

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 проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПЛАТФОРМА ДЛЯ ПОТОКОВОЇ ПЕРЕДАЧІ МУЗИКИ
3 ІНТЕГРАЦІЄЮ WEB-3 ТЕХНОЛОГІЙ
MUSIC-STREAMING PLATFORM WITH
WEB-3 INTEGRATION

Виконав(ла): студент(ка) 4 курсу, групи КНТ-131

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ТЕЛЕСПОВ Б.В.

(ПРИЗВИЩЕ та ініціали)

Керівник ФЕДОРОНЧАК Т.В.

(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
 “ ” 2025 року

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

ТЕЛЕСПОВА Богдана Вадимовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Платформа для потокової передачі музики з інтеграцією Web-3 технологій. Music-streaming platform with Web-3 integration.

керівник проєкту (роботи) к.т.н., доцент, ФЕДОРОНЧАК Тетяна Василівна ,

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “28” квітня 2025 року № 209

2. Строк подання студентом проєкту (роботи) 10 травня 2025 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. Опис програми; 3. Розробка програми; 4. Тестування програмного забезпечення; 5. Додатки.

5. Перелік графічного матеріалу (з 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	ФЕДОРОНЧАК Т.В., доцент		
Нормоконтроль	ДЕЙНЕНА Л.Ю., старший викладач		

7. Дата видачі завдання « 15 » квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	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)

(підпис) Богдан ТЕЛЄПОВ
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Тетяна ФЕДОРОНЧАК
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 101 с., 20 табл., 53 рис., 3 дод., 17 джерел.

CLIENT, SERVER, API-ЗАПИТИ, ФАЙЛИ, ФОРМИ, GRAPH QL, DAPP, USDC, CRYPTO, MUSIC, STREAMING, WEB-3.

Об'єкт дослідження – вебзастосунки для потокової передачі музики.

Предмет дослідження – вебзастосунок для потокової передачі музики із інтеграцією Web-3 технологій.

Мета роботи – розробка вебзастосунку для прослуховування (стримінгу) музики із інтеграцією Web-3 технологій.

Матеріали, методи та технічні засоби: WebStorm, персональний комп'ютер з процесором Ryzen7 5700u під управлінням операційної системи Linux Mint 21.

Результати. Розроблено вебзастосунок для потокової передачі аудіо файлів із інтеграцією Web-3 технологій для надання застосунку інтерактивності.

Висновки. Розроблене програмне забезпечення дозволяє прослуховувати музику із будь-якого пристрою, має зрозумілий інтерактивний інтерфейс та утилізує Web-3 технології.

Галузь використання – розваги та дозвілля, бізнес і маркетинг.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 101 pages, 20 tables, 53 images, 3 appendixes, 17 sources.

CLIENT, SERVER, API RECORDS, FILES, FORMS, QL GRAPHICS, DAPP, USDC, CRYPTO, MUSIC, STREAMING, WEB-3.

The object of consideration are web applications for streaming music.

The proposed subject is a web application for streaming music with the integration of Web-3 technology.

The goal of the work - development of web application for listening (streaming) music with the integration of Web-3 technologies.

Materials, methods and technical means: Web Storm, a personal computer with a Ryzen7 5700 processor running the Linux Mint 21 operating system.

Results. A web application for streaming music with the integration of Web-3 technology has been developed to provide interactivity features to the application.

Conclusions. The developed program allows to listen to music from any device, bringing intelligent interactive interfaces and utilizing Web-3 technologies.

Field of usage - entertainment and leisure, business and marketing.

ЗМІСТ

	С.
Abstract	5
Зміст	6
Перелік скорочень та умовних познач	8
Вступ	9
1 Аналіз предметної області	10
1.1 Огляд існуючих сервісів та платформ які виконують роль застосунку стримінгу музики	11
1.2 Стислий аналіз технічного завдання на розробку майбутньої платформи для потокової передачі музики	14
1.3 Висновки з розділу і постановівка задачі	14
2 Опис програми	15
2.1 Огляд існуючих методів вирішення завдання	15
2.2 Огляд існуючих інструментів вирішення завдання	16
2.3 Огляд технічного завдання	17
2.4 Висновки з розділу	22
3 Розробка програми	23
3.2 Основні рішення щодо розроблених модулів серверної частини	29
3.2 Основні рішення щодо розробки інтерфейсу	50
3.3 Основні рішення щодо розробки бази даних	45
3.4 Основні рішення щодо розроблених алгоритмів	54
3.5 Висновки з розділу	55
4 Тестування програмного забезпечення	57
4.1 Тестування клієнтської частини	57
4.1.1 Взаємодія із програмним забезпеченням	58
4.2 Тестування серверної частини	70
4.3 Тестування частини смарт-контрактів	71
4.4 Висновки з розділу	72
Висновки	74

Перелік джерел посилання	75
Додаток А Технічне завдання	77
Додаток Б Код програми (основні компоненти)	83
Додаток В Слайди презентації	92

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

JSX/TSX	– JavaScript XML / TypeScript XML;
GQL	– Graph Query Language;
ОС	– операційна система;
ПК	– персональний комп'ютер;
JWT	– Json Web Token;
ПЗ	– програмне забезпечення;
БД	– база даних;
JS	– JavaScript;
TS	– TypeScript;
IPFS	– Interplanetary File System.

ВСТУП

Більшу частину часу користувачі витрачають на пошук, організацію та прослуховування музики, а також управління власними плейлістами та музичними бібліотеками. Майже всі існуючі платформи для стримінгу музики зазвичай мають перевантажені інтерфейси що не відповідають більшій частині користувачів і відволікають від зручного управління медіатекою. Також, при зміні платформ (з десктопної на мобільну) користувачу кожен раз необхідно звикати до нового інтерфейсу, що також може затримувати взаємодію із медіа контентом.

Дуже цікавою темою сьогодення є розвиток Web-3 технологій, впровадження їх у більшість застосунків та програмних рішень. Вони відкривають нові можливості для взаємодії із користувачами та їх даними, а саме: децентралізоване зберігання, володіння цифровим контентом та надання інтерактивного досвіду користування застосунками, що утилізують ці технології.

Актуальною є задача створення стримінгового сервісу музики із Web-3 технологіями, який би у результаті відповідав всім вимогам сучасності, зручності використання та функціональності. Така платформа має забезпечити швидкий доступ до аудіоконтенту і надати розширений функціонал використання, з використанням децентралізованих технологій. У такому варіанті, підвищиться користувацький досвід, розшириться функціональність звичайного стримінгового сервісу музики та збільшиться кількість активних учасників Web-3 екосистеми.

Все вищезазначене надасть змогу побудувати нову модель поширення та споживання медіаконтенту, із наданням великої уваги на прозорість, безпеку даних і децентралізацію.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Вебтехнології із кожним днем непереривно розвиваються і стають все більш популярними. Майже кожна людина не може представити своє повсякдення без певних програм, які доступні тільки через браузер, щомісяця з'являються нові проекти, які покращують наше життя, роблячи його легшим і зручнішим.

Інтерфейс вебзастосунку зазвичай складається із модальних вікон, хедерів, футерів, банерів, меню тощо, що надають увесь можливий функціонал для взаємодії із системою.

Існуючі рішення по потоковій передачі (стримінгу) музики надають увесь необхідний функціонал для вирішення задачі саме прослуховування, не зважаючи на інші аспекти і можливі покращення в сторону інтерактивності взаємодії із програмою. Саме це і є основним недоліком цих програм.

Іншим, можливо не для всіх недоліком є переобтяженість інтерфейсу, або його недостатня інформативність, через спробу вмістити велику кількість даних на одній видимій сторінці застосунку, розробникам необхідно іти на певні компроміси, що можуть негативно вплинути на зручність використання, тим самим заплутуючи потенційно нового користувача. У іншому випадку інтерфейс займає багато місця для наданої інформації.

Для розробки вебзастосунку для прослуховування музики можна використати NextJS - для створення інтерактивного, зрозумілого та не переобтяженого інтерфейсу, та NestJS - для серверної частини. Під час розробки необхідно забезпечити можливість користувачам виконувати основні функції програми, а саме: відтворення треків, створення і редагування плейлистів, збереження треків, вивантаження треків. Також важливо врахувати дизайн інтерфейсу і серверну архітектуру для зручності та легкості використання та підтримки майбутнього програмного забезпечення.

1.1 Огляд існуючих сервісів та платформ які виконують роль застосунку стримінгу музики

1.1.2 Платформа “AppleMusic” (web)

Apple Music [1] - це сервіс для потокового прослуховування музики, розроблений компанією Apple. Користувачі мають змогу самі вибирати музику, яку бажають слухати на своєму пристрої. Сервіс включає модеровану радіостанцію Beats 1, блоґо-платформу Connect, яка дозволяє музикантам розміщувати в ній свої фото, дописи, відео та музику, та Radio, яке включає деякі з функцій iTunes Radio. Apple Music пропонує підказки та рекомендації, що базуються на вподобаннях слухача, а також має інтеграцію з голосовими командами Siri.

Перевагою даного застосунку є висока якість аудіо, але і в даному випадку не існує безкоштовної версії.

Роботу програми продемонстровано на рис. 1.1.

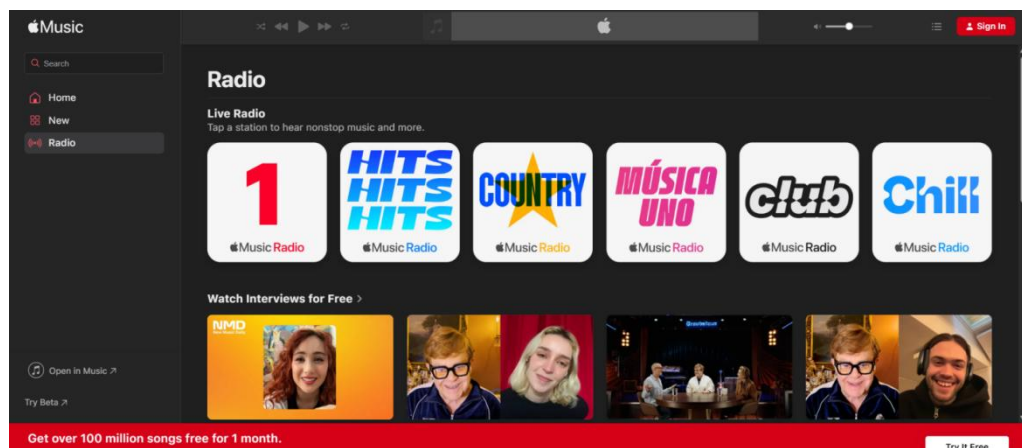


Рисунок 1.1 - Apple Music

1.1.3 Платформа “Spotify”

Сп’отіфай, також Сп’отіфай [2] - стримінговий сервіс потокового аудіо, що дозволяє слухати музичні композиції та подкасти. Надає послуги легального онлайнового стримінгу аудіозаписів основних світових і

незалежних лейблів, зокрема BBC, Sony, EMI, Warner Music Group та Universal. Запущений у жовтні 2008 року шведським стартапом «Spotify AB»

Перевага - наявність безкоштовної версії застосунку, недоліків не виявлено.

Роботу програми продемонстровано на рис. 1.2.

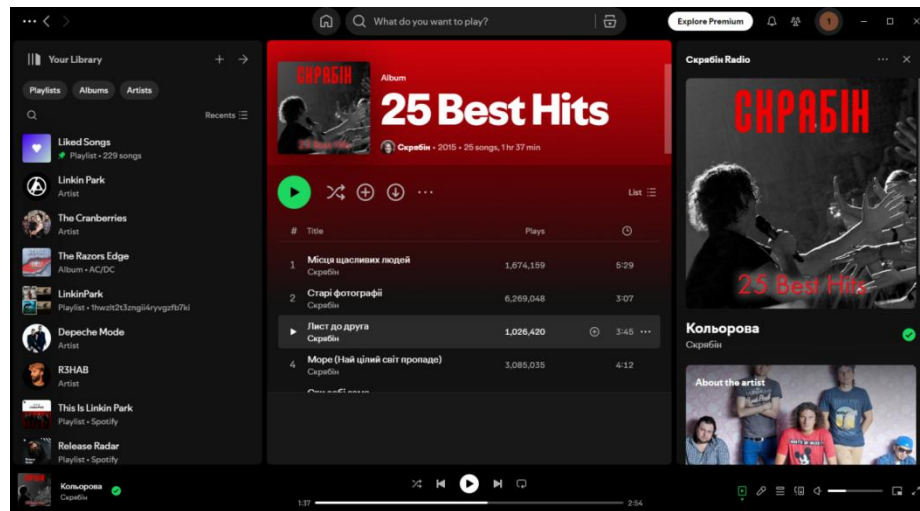


Рисунок 1.2 - Spotify

1.1.4 Платформа “SoundCloud”

SoundCloud [3] - стримінгова платформа та вебсайт для розповсюдження оцифрованої звукової інформації (наприклад, музичних творів), володіє функціями соціальної мережі та однойменної організації.

Перевагою є наявність безкоштовної версії застосунку, недоліком - переобтяжений інтерфейс.

Роботу програми продемонстровано на рис. 1.3.

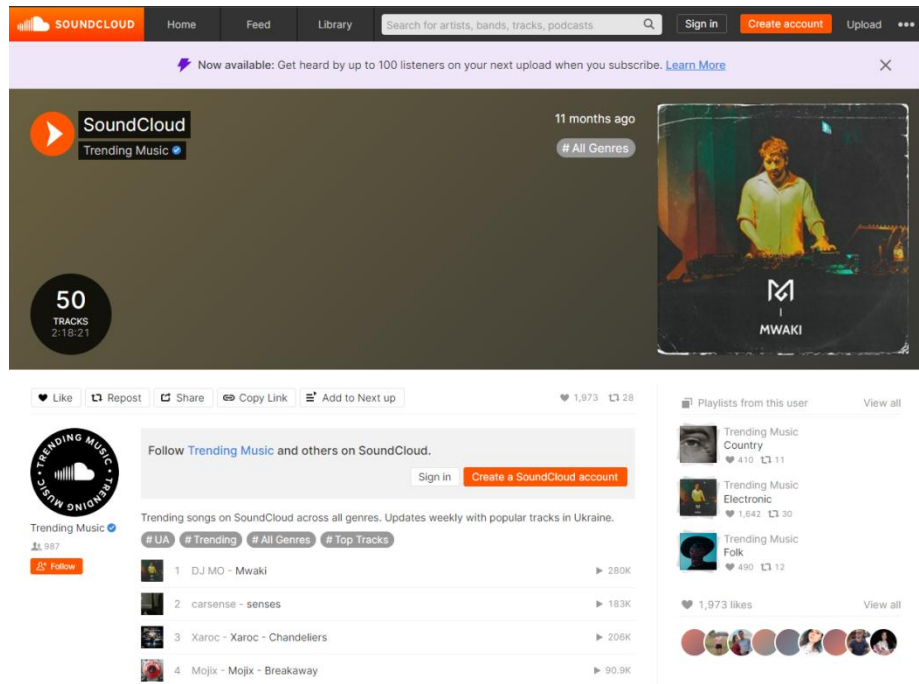


Рисунок 1.3 - SoundCloud

Аналіз аналогічних програм буде проводитися за такими параметрами:

- зручність інтерфейсу (0-5);
- кросплатформність (0-1);
- тип програми (free-paid);
- розмір програми (MB);
- актуальність інтерфейсу (0-5);

Аналіз програм наведено у табл. 1.1.

Таблиця 1.1 - Порівняння програм

Назва	Зручність	Кросплатформність	Тип підписки	Розмір	Актуальність інтерфейсу
Apple Music	4	1	paid	> 300MB	4
Spotify	4	1	free	> 100MB	5
SoundCloud	3	1	free	-	3

За результатами аналізу програмного забезпечення, можна зробити висновок, що певні програмні продукти мають не дуже зручний та не актуальний інтерфейс, але всі з них є кросплатформними рішеннями.

1.2 Стислий аналіз технічного завдання на розробку майбутньої платформи для потокової передачі музики

Загальний опис:

Розробити платформу для стримінгу музики із Web-3 інтеграціями для надання більшої інтерактивності застосунку.

Інтерфейс:

Забезпечити високу зручність користування та інтуїтивно зрозумілий інтерфейс.

Забезпечити актуальність дизайну.

Кросплатформність:

Реалізувати застосунок так, щоб він міг бути доступний у різних браузерах.

Функціональність:

Забезпечити повний набір базових функцій сервісу для потокової передачі музики (прослуховування треків, вивантаження треків, збереження треків, оперування плейлистами).

1.3 Висновки з розділу і постановка задачі

У даному розділі було проведено аналіз вже існуючих рішень, перелічено їх переваги та недоліки.

В результаті аналізу було визначено функціональні вимоги до розроблюваного програмного забезпечення для стримінгу аудіо з інтеграцією web-3 технологій, з метою надання застосунку інтерактивності. Повне технічне завдання наведено у додатку А.

2 ОПИС ПРОГРАМИ

В даному розділі буде розглянуто технічне завдання та основні особливості програмних засобів, за допомогою яких було розроблено вебплатформу для потокової передачі музики.

2.1 Огляд існуючих методів вирішення завдання

У роботі всього застосунку використана концепція використання middleware, та усіх інших допоміжних модулів NodeJS [4].

Концепцію роботи ExpressJS [5]-[6], як основної бібліотеки на якій розроблено фреймворк NestJS [7] показано на рис. 2.1.

