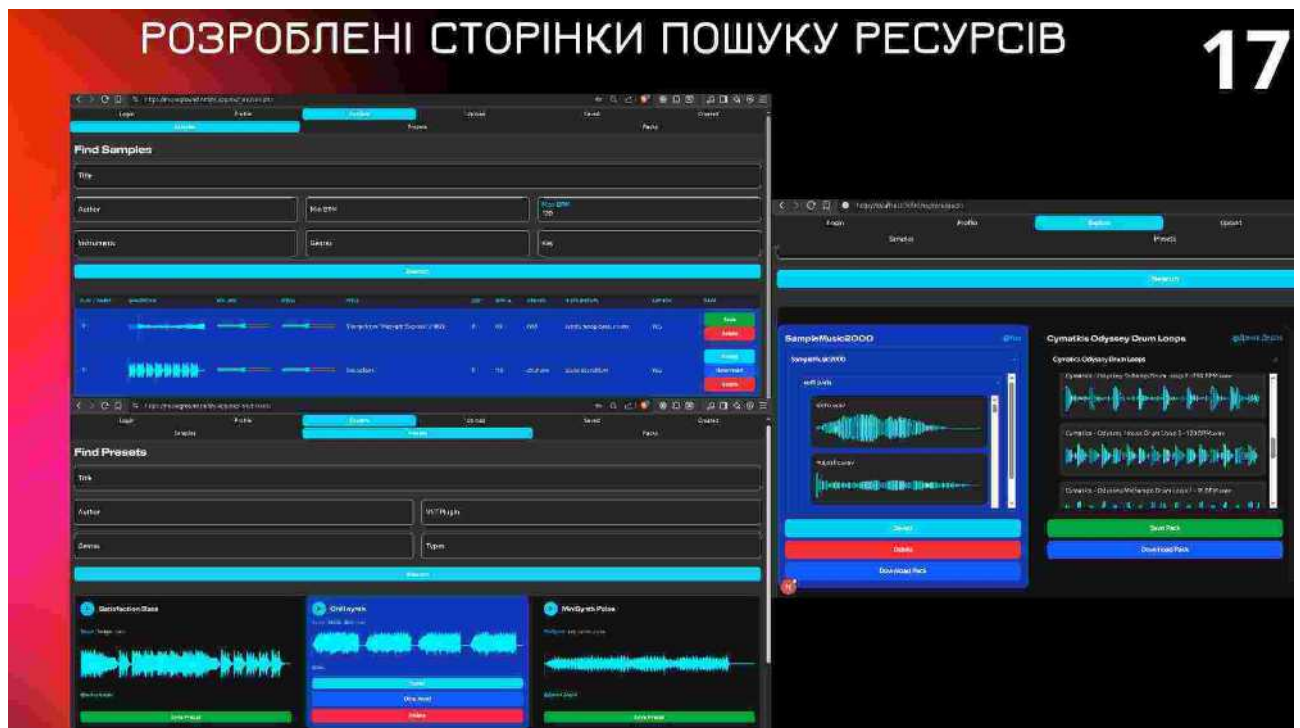


Рисунок В.17 — Розроблені сторінки пошуку ресурсів

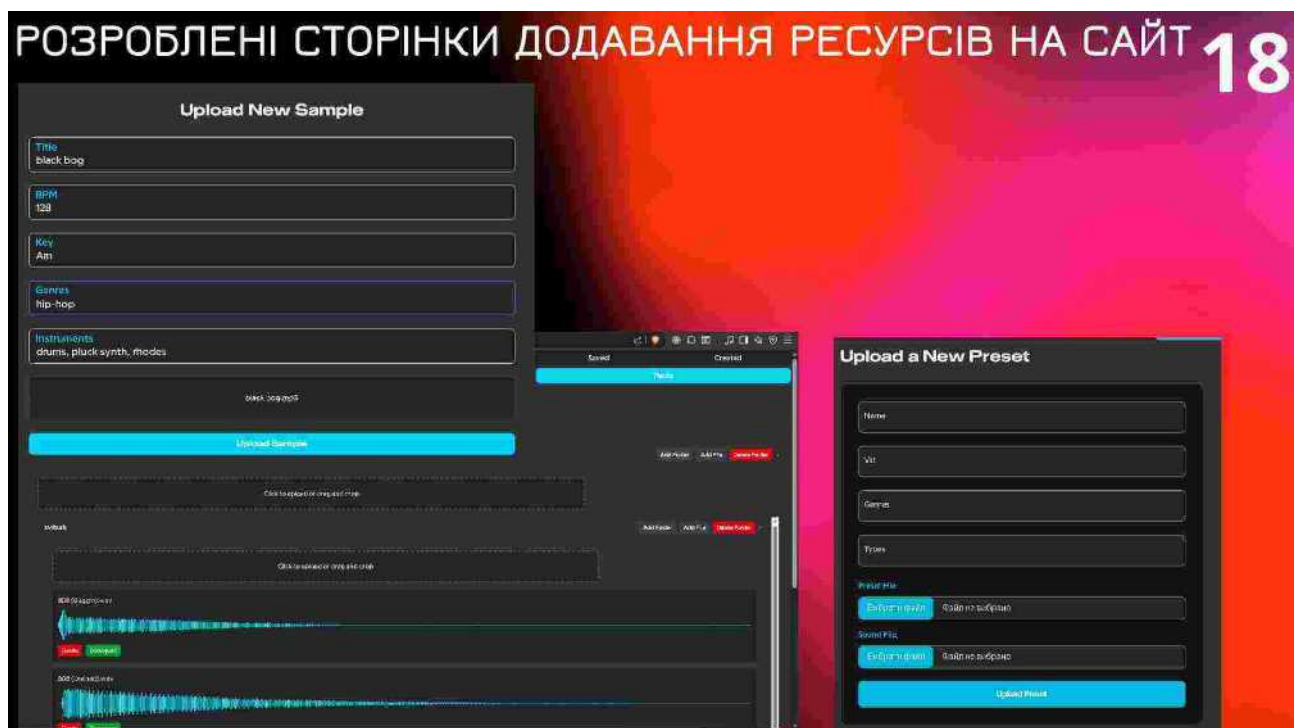


Рисунок В.18 — Розроблені сторінки додавання ресурсів на сайт

РОЗРОБЛЕНІ СТОРІНКИ РЕДАГУВАННЯ ДОДАНИХ РЕСУРСІВ 19

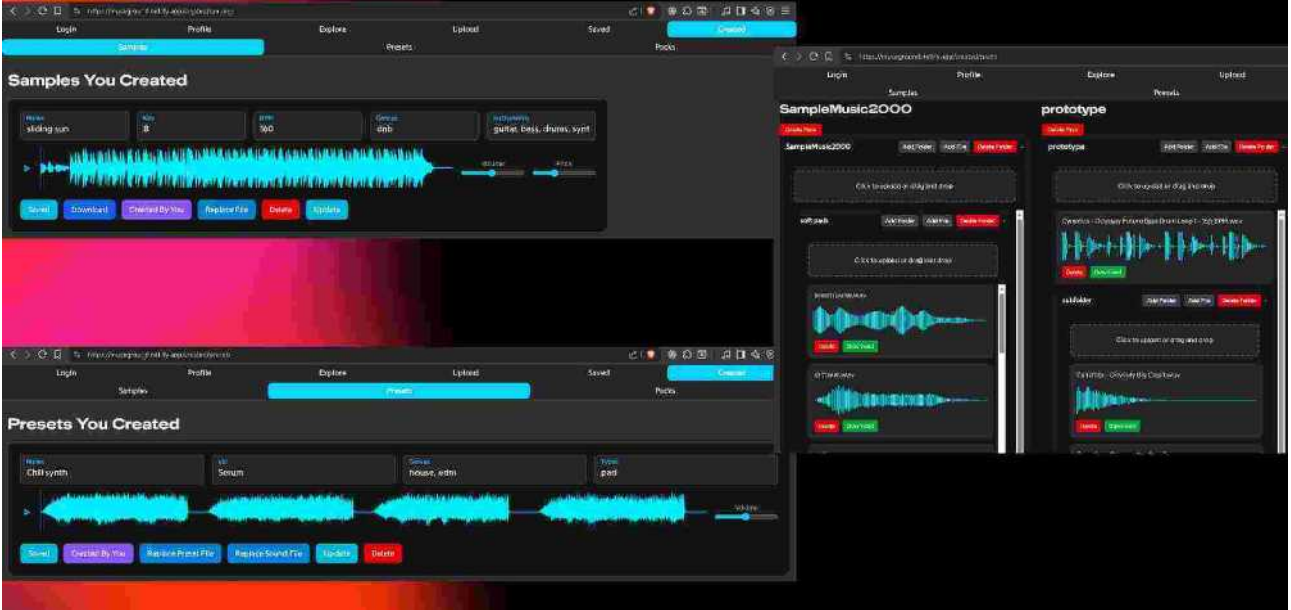


Рисунок В.19 — Розроблені сторінки редагування доданих ресурсів

ТЕСТУВАННЯ РОЗРОБЛЕНОГО ЗАСТОСУНКУ 20

ДЛЯ КЛІЄНТУ ЗАСТОСУНКА

ДЛЯ СЕРВЕРУ ЗАСТОСУНКА

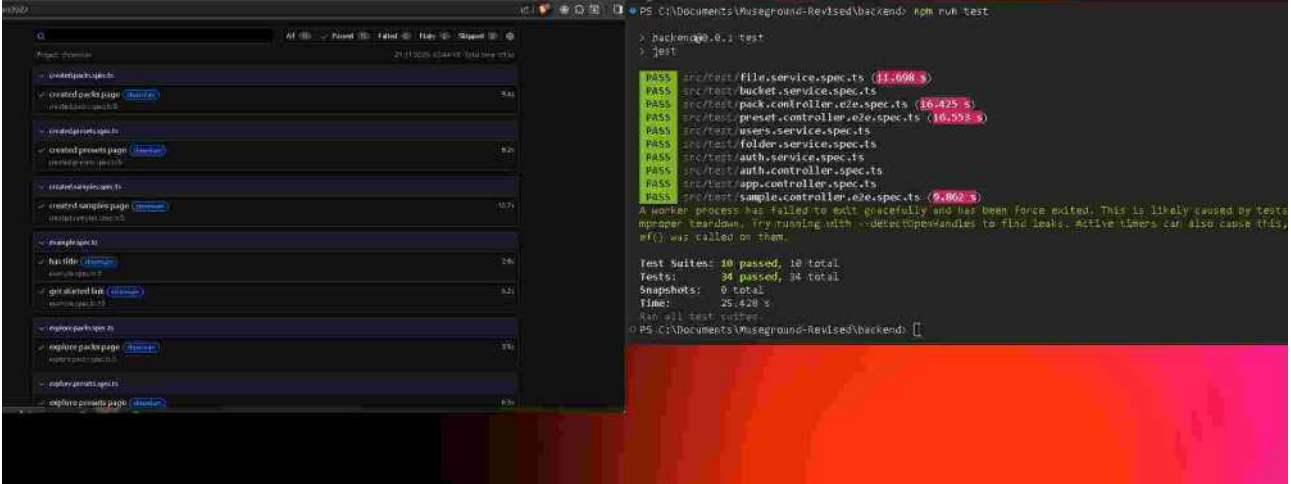


Рисунок В.20 — Тестування розробленого застосунку

ВИСНОВКИ

21

У ході виконання дипломної роботи, об'єктом дослідження якої є процес розповсюдження цифрових ресурсів для створення музики, а предметом — вебзастосунок для обміну та управління музичними ресурсами, було розроблено вебзастосунок «Museground», який надає користувачам зручні інструменти для пошуку, завантаження, збереження, редагування та обміну музичними матеріалами.

Практична цінність роботи полягає у створенні універсальної онлайн-платформи для обміну цифровими музичними ресурсами, яка може бути використана музичними продюсерами, саунд-дизайнерами та іншими учасниками музичної індустрії. Розроблений вебзастосунок може слугувати основою для подальшого розвитку системи, додавання соціальних функцій, рекомендаційних алгоритмів, інтеграції з DAW та розширення підтримки нових типів музичних ресурсів.

Отже, у результаті виконання дипломної роботи було досягнуто поставленої мети — розроблено та протестовано вебзастосунок для розповсюдження та обміну цифровими ресурсами для створення музики, який забезпечує зручний, стабільний та масштабований інструмент для роботи з музичним контентом у межах єдиної онлайн-платформи.

Рисунок В.21 - Висновки