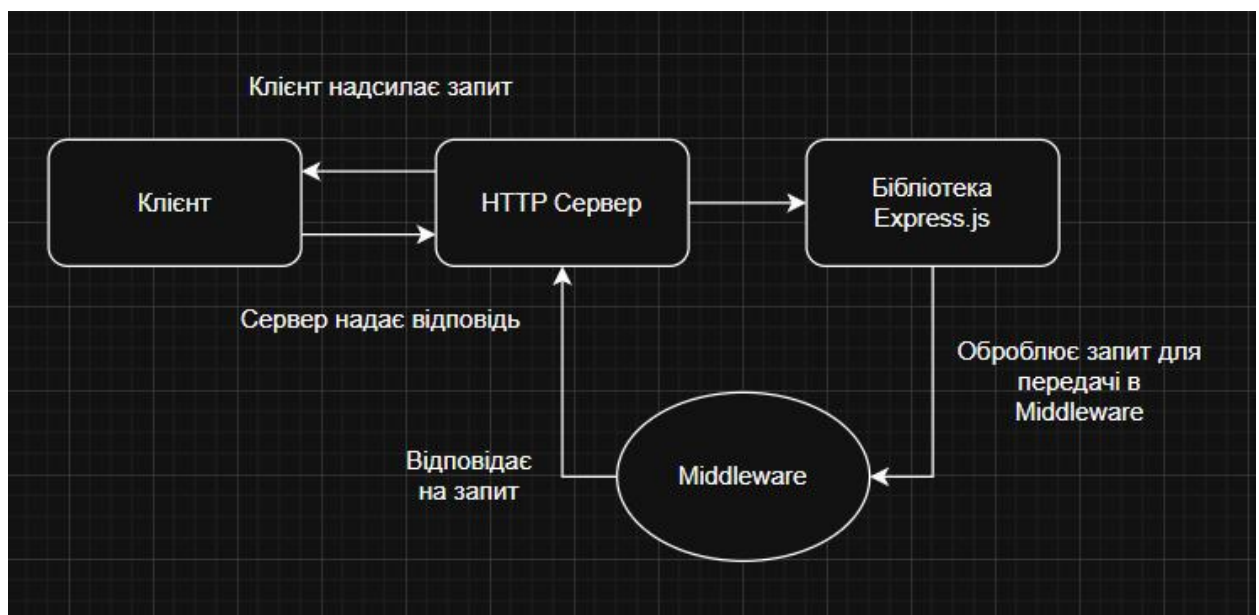


Рисунок 2.1 - ExpressJS в NestJS та NodeJS

На зображенні наведеному вище приведено принцип взаємодії клієнтської частини із серверною, використовуючи middleware [8] на стороні сервера, що спрощує реалізацію логіки для відповідного функціонування програмного забезпечення.

Було обрано технологію із концепцією такого типу, для надання переваг наступним можливостям фінального застосунку:

- розширюваність - middleware [8] дозволяє додавати функціональність до застосунку шляхом додавання або модифікації middleware-компонентів (модулям у NestJS). Це забезпечує легку розширюваність і зручність у внесенні змін;
- обробка запитів - технологія дозволяє реалізувати логіку маршрутизації запитів і їх подальшої обробки (контроллерам у NestJS);
- послідовність обробки - middleware викликається по порядку додавання їх до стеку, що дозволяє легко контролювати порядок обробки запиту та відповіді.

Загалом, використання фреймворку який утилізує відповідну технологію, зробить фінальний застосунок більш модульним і зручним для змін.

2.2 Огляд існуючих інструментів вирішення завдання

Коли говорять про NextJS [9] + NestJS + TailwindCSS, вказують на перелік технологій які є частиною великої екосистеми JavaScript та TypeScript для розробки web-додатків.

Було обрано даний стек технологій для відносно простої реалізації усього необхідного функціоналу майбутнього ПЗ - платформи для потокової передачі музики із інтеграцією із Web-3 технологіями.

NextJS - фреймворк для розробки клієнтської частини платформи, в основі якого лежить бібліотека для розробки інтерактивних інтерфейсів ReactJS [10], із використанням JSX [11] синтаксису.

NestJS - фреймворк для розробки серверної частини застосунку, побудований на бібліотеці ExpressJS на платформі NodeJS.

TailwindCSS - бібліотека стилів, що використовується для створення і стилізації легких і інтуїтивно зрозумілих інтерфейсів користувача. Дана

бібліотека надає велику кількість готових рішень щодо використання CSS стилів у застосунках подібного типу.

Технології, що перелічені вище надають велику швидкість розробки програмного продукту та високий рівень модульності і кросплатформності, роблячи їх потужним інструментом для створення, підтримки і розвитку вебпроектів.

2.3 Огляд технічного завдання

2.3.1 Аналіз вимог до функціоналу

Згідно технічного завдання майбутнє програмне забезпечення повинне надавати користувачу базовий перелік функцій для управління їх музичними постами (треками), плейлистами і т.д.

До таких функцій відносяться програвання трека, завантаження трека, додавання та видалення з плейлиста, створення і видалення власних плейлистів. Більше того, у платформі має бути можливість відвідувати профіль користувачів і модифікувати свій, наприклад змінювати ім'я та фонове зображення. Також на платформі мають бути доступними базові способи авторизації, створення та видалення акаунта. Додатково, всім користувачам необхідно надати можливість писати в умовну службу підтримки, використовуючи спеціальні форми зворотнього зв'язку і скаржитись на завантажені треки іншими користувачами для підтримки чистоти і якості розроблюваного програмного забезпечення.

У платформі необхідно реалізувати музичні змагання між користувачами утилізуючи блокчейн.

Основною ідеєю є надання користувачу можливості створювати змагання між двома треками різних користувачів, обов'язково включаючи трек користувача що є ініціатором змагання, додатково реалізувати функціонал голосування за обраний трек через крипто-ассети (USDC).

Після створення змагання, обом учасникам змагання необхідно надіслати повідомлення в акаунт, про старт відповідної події.

Переможцем змагання буде той, чий трек набере більше голосів.

Діаграми прецедентів базових функцій платформи наведено нижче на рис. 2.2 - 2.5.

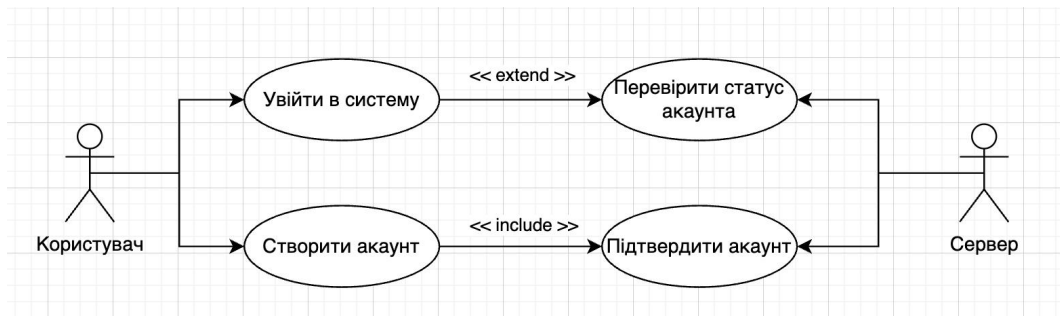


Рисунок 2.2 - Діаграма прецедентів про авторизацію в платформі

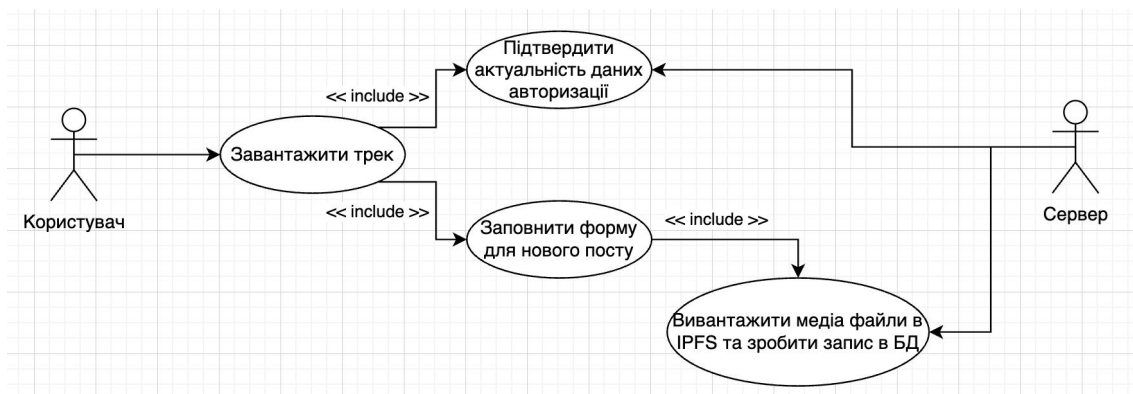


Рисунок 2.3 - Діаграма прецедентів про завантаження нового трека

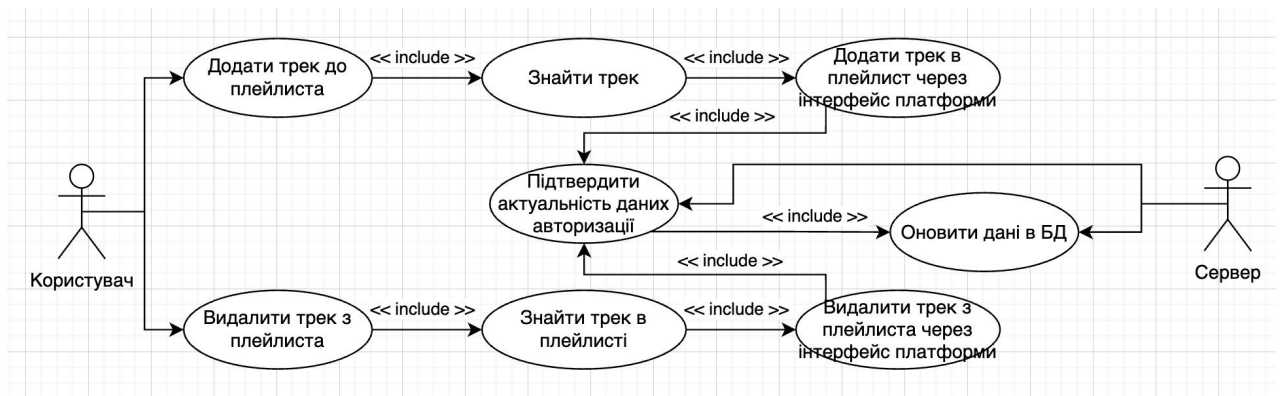


Рисунок 2.4 - Діаграма прецедентів про додавання та видалення треку з плейлиста

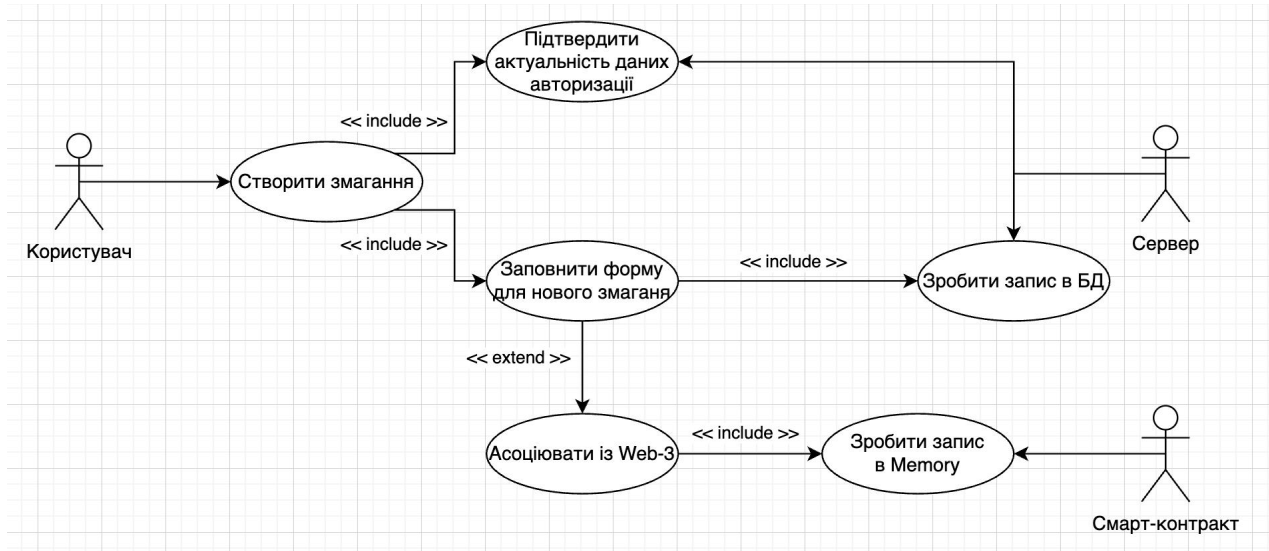


Рисунок 2.5 - Діаграма прецедентів про створення змагання і можливою інтеграцією новоствореного змагання із Web-3

Діаграму взаємодії користувача із смарт-контрактом при створенні змагання через майбутній інтерфейс платформи наведено на рис. 2.6.

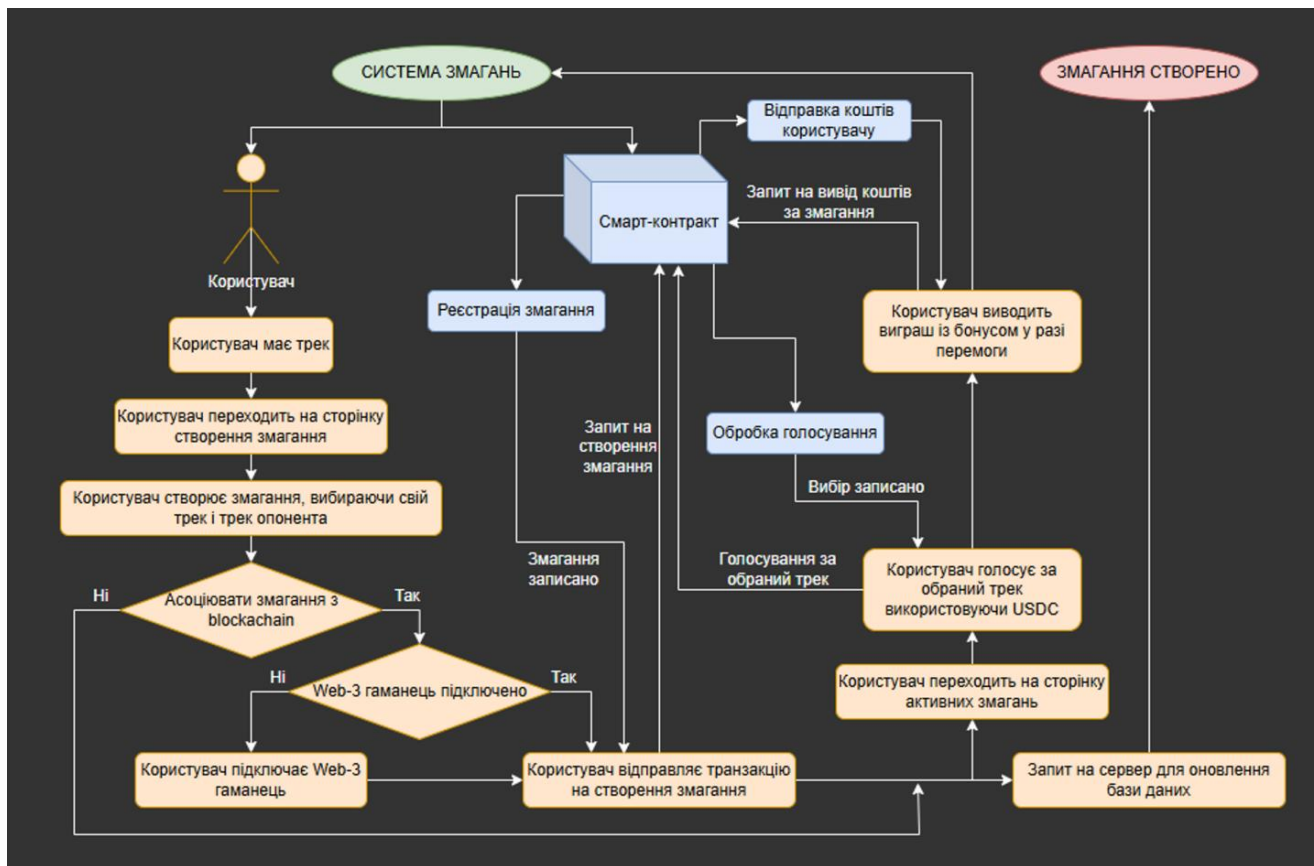


Рисунок 2.6 - Взаємодія користувача із Web-3

Окрім того, необхідно впровадити систему рівнів профілів користувача, відображаючи досягнення в їх особистому та публічному профілях. Досягнення не надаватимуть жодних переваг перед іншими користувачами.

Також, програма повинна мати можливість оброблювати виключні ситуації під час взаємодії із користувачем.

Через те що у результаті необхідно отримати застосунок (платформу) типу клієнт-сервер, необхідно розробити відповідну логіку на стороні клієнта і сервера (за допомогою HTTP та HTTP+GraphQL [12] запитів).

2.3.2 Стратегія виконання

Оперуючи технічним завданням і обраною мовою програмування, найдоречнішою комбінацією буде NestJS для серверної частини + NextJS на клієнтській стороні. Такий вибір було зроблено виходячи із безпеки і можливості швидкого розширення майбутнього програмного забезпечення.

Першим етапом буде реалізація усього необхідного функціоналу на серверній частині, а саме створення необхідних модулів, підключення бібліотек і реалізації RestAPI із GraphQL обробниками та проектування структури бази даних.

Другим етапом буде реалізація верстки на клієнтській частині, задання структури усіх можливих сторінок NextJS застосунку.

Третім етапом буде зв'язування серверної (NestJS) і клієнтської (NextJS) частин шляхом створення запитів на стороні клієнта і додавання усього функціоналу для майбутньої взаємодії користувача із платформою, включаючи завантаження та читання з IPFS [13] сховища.

Четвертим етапом буде розробка смарт-контрактів для реалізації змагань між користувачами утилізуючи можливості блокчейн.

Виходячи з вимог технічного завдання, схему функціонування майбутнього застосунку із використанням усіх допоміжних бібліотек наведено нижче на рис. 2.7.

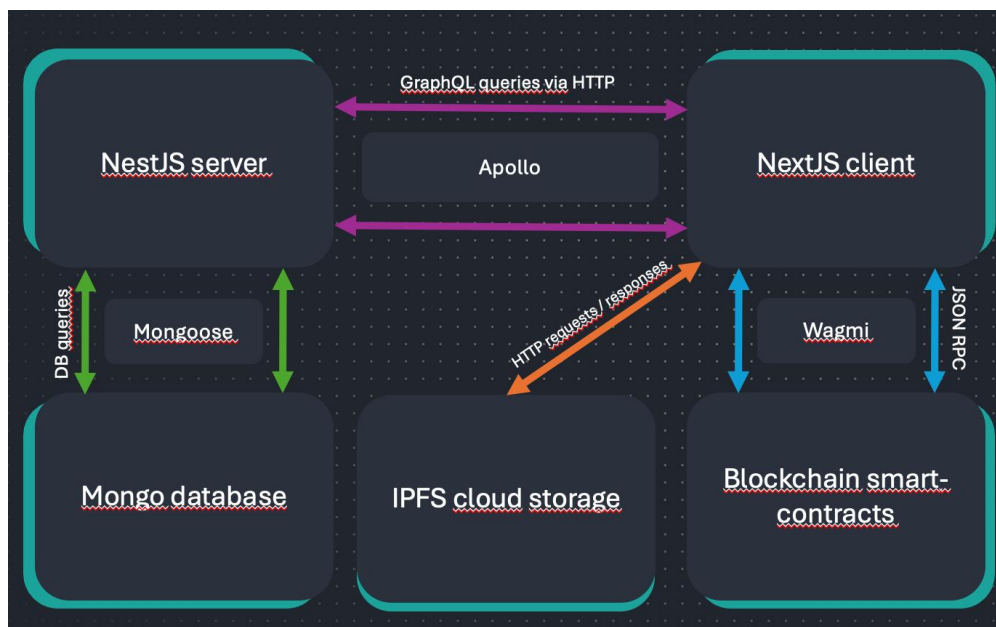


Рисунок 2.7 - Схема функціонування майбутнього програмного забезпечення

2.3.3 Вірогідність ризиків

Виходячи з того що у проекті буде використано добре протестовані фреймворки та бібліотеки - вірогідність появи будь-яких ризиків, окрім ризику затрати часу буде мінімальною.

Беручи до уваги те що реалізація майбутньої платформи буде дотримуватись паттерну MVC (Model-View-Controller), можна буде без будь-яких складностей змінити та оновити незалежні компоненти системи.

Завдяки використанню відповідних технологій і паттерну розробки, розроблюване програмне забезпечення набуде таких позитивних якостей: модульність, гнучкість, масштабованість.

2.4 Огляд особливостей мови програмування

TypeScript (TS) - мова програмування, що призначена для створення web-застосунків, в цілому є обгорткою до JavaScript. Основні характеристики визначені стандартом ECMAScript.

TS має строгу типізацію, завдяки чому під час виконання програми не має можливості зміни типу змінних, тим самим покращуючи процес розробки. Синтаксис мови подібний до C#, Java та C++.

TS підтримує функціональне, об'єктно-орієнтоване та асинхронне програмування завдяки використанню promise-ів із async+await, зазвичай використовується для розробки фронтенду web-застосунку і його серверної частини на платформі NodeJS.

2.4 Висновки з розділу

У цьому розділі було розібрано технічне завдання майбутнього програмного забезпечення та обгрунтовано засоби, використані для розробки. Викладено стратегію розробки і проаналізовано вимоги до функціоналу. Визначено високорівневу архітектуру застосунку.

3 РОЗРОБКА ПРОГРАМИ

В даному розділі будуть розглянуті та описані основні рішення з реалізації компонентів системи, як основних та невід’ємних компонентів програми.

3.1 Огляд класової ієрархії

Програмне забезпечення складається із трьох компонентів:

- серверна частина - частина що відповідає за більшу частину логіки розроблюваної платформи;
- клієнтська частина - відповідає за взаємодію із фінальними користувачами;
- частина смарт-контрактів - реалізує логіку одного з компонентів системи, утилізуючи блокчейн.

3.1.1 Серверна частина

Серверна частина побудована на фреймворку NestJS і задовільняє структурі більшості проєктів побудованих на такому ж рішенні. Усі системні компоненти розбиті на модулі, кожен з яких відповідає за свою частину функціональності програмного забезпечення. Усю необхідну логіку по роботі із застосунком розміщено у директорії “entities”.

Структура серверної частини:

- /mfnServer - корінь;
- /node_modules - залежності застосунку;
- /src - директорія у якій знаходяться майже всі власноруч створені модулі;
- /auth - директорія що тримає модулі, які відповідають за аутентифікацію / авторизацію користувача;

- /common - розташування залежностей модулів що більше всього перевикористовуються;
- /entities - усі можливі сутності проєкту (трек, користувач, і т.д.);
- /graphql - ініціалізація graphql;
- /utils - допоміжні компоненти;
- app.module.ts - головний модуль (імпортує всі модулі в себе);
- main.ts - точка входу;
- package.json - список залежностей і скриптів.

Структуру серверної частини платформи наведено на рис. 3.1.

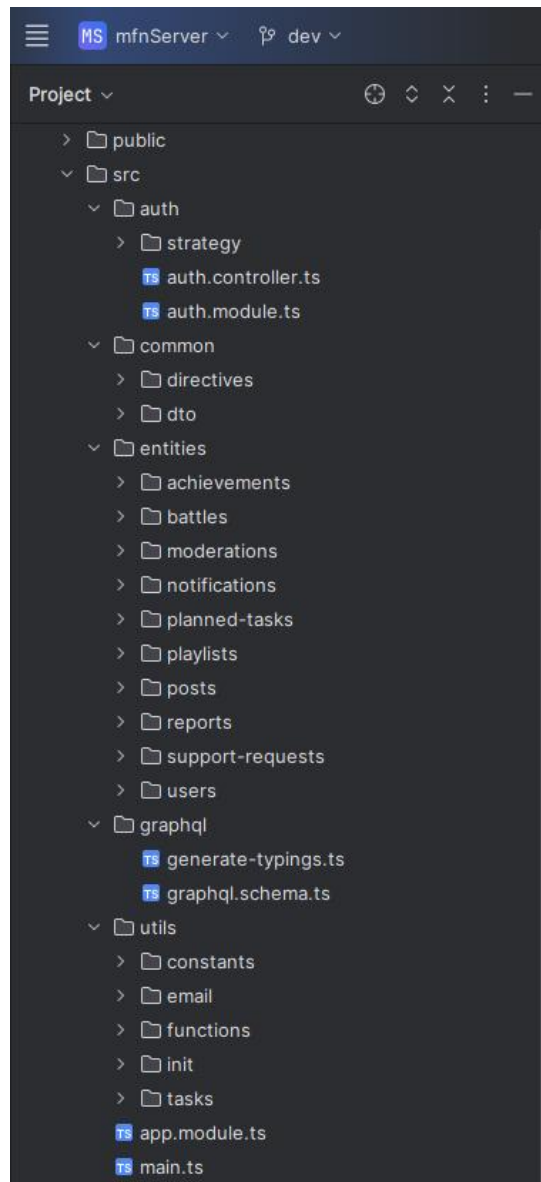


Рисунок 3.1 - Структура серверної частини

Структуру використання модулів наведено на рис. 3.2

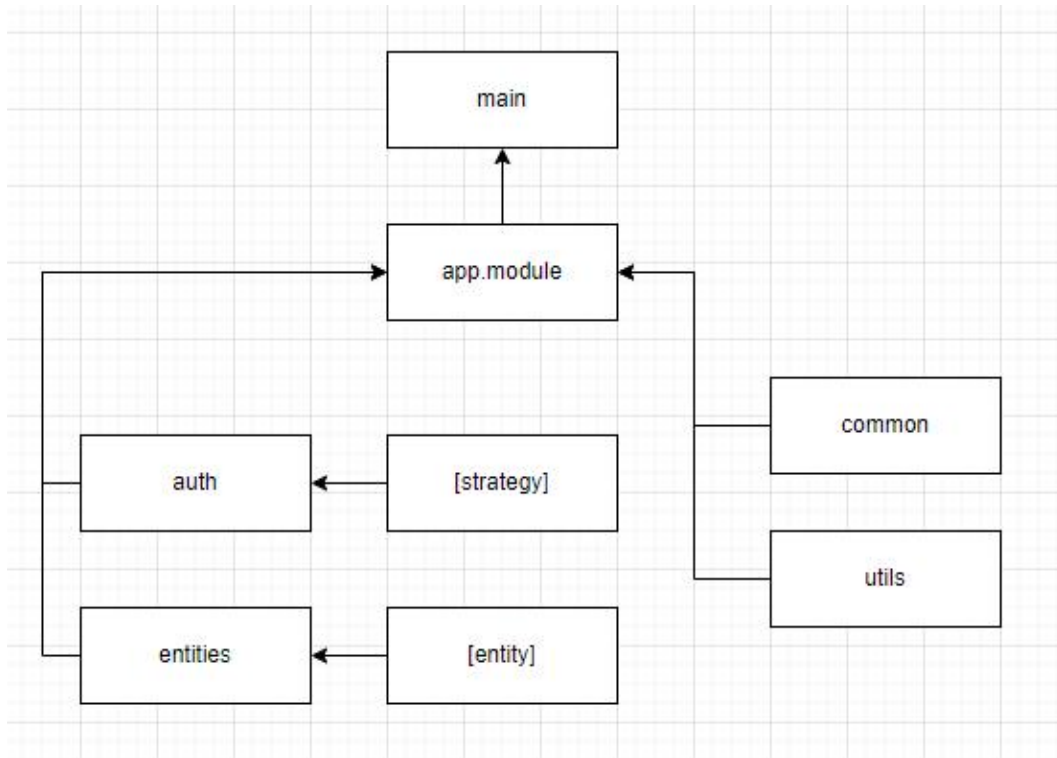


Рисунок 3.2 - Структура використання модулів

На рисунку наведеному вище зображена ієрархія використання модулів NestJS застосунку, [модуль] - означає те що модулів такого типу більше 2х, головним модулем є “app.module.ts”, точка входу - “main.ts”. Обробка помилок реалізована усередині кожного модуля, серверна частина не потребує глобальної обробки помилок.

3.1.2 Клієнтська частина

Клієнтську частину розділено на файли, кожен з яких відповідає за частинку інтерфейсу тої чи іншої сторінки, сторінки розташовані у `src->app->[lang]->сторінка`. Кожна сторінка складається із компонентів (components). Структура клієнтської частини відповідає звичайній структурі NextJS проекту.

Структура клієнтської частини:

- /mfnClient;
- /node_modules - залежності застосунку;
- /src - головна директорія застосунку;
- /actions - директорія що містить у собі функції, призначені для виклику NextJS-серверних подій (наприклад ре-рендер сторінки);
- /app - сторінки застосунку;
- /components - компоненти із яких складаються усі сторінки;
- /config - конфігурація застосунку;
- /lib - налаштування сторонніх бібліотек;
- /types - глобільні типи для TypeScript;
- /utils - функції-утіліти що використовуються в компонентах;
- middleware.ts - логіка що буде виконуватись перед тим як користувач перейде на сторінку;
- package.json - список залежностей і скриптів.

Структуру компоновки клієнтської частини наведено нижче на рис.

3.3.

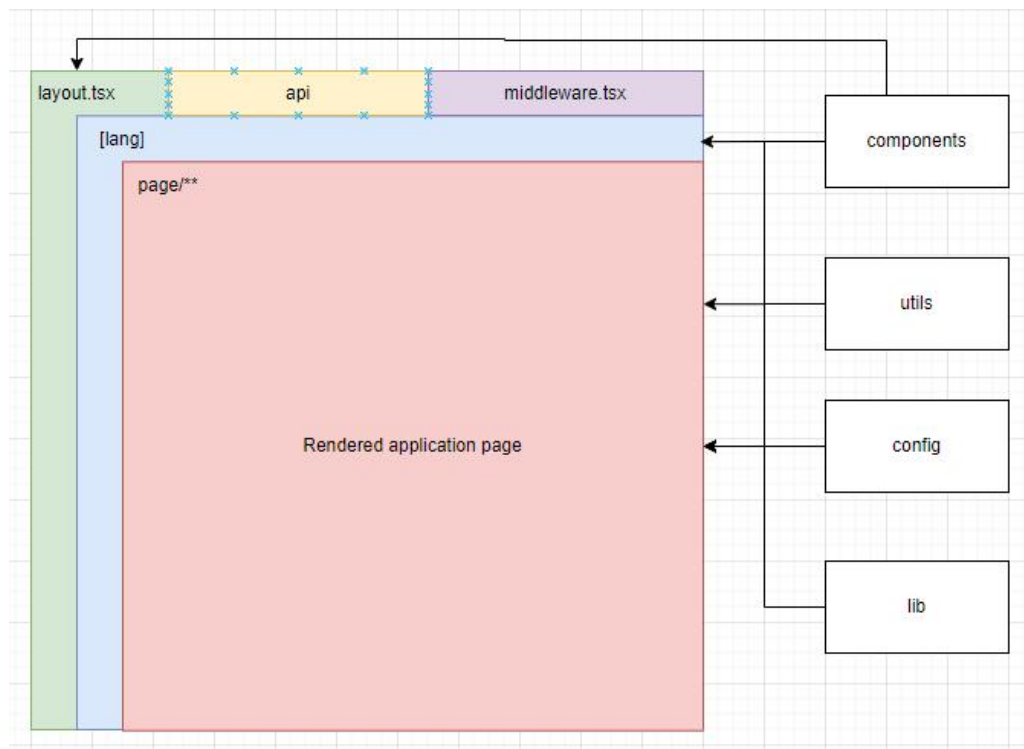


Рисунок 3.3 – Структура клієнтської (NextJS) частини

Структуру клієнтської частини зображено на рис. 3.4.

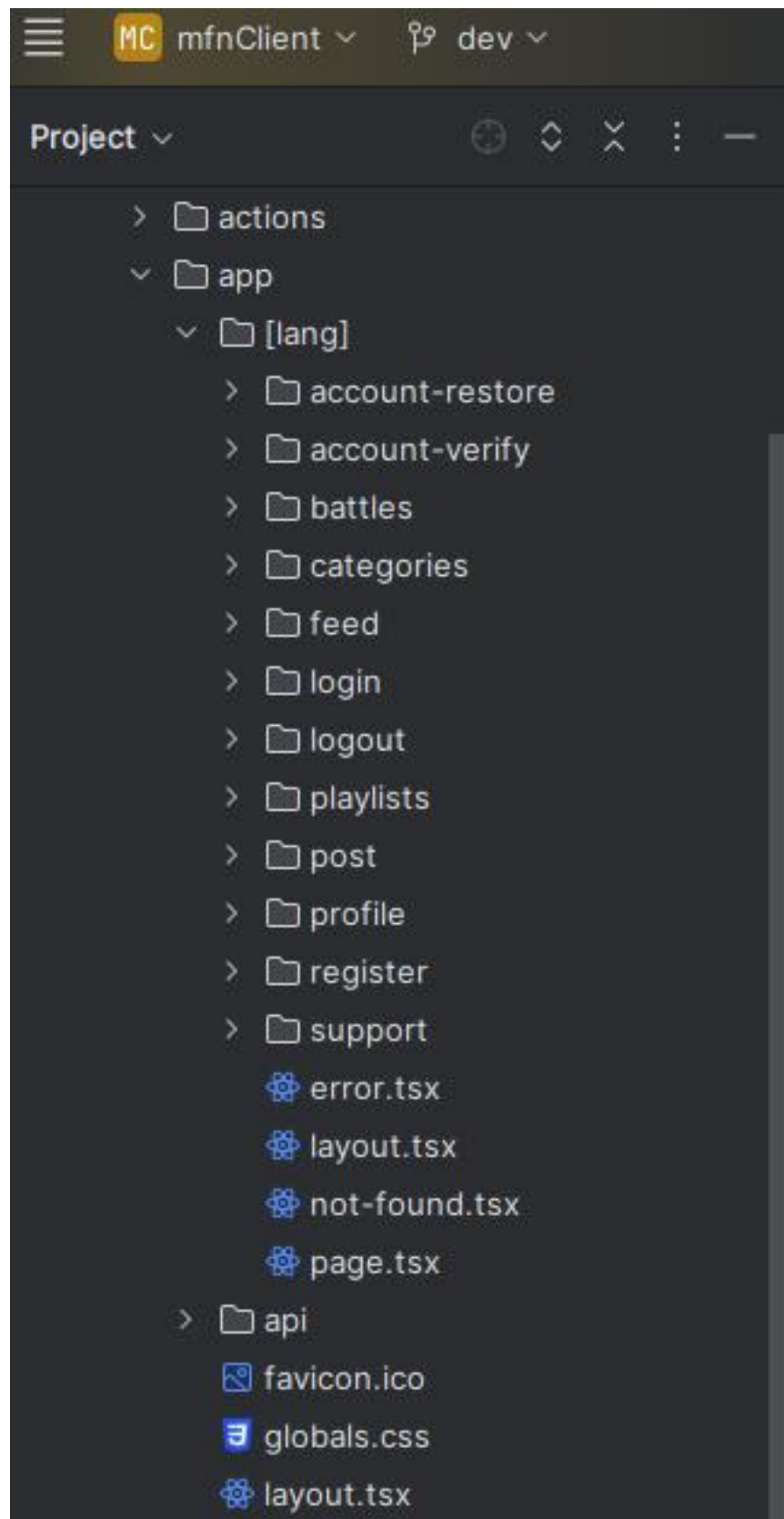


Рисунок 3.4 - Структура клієнтської частини

На рисунку вище зображено принцип роботи NextJS застосунку, точкою входу є файл “layout.tsx”, api - запити до серверу (на якому запущено NextJS проект), middleware - тип файлу що виконується завжди перед початком рендеру сторінки (служить для перевірки на валідність користувача). Components - директорія що містить у собі всі компоненти із яких будується сторінка. Lib - сторонні бібліотеки (Redux - для зберігання глобального стейту програми і Apollo client для відправки GraphQL запитів). Config - всі налаштування для клієнтської частини.

3.1.3 Частина смарт-контрактів

Оскільки застосунок взаємодіє із Web-3 і виконує певну логіку утілізуючи blockchain, програмний застосунок має в комплекті умовний смарт-контракт (програма, яка працює на блокчейні Ethereum, щоб автономно проводити, перевіряти чи здійснювати надійні транзакції), що вивантажено у мережу для подальшої взаємодії із ним через клієнтську частину розроблюваного програмного забезпечення.

Структуру частини застосунку для побудови смарт-контрактів:

- /mfnContracts;
- /node_modules - залежності;
- /artifacts - зкомпільовані дані контрактів;
- /contracts - контракти;
- /scripts - скрипти для взаємодії із контрактами прямо із середовища розробки;
- /test - модульні тести для контрактів;
- package.json - список залежностей і скриптів;
- hardhat.config.ts - конфіг бібліотеки, через яку ведеться розробка контрактів.

3.2 Основні рішення щодо розроблених модулів серверної частини

3.2.1 Точка входу серверної частини

Точкою входу до серверної частини є файл `main.ts` - відповідає за запуск та ініціалізацію основного модуля “`app.module.ts`”.

3.2.2 Головний модуль серверної частини

Головним модулем є “`app.module.ts`”, він відповідає за ініціалізацію та імпорт усіх інших модулів програмного продукту.

3.2.3 Модулі взаємодії із базою даних

У цьому підрозділі будуть розглянуті основні модулі та їх структура, що реалізують взаємодію застосунку із базою даних, зазвичай вони складаються з:

- модуля-ініціалізатора - `[part].module.ts`;
- логіки - `[part].service.ts`;
- контролера - `[part].controller.ts`, у нашому випадку `[part].resolver.ts` - для GraphQL;
- структури для бази даних - `[part].schema.ts`;
- схеми для GraphQL - `[part].graphql`.

Кожен модуль по взаємодії із БД має також директорію із модульними тестами і деякими типами даних, що буде приймати `resolver`.

Переліки цих модулів із описом всіх функцій наведено нижче.

Кожна GraphQL схема має чітко визначені “пункти”, що мають міститись у кожному подібному файлі:

- `mutation` - схема запитів (функцій) на модифікацію даних;
- `query` - схема запитів (функцій) для запиту, читання даних;

Src/entities/achievements реалізує логіку зберігання умовних досягнень користувачів.

Основні функції даного модуля наведено у табл. 3.1.

Таблиця 3.1 - Функції модуля Src/entities/achievements

Назва GraphQL ендпоїнту	Функція-обробник	Призначення
Query allAchievements	allAchievements()	Повертає список усіх можливих досягнень
Query achievementsByIds(ids : [ID!]!)	achievementsByIds(@Args('ids') ids: string[])	Повертає список досягнень за їх унікальними ключами в БД
Query achievementsByPos(p os: [Int!]!)	achievementsByPos(@Args('pos') pos: number[])	Повертає список досягнень за їх унікальними pos-номерами
Query achievenmntsCount	achievenmntsCount ()	Повертає кількість усіх можливих досягнень

Src/entities/achievements реалізує логіку по роботі із музичними змаганнями.

Основні функції даного модуля наведено у табл. 3.2.

Таблиця 3.2 - Функції модуля Src/entities/battles

Назва GraphQL ендпоїнту	Функція-обробник	Призначення
Query battleById(_id: ID!)	battleById(@Args('_id') _id: string)	Повертає змагання за унікальним ідентифікатором
Query battlesByStatus(f inished: Boolean!, offset: Int!, limit: Int!)	battlesByStatus(@Args('fi nished', ParseBoolPipe) finished: boolean, @Args('offset', ParseIntPipe) offset: number, @Args('limit', ParseIntPipe) limit: number,)	Повертає список змагань за статусом 'finished' із параметрами offset та limit.
Query battlesUserPartic ipatedIn(userId: ID!, offset: Int!, limit: Int!)	battlesUserParticipatedIn(@Args('userId') userId: string, @Args('offset', ParseIntPipe) offset: number, @Args('limit', ParseIntPipe) limit: number,)	Повертає список змагань у яких користувач певним чином приймав участь (наприклад голосував або був одним з гравців)
Mutation battleCreate(inpu t: AddNewBattleB yPostsIdsInput!)	battleCreate(@Args('input') dto: CreateBattleDto)	Створює нове змагання

Продовження таблиці 3.2

Назва GraphQL ендпоїнту	Функція-обробник	Призначення
Mutation battleDeleteById(_id: ID!)	battleDeleteById(@A rgs('_id') _id: string)	Видаляє змагання за його унікальним ідентифікатором в БД.
Mutation battleMakeVote(input: MakeBattleVoteInput!)	battleMakeVote(@Ar gs('input') dto: MakeBattleVoteDto)	Надає користувачу функціонал голосування

Src/entities/moderations реалізує логіку по роботі із зміною приватних даних користувача (пароль, пошта і т.д), створює запити на зміну цих даних.

Основні функції даного модуля наведено у табл. 3.3.

Таблиця 3.3 - Функції модуля Src/entities/moderations

Назва GraphQL ендпоїнту	Функція-обробник	Призначення
Query moderationActionValidate(input : ModerateActionInput!)	moderationActionVali date(@Args('input') input: ModerationDto)	Перевіряє валідність даних під час відновлення доступу до акаунта
Moderation moderationActionCreate(input: CreateModerationActionInput!)	moderationActionCrea te(@Args('input') input: CreateModerationDto)	Створює запис у БД для подальшої валідації під час відновлення акаунта

Продовження таблиці 3.3

Назва GraphQL ендпоіту	Функція-обробник	Призначення
Moderation moderationActionDelete(input: ModerateActionInput!)	moderationActionDelete(@Args('input') input: ModerationDto)	Видаляє запис для неможливості повторно зкористатись під час відновлення акаунта

Src/entities/notifications реалізує логіку по роботі із повідомленнями для користувачів.

Основні функції даного модуля наведено у табл. 3.4.

Таблиця 3.4 - Функції модуля Src/entities/notifications

Назва GraphQL ендпоіту	Функція-обробник	Призначення
Query notifications(receiver Id: ID!, checked: Boolean!, offset: Int!, limit: Int!)	notifications(@Arg s('receiverId') receiverId: string, @Args('checked') checked: boolean, @Args('offset', ParseIntPipe) offset: number, @Args('limit', ParseIntPipe) limit: number)	Повертає список повідомлень для конкретного користувача
Query notificationsByIds(ids: [ID!]!)	notificationsByIds(@Ar gs('ids') ids: string[])	Повертає список по повідомлень по їх унікальним ідентифікаторам

Продовження таблиці 3.4

Назва GraphQL ендпоїнту	Функція-обробник	Призначення
Mutation notificationCreate(input: CreateNotificationIn put!)	notificationCreate(@Ar gs('input') dto: CreateNotificationDto)	Створення повідомлення
Mutation notificationDeleteBy Id(_id: ID!)	notificationDeleteById(@Args('_id') _id: string)	Видалення повідомлення
Mutation notificationsDeleteB yIds(ids: [ID!]!)	notificationsDeleteByIds (@Args('ids') ids: string[])	Видалення повідомлень по їх унікальним ідентифікаторам
Mutation notificationMarkAsR eadById(_id: ID!)	notificationMarkAsRead ById(@Args('_id') _id: string)	Зміна статусу повідомлення на прочитане
Mutation notificationsMarkAs ReadByIds(ids: [ID!]!)	notificationsMarkAsRea dByIds(@Args('ids') ids: string[])	Зміна статусу повідомлень на прочитане за їх унікальними ідентифікаторами

Src/entities/planned-tasks реалізує логіку по роботі із запланованими подіями на стороні сервера, наприклад завершити змагання по завершенню таймера і розрахувати переможця.

Методи даного модуля не використовуються на клієнтській стороні, тому вони не матимуть декораторів Query або Mutation.

Основні функції модуля наведено у табл. 3.5.

Таблиця 3.5 - Функції модуля Src/entities/planned-tasks

Назва	Призначення
createPlannedTask(dto: CreateTaskDto)	Створення запланованої події
deletePlannedTask(taskType: string, relatedEntityId: string)	Видалення події за 2-ма ключами
getAllTasks()	Повертає список усіх запланованих подій

Src/entities/playlists реалізує логіку по роботі із плейлистами користувача.

Основні функції модуля наведено у табл. 3.6.

Таблиця 3.6 - Функції модуля Src/entities/playlists

Назва GraphQL ендпоїнту	Функція-обробник	Призначення
Query playlist(_id: ID!)	playlist(@Args('_id') _id: string)	Повертає плейлист за вказаним унікальним ідентифікатором
Query playlistsByTitle(title: String!)	playlistsByTitle(@Args('title') title: string)	Повертає список плейлистів за назвою
Query playlistsByOwnerId(owner: ID!, offset: Int!, limit: Int!)	playlistsByOwnerId(@Args('owner') owner: string, @Args('offset', ParseIntPipe) offset: number, @Args('limit', ParseIntPipe) limit: number)	Повертає список плейлистів за власником (owner)

Продовження таблиці 3.6

Назва GraphQL ендпоїнту	Функція-обробник	Призначення
Query playlistsPublicAvailable(offset: Int!, limit: Int!)	playlistsPublicAvailable(@Args('offset', ParseIntPipe) offset: number, @Args('limit', ParseIntPipe) limit: number)	Повертає публічно-доступні плейлисти
Mutation playlistCreate(input: CreatePlaylistInput!)	playlistCreate(@Args('input') dto: CreatePlaylistDto, @CurrentUser() user: UserDocument)	Створює плейлист і повертає його
Mutation playlistDeleteById(_id: ID!)	playlistDeleteById(@Args('_id') _id: string, @CurrentUser() user: UserDocument)	Видаляє плейлист за унікальним ідентифікатором
Mutation playlistSwitchTrack(input: SwitchTrackInPlaylistInput)	playlistSwitchTrack(@Args('input') dto: SwitchTrackDto, @CurrentUser() user: UserDocument)	Додає або видаляє трек із плейлиста

Src/entities/posts реалізує логіку по роботі із вивантаженими треками користувача. Він відповідає за надання усього необхідного функціоналу для гнучкого управління треками з клієнтської частини розроблюваного застосунку.

Всі функції модуля наведено у табл. 3.7.

Таблиця 3.7 - Функції модуля Src/entities/posts

Назва GraphQL ендпоїнту	Функція-обробник	Призначення
Query post(_id: ID!)	post(@Args('_id') _id: string)	Повертає трек за унікальним ідентифікатором
Query posts(offset: Int!, limit: Int!)	posts(@Args('offset', ParseIntPipe) offset: number, @Args('limit', ParseIntPipe) limit: number)	Повертає список треків
Query postsByOwner(owner: ID!, offset: Int!, limit: Int!)	postsByOwner(@Args('owner') owner: string @Args('offset', ParseIntPipe) offset: number, @Args('limit', ParseIntPipe) limit: number)	Повертає список треків за їх власником
Query postsByTitle(input: PostsByTitleInput!)	postsByTitle(@Args('input') dto: PostsByTitleDto)	Повертає список треків за їх назвою
Query postsByIds(ids: [ID!]!)	postsByIds(@Args('ids') ids: string[])	Повертає список треків за їх унікальними ідентифікаторами
Query postsMostPopular(date: Date!)	postsMostPopular(@Args('date') date: string)	Повертає список треків за популярністю і стартовою датою

Продовження таблиці 3.7

Назва GraphQL ендпоїнту	Функція-обробник	Призначення
Query postsByCategory(category: String!, offset: Int!, limit: Int!)	postsByCategory(@Args('category') category: string, @Args('offset', ParseIntPipe) offset: number, @Args('limit', ParseIntPipe) limit: number,)	Повертає список треків за категорією
Query postsSavedByUser(user: ID!, offset: Int!, limit: Int!)	postsSavedByUser(@Args('user') user: string, @Args('offset', ParseIntPipe) offset: number, @Args('limit', ParseIntPipe) limit: number)	Повертає список збережених треків користувача
Query postsByCategoryCount	postsByCategoryCount()	Повертає кількість треків у категорії
Query postsMostRecent	postsMostRecent()	Повертає список найновіших треків
Query postsMostRecentByFollowing(user: ID!)	postsMostRecentByFollowing(@Args('user') userId: string)	Повертає список найновіших треків від відстежуваних користувачів

Продовження таблиці 3.7

Назва GraphQL ендпоїнту	Функція-обробник	Призначення
Mutation postCreate(input: AddPostInput!)	postCreate(@Args('input') dto: CreatePostDto, @CurrentUser() user: UserDocument)	Сворює і повертає трек
Mutation postUpdate(input: UpdatePostInput!)	postUpdate(@Args('input') { post, value, what }: UpdatePostDto, @CurrentUser() user: UserDocument)	Оновлює і повертає трек
Mutation postDeleteById(_id: ID!)	postDeleteById(@Args('_id') _id: string, @CurrentUser() user: UserDocument)	Видаляє трек за вказаним унікальним ідентифікатором

Src/entities/reports реалізує логіку по роботі із скаргами.

Основні функції модуля наведено у табл. 3.8.

Таблиця 3.8 - Функції модуля Src/entities/reports

Назва GraphQL ендпоїнту	Функція-обробник	Призначення
Query report(_id: ID!)	report(@Args('_id') _id: string)	Повертає скаргу за вказаним ідентифікатором

Продовження таблиці 3.8

Назва GraphQL ендпоїнту	Функція-обробник	Призначення
Queryreports(offset: Int!, limit: Int!)	reports(@Args('offset', ParseIntPipe) offset: number, @Args('limit', ParseIntPipe) limit: number,)	Повертає список скарг
Mutation reportCreate(input: CreateReportInput!)	reportCreate(@Args('input') dto: CreateReportDto)	Створює і повертає скаргу
Mutation reportClose(_id: ID!)	reportClose(@Args('_id') _id: string)	Закриває скаргу (скаргу було розглянуто)

Src/entities/support-requests реалізує логіку по роботі із запитами в службу підтримки.

Основні функції модуля наведено у табл. 3.9.

Таблиця 3.9 - Функції модуля Src/entities/support-requests

Назва GraphQL ендпоїнту	Функція-обробник	Призначення
Query supportRequest(_id: ID!)	supportRequest(@Args('_id') _id: string)	Повертає звернення за вказаним ідентифікатором

Продовження таблиці 3.9

Назва GraphQL ендпоїнту	Функція-обробник	Призначення
Query supportRequests(offset: Int!, limit: Int!)	supportRequests(@Args('offset', ParseIntPipe) offset: number, @Args('limit', ParseIntPipe) limit: number)	Повертає список звернень
Mutation supportRequestCreate(input: CreateSupportRequestInput!)	supportRequestCreate(@Args('input') dto: CreateSupportRequestDto)	Створює і повертає звернення
Mutation supportRequestClose(_id: ID!)	supportRequestClose(@Args('_id') _id: string)	Закриває звернення (було розглянуто)

Src/entities/users реалізує логіку по роботі із обліковими записами користувачів.

Основні функції модуля наведено у табл. 3.10.

Таблиця 3.10 - Функції модуля Src/entities/users

Назва GraphQL ендпоїнту	Функція-обробник	Призначення
Query user(_id: ID!)	user(@Args('_id') _id: string)	Повертає користувача за унікальним ідентифікатором

Продовження таблиці 3.10

Назва GraphQL ендпоїнту	Функція-обробник	Призначення
Query notificationsMarkAsReadBy users	notificationsMarkAsReadBy Ids(@Args('ids') ids: string[])	Повертає усіх користувачів
Query userByEmail(email: String!)	userByEmail(@Args('email') email: string)	Повертає користувача за email
Query usersByIds(ids: [ID!]!)	usersByIds(@Args('ids') ids: string[])	Повертає користувачів за унікальними ідентифікаторами
Query usersByNickname(nick: String!)	usersByNickname(@Args('n ick') nick: string)	Повертає користувачів за нікнеймами
Query userAchievementsData(_id: ID!)	userAchievementsData(@Ar gs('_id') _id: string)	Повертає досягнення користувачів
Query userPinnedTracks(_id: ID!)	userPinnedTracks(@Args('_ id') _id: string)	Повертає список закріплених треків користувача
Mutation userCreate(input: AddUserInput!)	userCreate(@Args('input') input: CreateUserDto)	Створює і повертає користувача

Продовження таблиці 3.10

Назва GraphQL ендпоїнту	Функція-обробник	Призначення
Mutation userUpdate(input: UpdateUserInput!)	userUpdate(@Args('input') { _id, what, value }: UpdateUserDto, @CurrentUser() user: UserDocument)	Оновлює дані користувача
Mutation userSwitchSubscr iption(input: SwitchSubscriptio nOnUserInput!)	userSwitchSubscription(@A rgs('input') { subscriberId, userId }: SwiathSubscriptionDto, @CurrentUser() user: UserDocument)	Створює підписку на користувача, або скасовує її
Mutation userConfirmAcco unt(input: AccountConfirmI nput!)	userConfirmAccount(@Arg s('input') dto: ConfirmAccountDto)	Підтверджує власника акаунта
Mutation userRestoreAccou nt(input: AccountRestoreIn put!)	userRestoreAccount(@Args ('input') dto: RestoreAccountDto)	Змінює дані через форму відновлення
Mutation userPrepareAccou ntToRestore(input : PrepareAccountT oRestoreInput)	userPrepareAccountToResto re(@Args('input') dto: PrepareToRestoreDto)	Підготовлює акаунт до відновлення

Продовження таблиці 3.10

Назва GraphQL ендпоїнту	Функція-обробник	Призначення
Mutation userUnlinkGoogle(_id: ID!)	userUnlinkGoogle(@Args('_id') _id: string, @CurrentUser() user: UserDocument)	Скасовує прив'язку Google акаунта
Mutation userUnlinkFacebook(_id: ID!)	userUnlinkFacebook(@Args('_id') _id: string, @CurrentUser() user: UserDocument)	Скасовує прив'язку Facebook акаунта
Mutation userUnlinkTwitter(_id: ID!)	userUnlinkTwitter(@Args('_id') _id: string, @CurrentUser() user: UserDocument)	Скасовує прив'язку Twitter (X) акаунта
Mutation userSwitchLike(input: SwitchLikeOrPostInSaved Input!)	userSwitchLike(@Args('input') dto: SwicthLikeOrSaveDto, @CurrentUser() user: UserDocument)	Ставить / знімає вподобайку з трека
Mutation userSwitchSave(input: SwitchLikeOrPostInSaved Input!)	userSwitchSave(@Args('input') dto: SwicthLikeOrSaveDto, @CurrentUser() user: UserDocument)	Додає / видаляє трек із збережених
Mutation userSwitchPostPinned(use rId: ID!, postId: ID!)	userSwitchPostPinned(@Args('userId') userId: string, @Args('postId') postId: string, @CurrentUser() user: UserDocument)	Додає / видаляє трек із закріплених

Продовження таблиці 3.10

Назва GraphQL ендпоїнту	Функція-обробник	Призначення
Mutation userSwitchPostPinned(use rId: ID!, postId: ID!)	userLinkEmailRequest(@Ar gs('input') input: LinkEmailDto, @CurrentUser() user: UserDocument)	Створює запит на прив'язку пошти до акаунта

3.2 Основні рішення щодо розробки бази даних

База даних складатиметься із декількох основних компонентів (документів), структуру розроблюваної БД наведено на рис. 3.5.

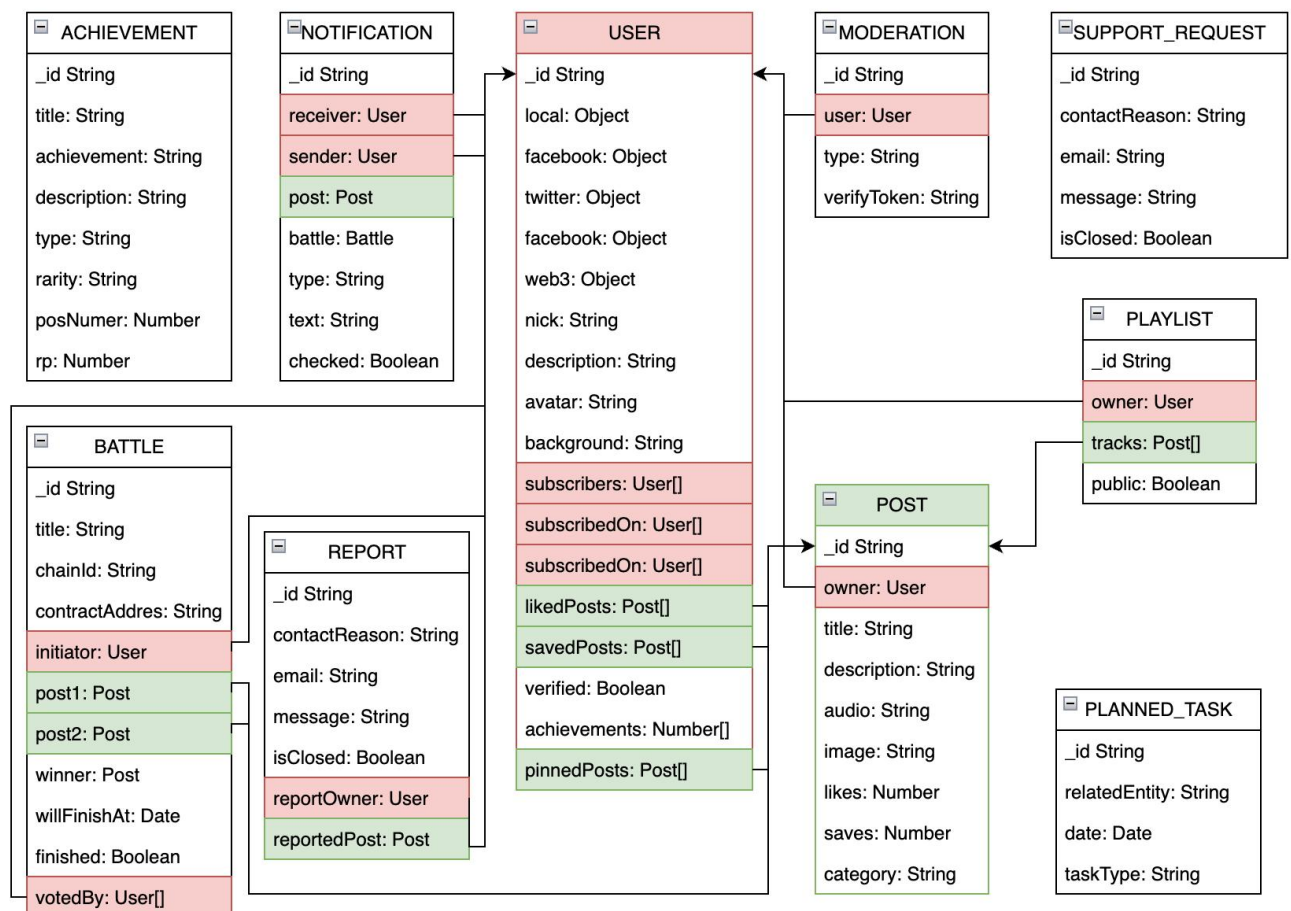


Рисунок 3.5 - Структура БД

Опис структури БД наведено нижче у табл. 3.11 - 3.19.

Нижче наведено таблицю, що відповідає за структуру запису користувача у базі даних, як головної сутності застосунку.

Таблиця 3.11 - Опис структури USER

Поле	Призначення
_id	Унікальний ідентифікатор
local.email	Електронна пошта
local.password	Пароль
facebook	Дані облікового запису Facebook
twitter	Дані облікового запису Twitter
google	Дані облікового запису Google
web3	Дані крипто-гаманця
nick	Псевдонім
description	Опис профілю
avatar	Посилання на зображення у IPFS
background	Посилання на зображення у IPFS
subscribers	Список підписників
subscribedOn	Підписки користувача
likedPosts	Уподобані пости
savedPosts	Збережені пости
verified	Статус перевірки
achievements	Досягнення користувача
pinnedPosts	Закріплені пости

За формування структури досягнення користувачів в базі даних відповідає табл. 3.12, у даній структурі можна побачити 2 ключові поля: _id та posNumber, в даному випадку _id виконує свою стандартну роль, а posNumber відповідає за унікальний номер досягнення, що не буде генеруватись автоматично базою даних.

Таблиця 3.12 - Опис структури ACHIEVEMENT

Поле	Призначення
_id	Унікальний ідентифікатор
title	Назва досягнення
achievement	Опис досягнення
type	Тип досягнення
rarity	Рідкість досягнення
gp	Кількість нагородних балів
posNumber	Номер досягнення (не залежить від _id)

Сутність повідомлення, відповідає за загальне інформування користувачів про важливі події, наприклад створення змагання з їх участю.

Загальну структуру повідомлення наведено у табл. 3.13.

Таблиця 3.13 - Опис структури NOTIFICATION

Поле	Призначення
_id	Унікальний ідентифікатор
receiver	Користувач, який отримує
sender	Користувач, який відправляє
post	Пов'язаний трек
battle	Пов'язаний батл
type	Тип повідомлення
text	Текст повідомлення
checked	Статус перегляду

Структура змагання описує основні складові необхідної інформації для відображення в інтерфейсі користувача, детальніше у табл. 3.14.

Таблиця 3.14 - Опис структури BATTLE

Поле	Призначення
_id	Унікальний ідентифікатор
title	Назва битви
chainId	Ідентифікатор Web3 мережі
contractAddress	Адреса смарт-контракту у blockchain
initiator	Ініціатор битви
post1	Трек 1
post2	Трек 2
post1Score	Результат посту 1
post2Score	Результат посту 2
winner	Трек-переможець
willFinishAt	Дата завершення батлу
finished	Статус завершення
votedBy	Список користувачів що проголосували

У застосунку також існують заплановані системні події, що використовуються тільки сервером і ніколи не повертаються на клієнтську частину, загальну структуру запису запланованої події у БД наведено у табл. 3.15.

Таблиця 3.15 - Опис структури PLANNED_TASK

Поле	Призначення
_id	Унікальний ідентифікатор
relatedEntityId	Ідентифікатор пов'язаної сутності
date	Дата для запуску на виконання
taskType	Тип задачі

За зберігання інформації про скарги або звертання користувачів відповідає структура, наведена у табл. 3.16.

Таблиця 3.16 - Опис структури REPORT

Поле	Призначення
_id	Унікальний ідентифікатор
contactReason	Причина звернення
email	Електронна пошта користувача
message	Деталі звернення
isClosed	Статус на розглянутість
reportOwner	Автор скарги
reportedPost	Пов'язаний трек

За формування основного контенту вебзастосунку - треків користувачів, відповідають записи в базі даних із відповідною структурою, що наведено в табл. 3.17.

Таблиця 3.17 - Опис структури POST

Поле	Призначення
_id	Унікальний ідентифікатор
owner	Власник треку
title	Назва треку
description	Опис треку
audio	Посилання на аудіо у IPFS
image	Посилання на зображення у IPFS
likes	Кількість вподобань
saves	Кількість збережень
category	Жанр
downloadsAllowed	Флаг на дозвіл завантаження

За формування плейлистів користувачів виступає структура “Playlist”, опис полів наведено у табл. 3.18.

Таблиця 3.18 - Опис структури PLAYLIST

Поле	Призначення
_id	Унікальний ідентифікатор
owner	Власник плейлиста
tracks	Список треків

За зберігання звертань користувачів для отримання технічної підтримки відповідає структура “Support Request”, опис її полів наведено у табл. 3.19.

Таблиця 3.19 - Опис структури SUPPORT_REQUEST

Поле	Призначення
_id	Унікальний ідентифікатор
contactReason	Причина звернення
email	Електронна пошта юзера що звертається
message	Повідомлення
isClosed	Статус розглянутості

3.3 Основні рішення щодо розробки інтерфейсу

Клієнтська частина складається із компонентів, візуалізацію основних компонентів наведено нижче у відповідних підрозділах.

3.3.2 Компонент “appbar”

Відповідає за навігацію застосунку і посилається на основні можливості і контент платформи.

Візуалізація компоненту зображена на рис. 3.6.

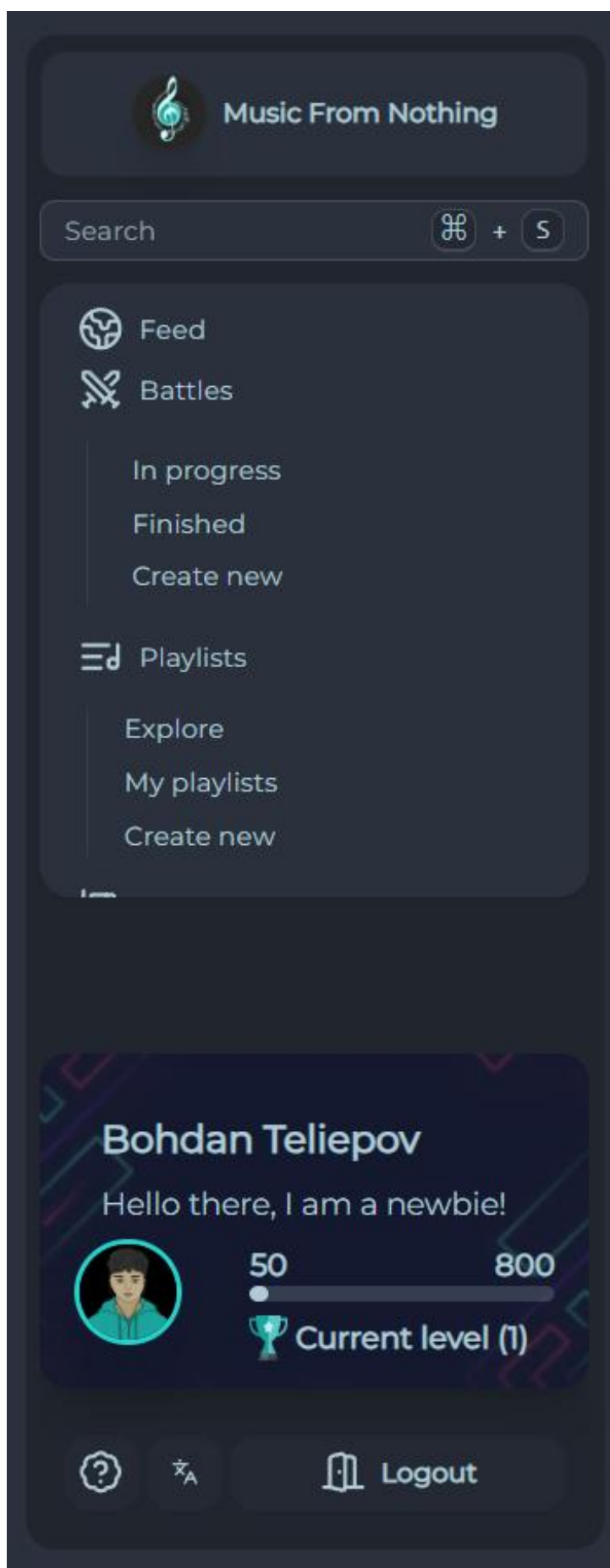


Рисунок 3.6 - Компонент “appbar”

3.3.3 Компонент “battle”

Відповідає за змагання між користувачами (голосування проходить за більш вдалий трек).

Візуалізація компоненту зображена на рис. 3.7.

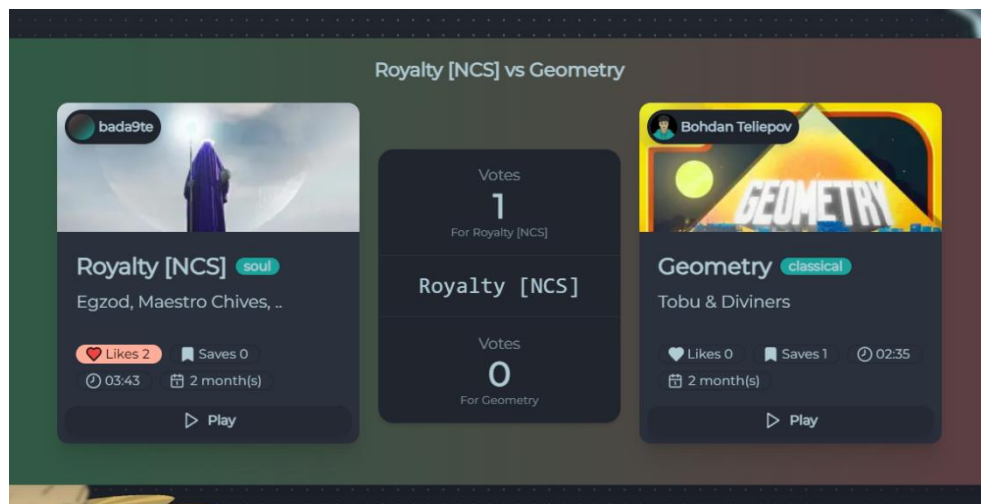


Рисунок 3.7 - Компонент змагань

3.3.4 Компонент “player”

Відповідає за відтворення музичного контенту.

Візуалізація компоненту зображена на рис. 3.8.

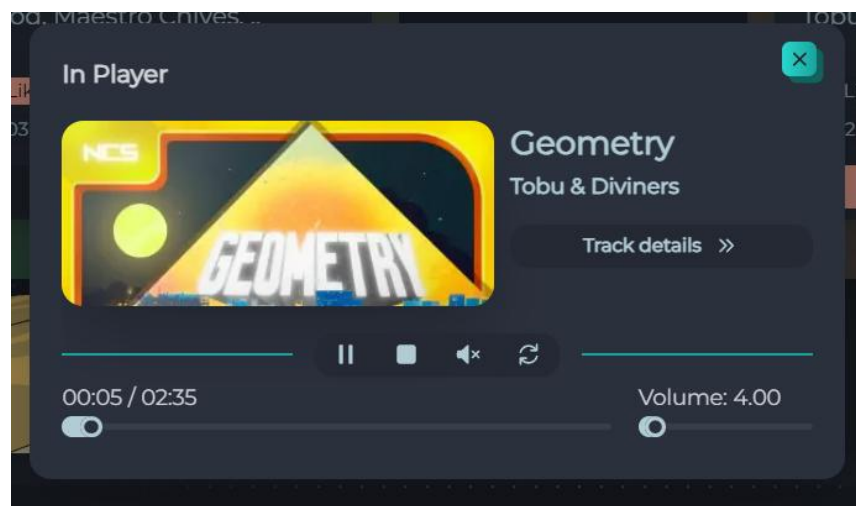


Рисунок 3.8 - Музичний програвач

3.3.5 Компонент “forms/*”

Відповідає за отримання і оперування даними користувача.

Візуалізація концепції компонентів форм на всій платформі зображена на рис. 3.9.

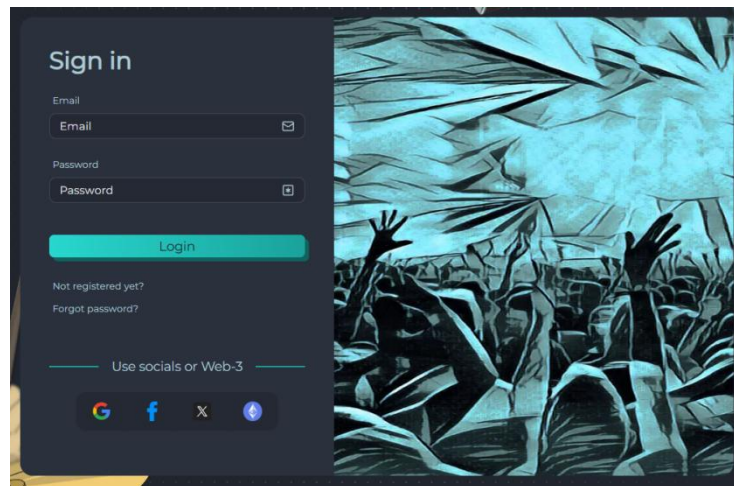


Рисунок 3.9 - Візуалізація концепції усіх форм платформи

3.3.6 Компонент “playlist”

Відповідає за зберігання треків користувачів за певними спільними характеристиками.

Візуалізація компоненту зображена на рис. 3.10.

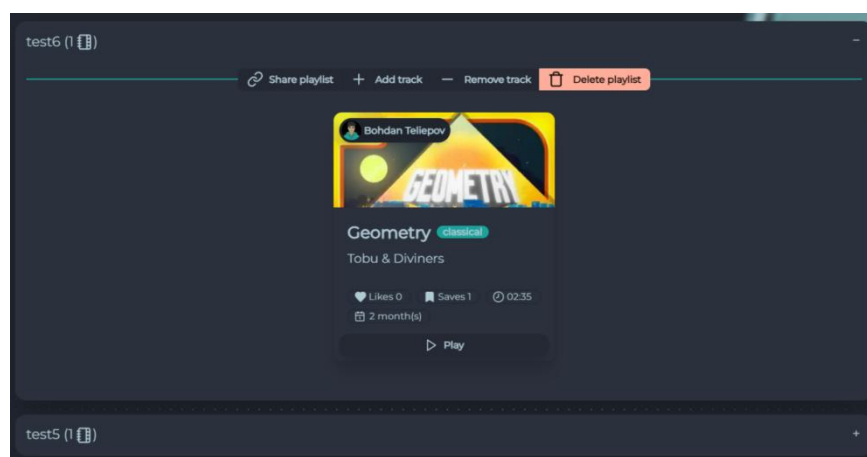


Рисунок 3.10 - Плейлист користувача

3.3.7 Компонент “notification”

Відповідає за інформування користувачів про важливі події що можуть бути пов’язані із їх акаунтами.

Візуалізація компоненту зображена на рис. 3.11.

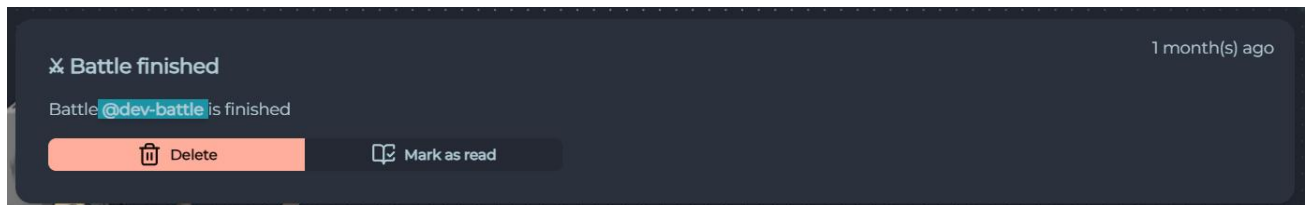


Рисунок 3.11 - Компонент повідомлення

3.3.1 Компонент “post”

Відповідає за відображення контенту всіх користувачів (їх вивантажених треків).

Візуалізація компоненту зображена на рис. 3.12.

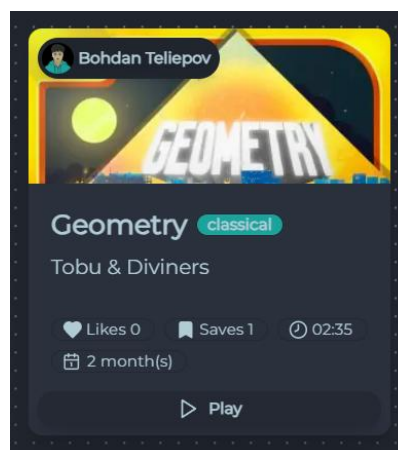


Рисунок 3.12 - Трек користувача

3.4 Основні рішення щодо розроблених алгоритмів

Нижче наведено принцип роботи алгоритму, який використовується у

смарт-контракті, і виконує роль підрахування нагород у змаганнях для користувача що відправив запит рис. 3.13.

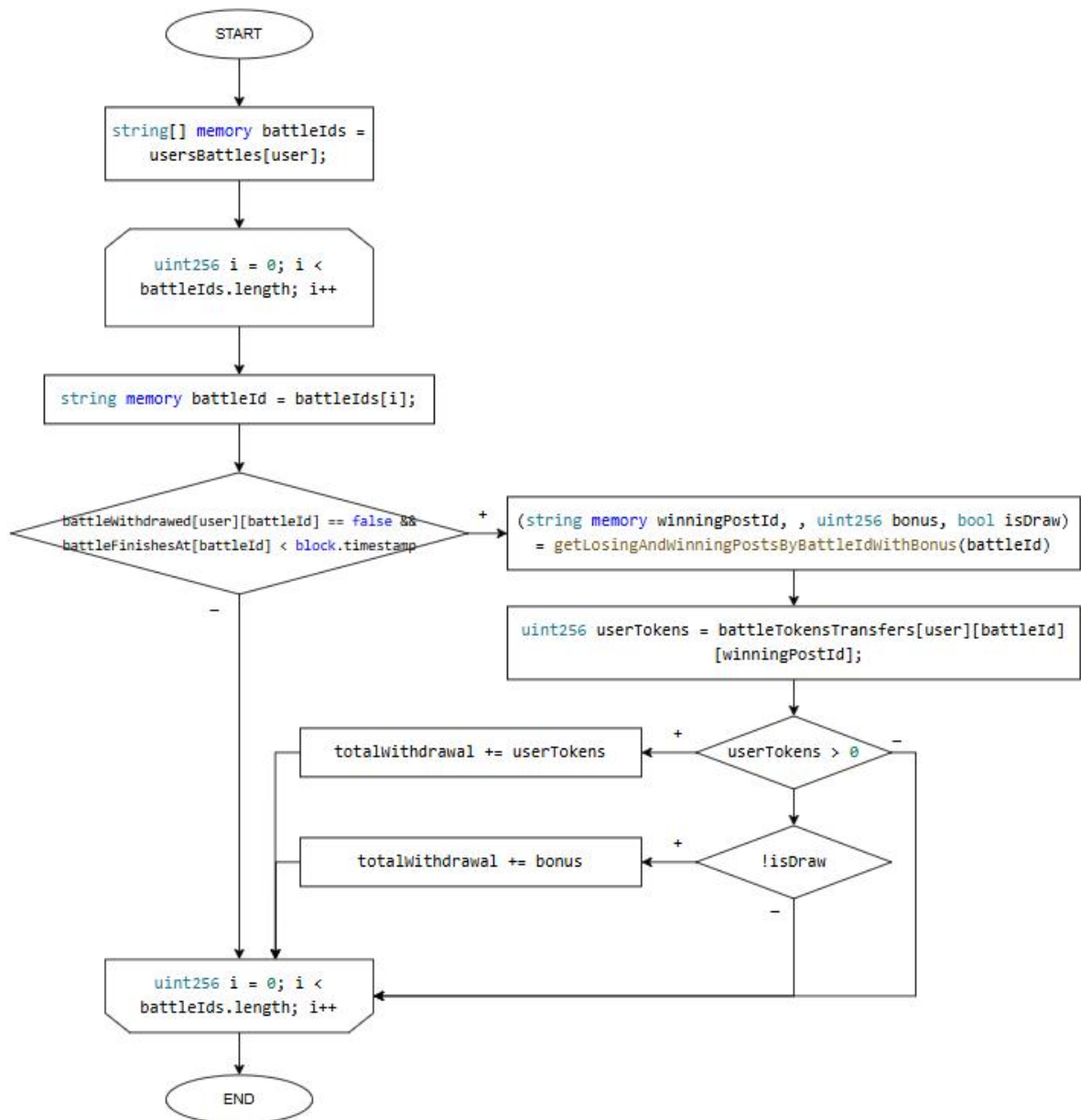


Рисунок 3.13 - Алгоритм підрахунку нагород у змаганнях користувача

3.5 Висновки з розділу

В третьому розділі було детально розглянуто кожну частину застосунку, включаючи проектування БД, реалізації основних алгоритмів і опис принципу їх функціонування.

При створенні бази даних було обрано NoSQL базу даних, а саме MongoDB [14], для можливості дуже швидко розширювати застосунок без прив'язки до структури попередніх реалізацій структур сутностей. Усього спроектовано 9 моделей, кожна з яких містить всю необхідну інформацію, що може знадобитися при зв'язуванні усього функціоналу застосунку.

При створенні серверної частини, використовуючи фреймворк NestJS та мову програмування TypeScript, було продумано та реалізовано основні GraphQL ендпоінти для отримання та модифікації даних. Потім прив'язано дані ендпоінти до функцій модулів для взаємодії із БД, реалізованих раніше.

Для створення клієнтської частини використано фреймворк NextJS, також на мові програмування TypeScript, було створено та підключено всі сторінки застосунку і їх функціонал до фінальної версії програми. Для підключення функціоналу застосунку використано HTTP/GraphQL запити.

При реалізації частини із смарт-контрактами використано мову програмування Solidity, на основі фреймворка Hardhat, скрипти для вивантаження програмного коду в мережу написані на мові програмування TypeScript.

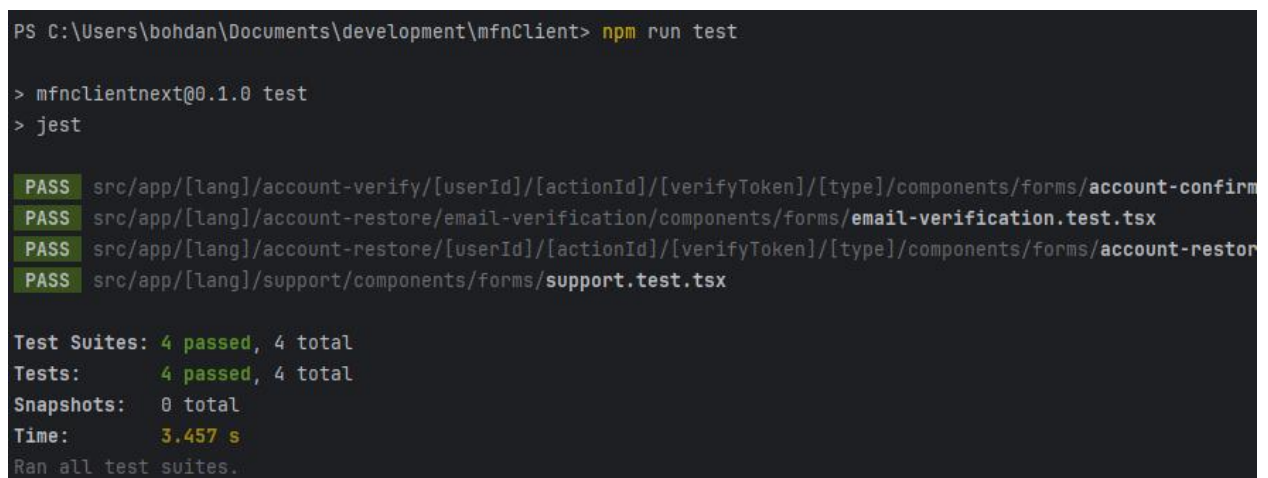
4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для тестування і перевірки якості розроблюваного програмного продукту і коду було створено та проведено unit-тестування [15]-[17] усіх основних компонентів системи.

Під кожную версію платформи, у разі кардинальної зміни логіки роботи модуля і структури повертаючих даних із його методів - необхідно переналаштовувати конфліктуєчі тест-кейси.

4.1 Тестування клієнтської частини

На стороні клієнтської частини було розроблено 4 основні тест-кейси для перевірки працездатності основних форм взаємодії із користувачами. Результати тестування наведено на рис. 4.1.



```
PS C:\Users\bohdan\Documents\development\mfncClient> npm run test

> mfncClientNext@0.1.0 test
> jest

PASS src/app/[lang]/account-verify/[userId]/[actionId]/[verifyToken]/[type]/components/forms/account-confirm
PASS src/app/[lang]/account-restore/email-verification/components/forms/email-verification.test.tsx
PASS src/app/[lang]/account-restore/[userId]/[actionId]/[verifyToken]/[type]/components/forms/account-restore
PASS src/app/[lang]/support/components/forms/support.test.tsx

Test Suites: 4 passed, 4 total
Tests: 4 passed, 4 total
Snapshots: 0 total
Time: 3.457 s
Ran all test suites.
```

Рисунок 4.1 - Загальні тести основних форм клієнтської частини (4 загальних тести)

Також було оцінено зручність використання платформи в різних браузерах, та на пристроях із різною роздільною здатністю екрана. Під час тестування інтерфейсу було наділено велику увагу до розположення елементів навігації, розташування основного контенту та передбачуваність

інтерфейсу при виконанні основних операцій, таких як програвання треку, або редагування акаунта.

4.1.1 Взаємодія із програмним забезпеченням

Для відтворення треку необхідно скористатися відповідною кнопкою (рис. 4.2) для запуску і паузи (рис. 4.3).

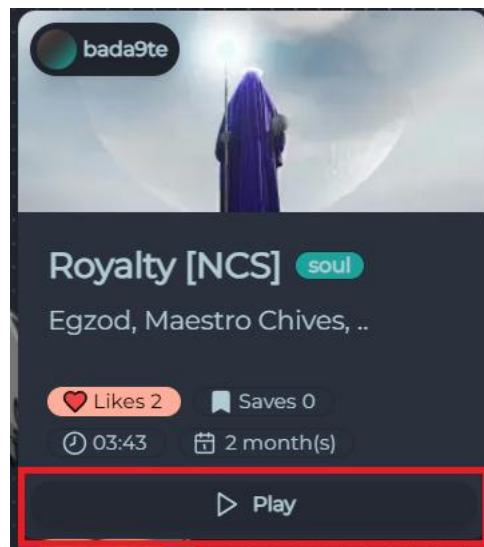


Рисунок 4.2 - Можливість відтворити трек

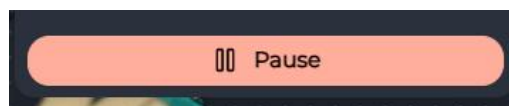


Рисунок 4.3 - Можливість зупинити програвання треку

Для додавання треків у обране необхідно скористатися відповідною кнопкою (рис. 4.4 - 4.5).

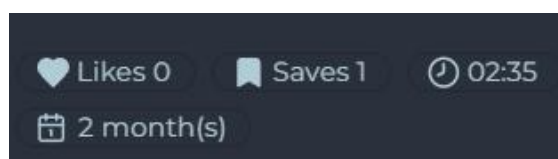


Рисунок 4.4 - Не збережений трек

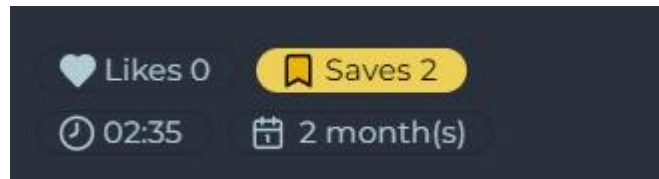


Рисунок 4.5 - Збережений трек

Для створення нового плейлиста необхідно заповнити форму (рис. 4.6) і спеціальними функціональними кнопками можна буде проводити операції над плейлистом (рис. 4.7).

 A 'Playlist setup' modal form on a dark background. It has a title 'Playlist setup' in large white text. Below it is a 'Playlist title' label and a text input field containing 'Playlist title' with a menu icon on the right. Underneath is a 'Publically available' label and an unchecked checkbox. At the bottom is a large teal button labeled 'Create playlist'.

Рисунок 4.6 - Форма створення плейлиста

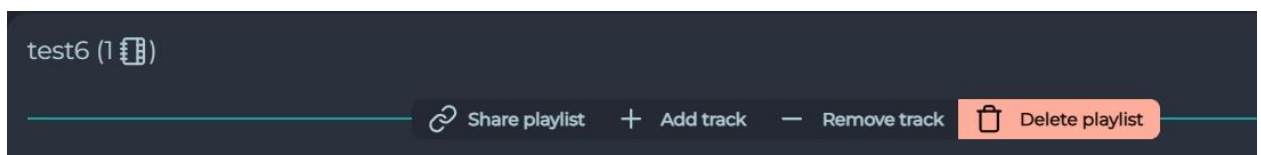


Рисунок 4.7 - Кнопки оперування плейлистом

Для редагування створеного користувачем треку необхідно натиснути у випадаючому меню на “edit” - редагування (рис. 4.8), після чого користувача буде переспрямовано на сторінку редагування (рис. 4.9).

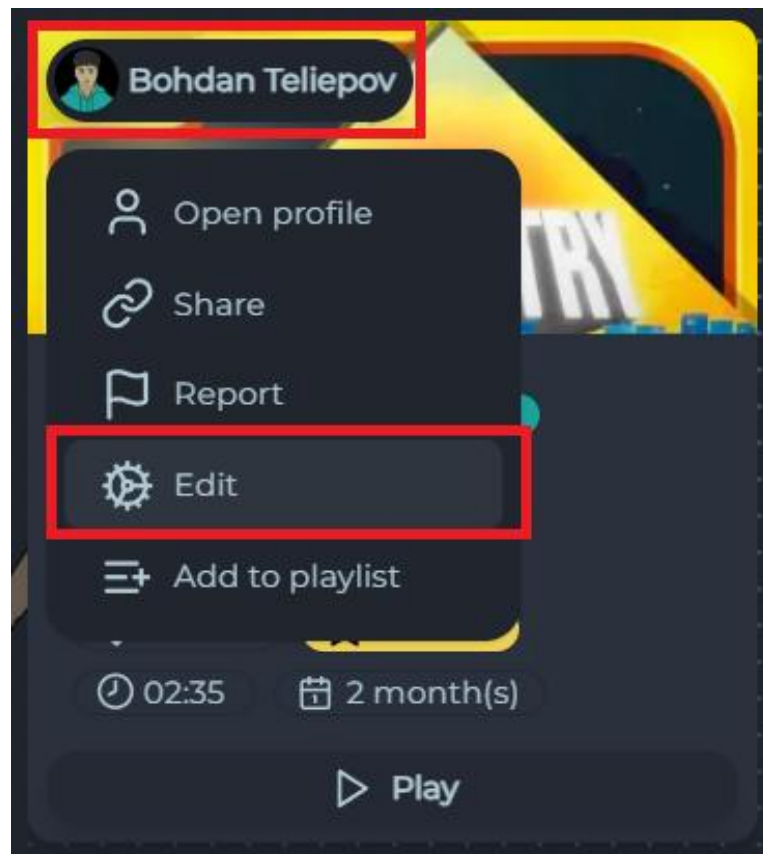


Рисунок 4.8 - Контекстне меню для переходу до редагування треку

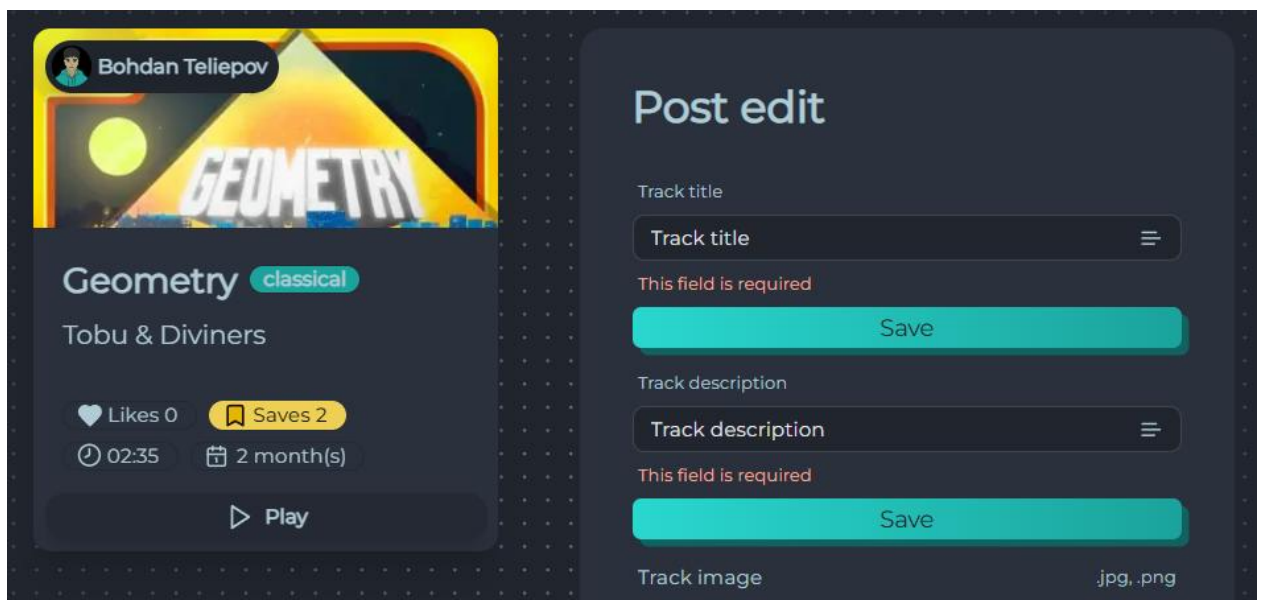
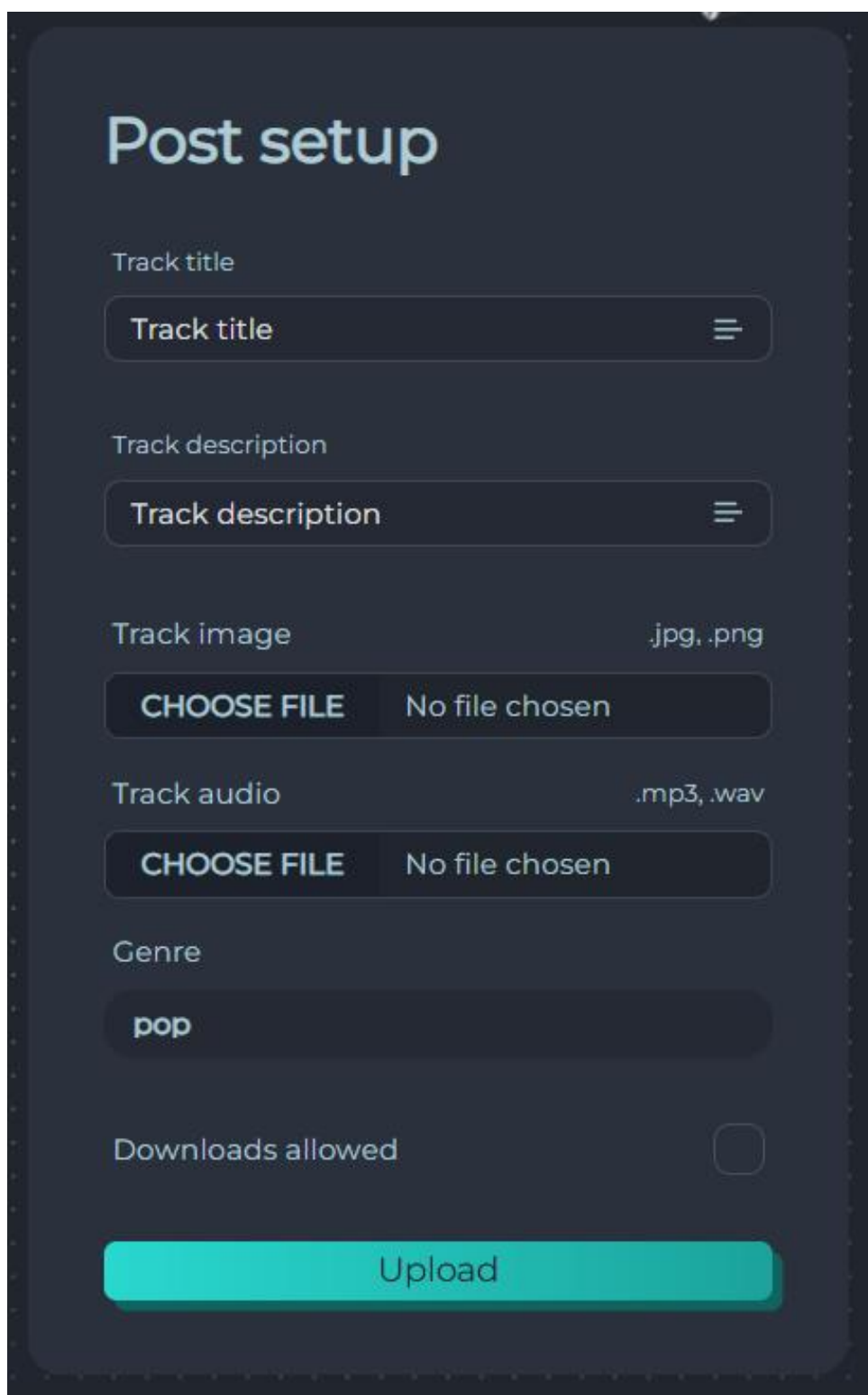


Рисунок 4.9 - Сторінка редагування вивантаженого треку

Для створення треку (його вивантаження), необхідно перейти на відповідну сторінку і скористатись спеціальною формаю для подальших кроків (рис. 4.10).



The image shows a 'Post setup' form with a dark background. It contains several input fields and buttons. At the top is the title 'Post setup'. Below it are fields for 'Track title' and 'Track description', each with a text input and a menu icon. The 'Track image' section has a 'CHOOSE FILE' button and the text 'No file chosen', with supported formats '.jpg, .png' listed to the right. The 'Track audio' section has a similar 'CHOOSE FILE' button and 'No file chosen' text, with supported formats '.mp3, .wav' listed to the right. The 'Genre' section has a dropdown menu currently showing 'pop'. The 'Downloads allowed' section has a checkbox that is currently unchecked. At the bottom is a large teal 'Upload' button.

Post setup

Track title

Track title

Track description

Track description

Track image .jpg, .png

CHOOSE FILE No file chosen

Track audio .mp3, .wav

CHOOSE FILE No file chosen

Genre

pop

Downloads allowed

Upload

Рисунок 4.10 - Форма створення посту

Для редагування профілю користувача необхідно на авторизованому акаунті натиснути на аватар користувача і у випадяючому меню натиснути на відповідну кнопку (рис. 4.11).

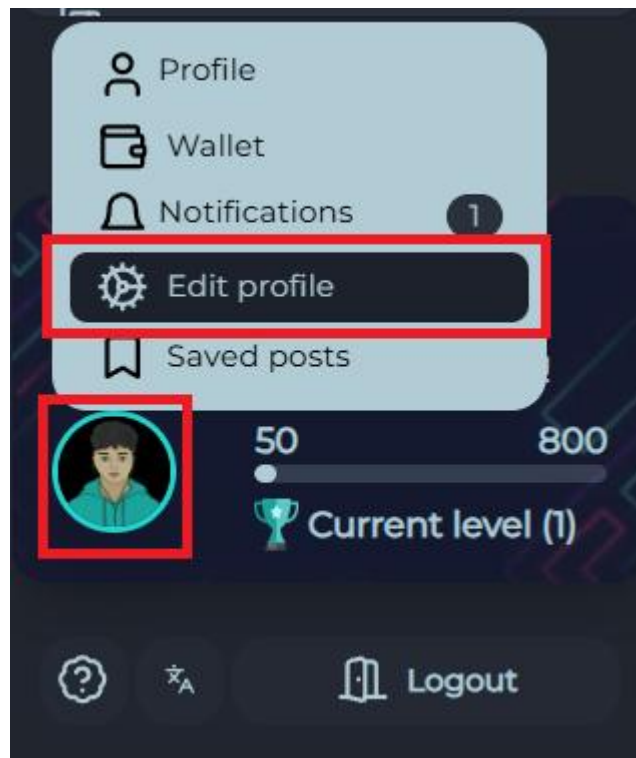


Рисунок 4.11 - Перехід на сторінку редагування профілю

Для зміни заднього фону та аватару користувача можна скористатись спеціальними полями угорі на сторінці (рис. 4.12).

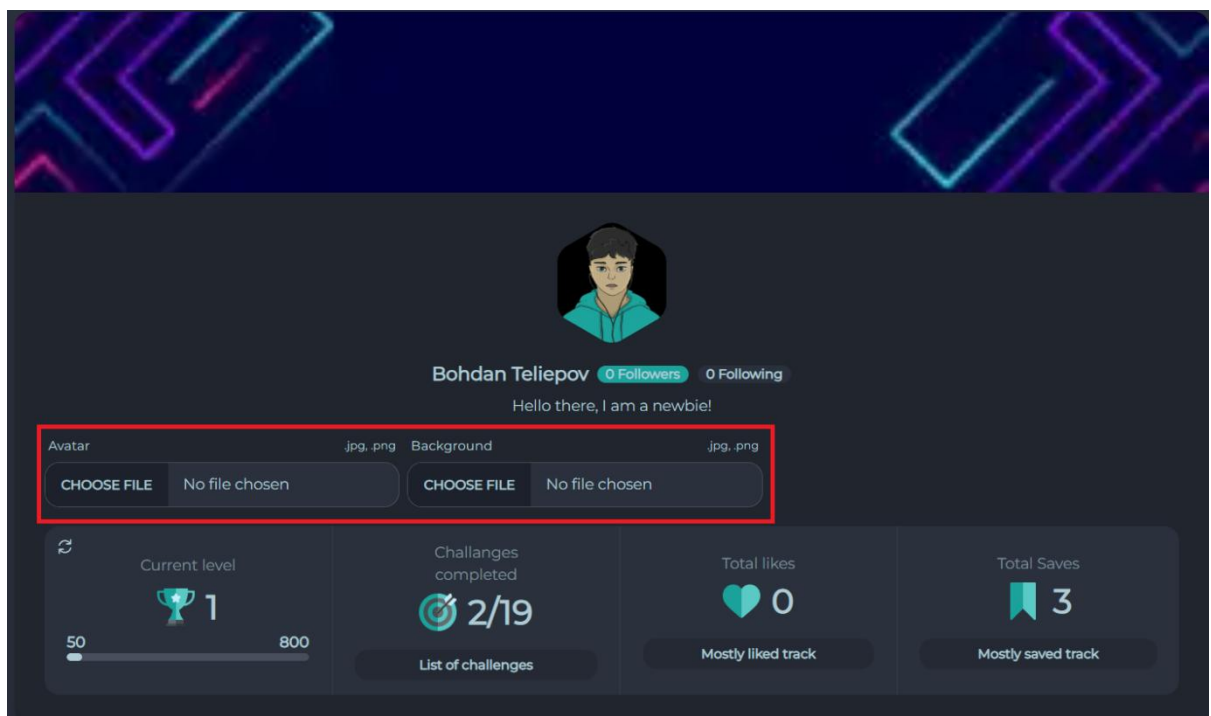


Рисунок 4.12 - Можливість зміни зображень у профілі

Зміна інших параметрів і даних (включаючи соціальні мережі) користувача необхідно використати форму нижче на тій же сторінці (рис. 4.13).

The screenshot shows a settings page with a dark background. At the top, there's an 'Email' section with a message: 'It is highly recommended to link an email address to protect your account!'. Below this is a 'New email' input field with a placeholder 'New email' and an email icon. A red 'Request' button is positioned below the input field. The next section is 'Password', with a message: 'Due to lack of email address, password recovery is not available for your account yet!'. The 'Socials' section follows, featuring 'Connect Twitter' and 'Connect Facebook' buttons, each with a corresponding social media icon and a red 'X' icon to the right. Below these is a grey bar displaying the email 'bogdantelep@gmail.com' with a red 'X' icon to its right.

Рисунок 4.13 - Додаткові можливості по редагуванню

Для додавання треку у закріплене можна визвати модальне вікно у профілі користувача натиснувши кнопку “select” - вибір (рис. 4.14).

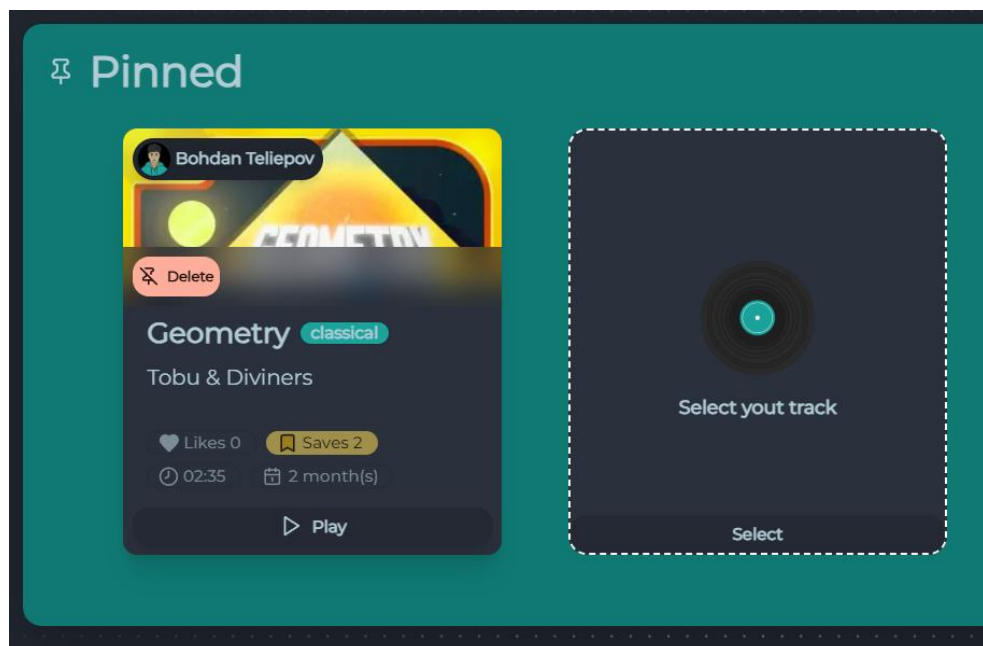


Рисунок 4.14 - Можливість додати трек у закріплене

Після натискання кнопки для вибору треку, відкриється спеціальне модальне вікно для вибору за назвою із свого вивантаженого списку (рис. 4.15).

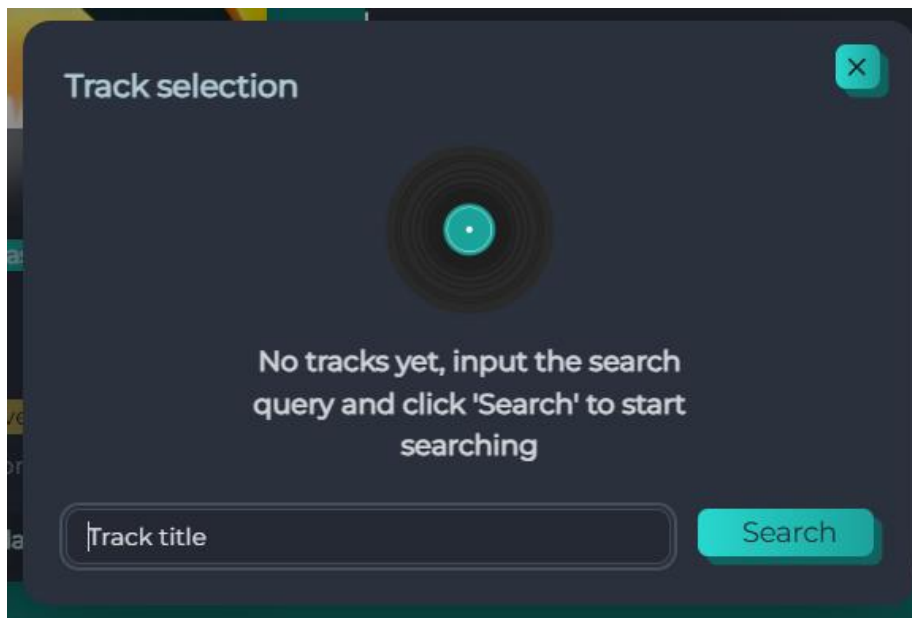


Рисунок 4.15 - Модальне вікно пошуку треків

Для пошуку контенту по застосунку необхідно скористатися полем у лівій панелі на будь-якій сторінці застосунку (рис. 4.16).

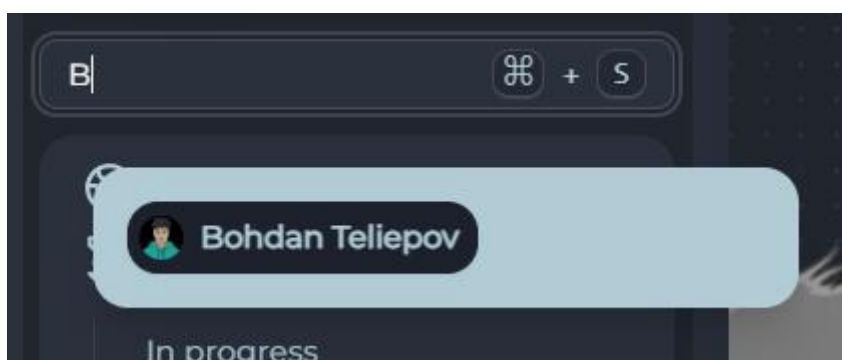


Рисунок 4.16 - Пошук контенту по платформі

Для створення музичного змагання необхідно обрати два треки користувачів (один із них повинен належати авторизованому користувачу за необхідності асоціювати змагання із блокчейн (рис. 4.17).

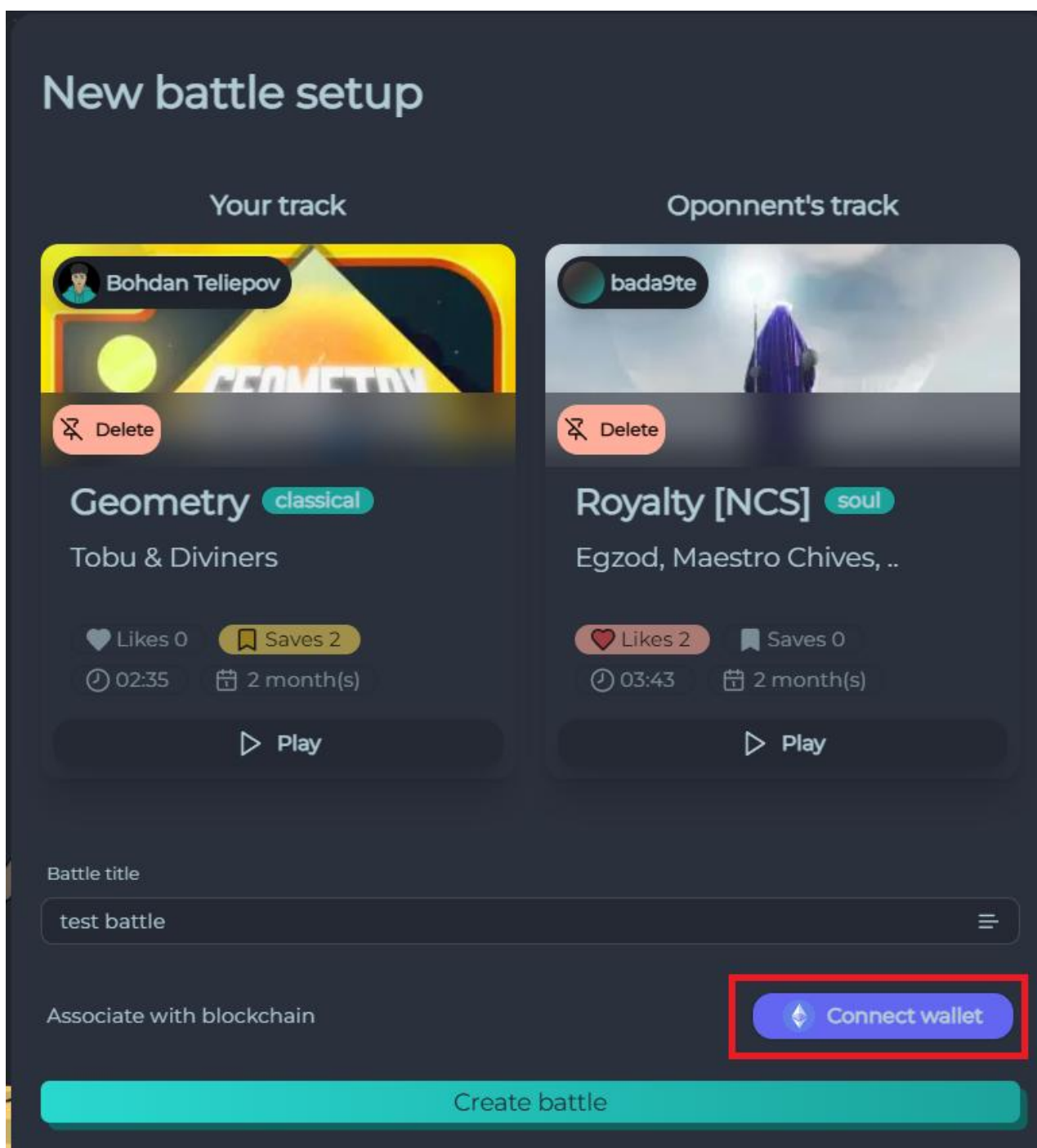


Рисунок 4.17 - Створення змагання

При відсутності підключеного гаманця, інтерфейс програми надасть можливість підключити один з можливих у відповідному модальному вікні (рис. 4.18)

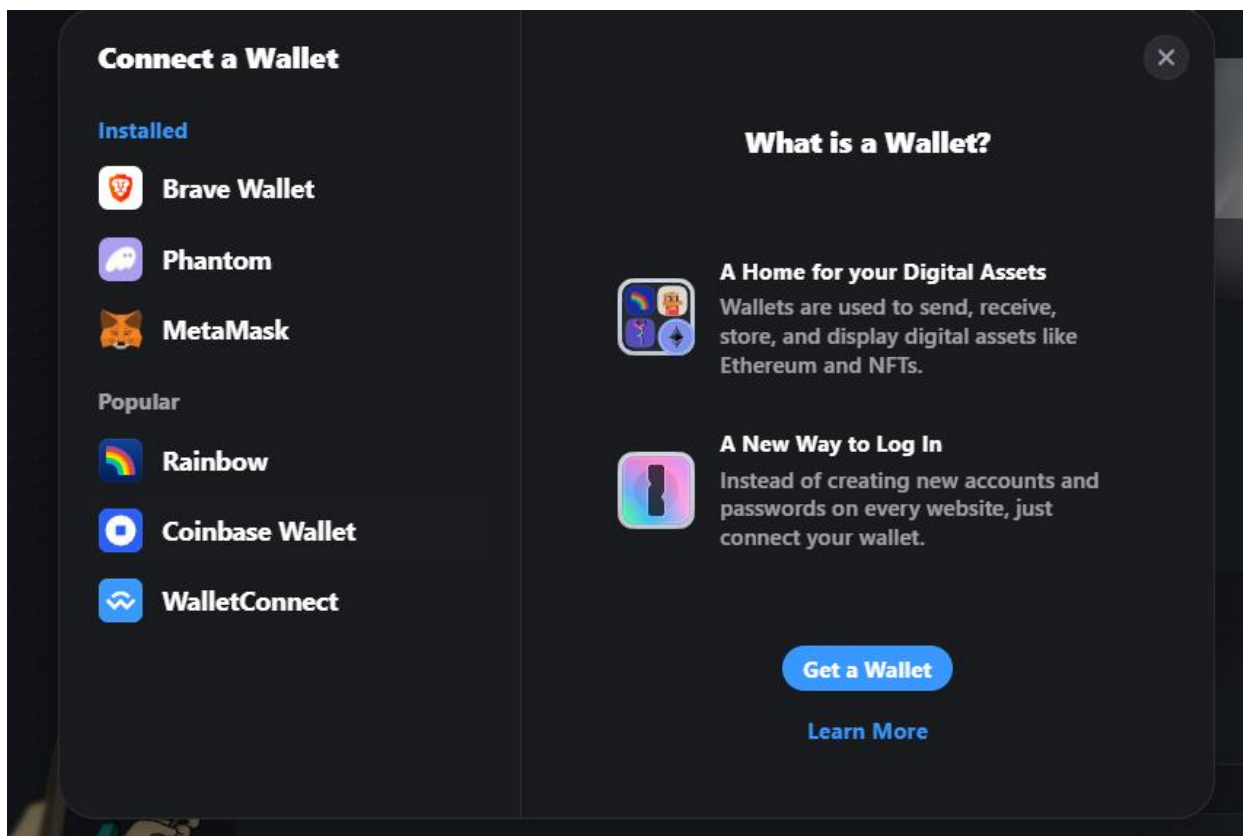


Рисунок 4.18 - Підключення гаманця

Після успішного підключення гаманця користувачу на вибір будуть надані EVM-мережі для створення відповідного змагання (рис. 4.19).

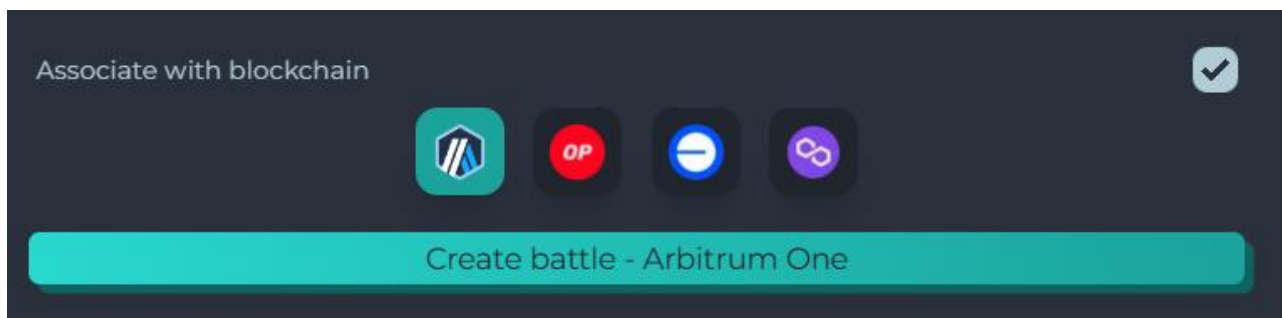


Рисунок 4.19 - Вибір мережі для створення змагання

Функція голосування за найкращий трек у музичних змаганнях (як звичайне голосування, так і через Web-3 USDC (рис. 4.20) якщо змагання прив'язано до Blockchain). Користувач повинен мати можливість голосувати у музичних змаганнях за обраний трек (рис. 4.21).

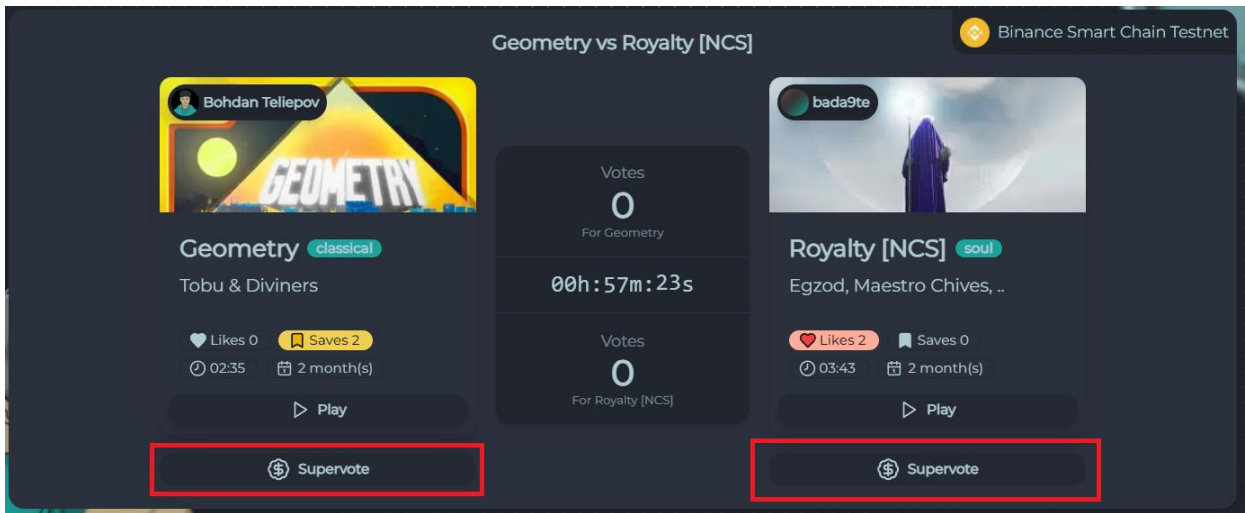


Рисунок 4.20 - Можливість голосувати за треки

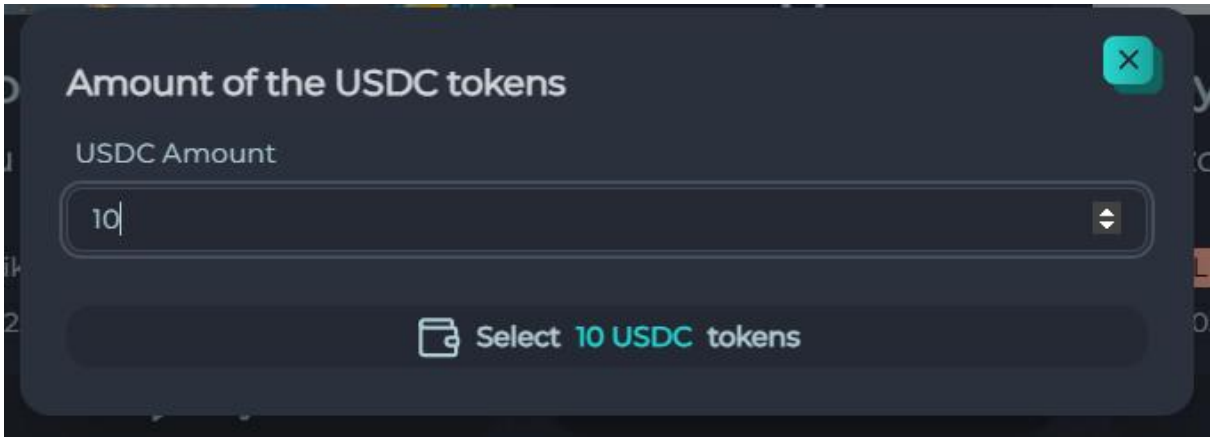


Рисунок 4.21 - Вибір кількості USDC для голосування

Далі, після вибору бажаної кількості USDC для голосування, користувачу буде показано вікно гарантії для підтвердження транзакції (рис. 4.23), при успішному проходженні транзакції, буде нараховано відповідну кількість балів для обраного треку (рис. 4.22).

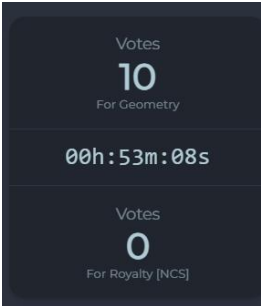


Рисунок 4.22 - Результат успішної транзакції

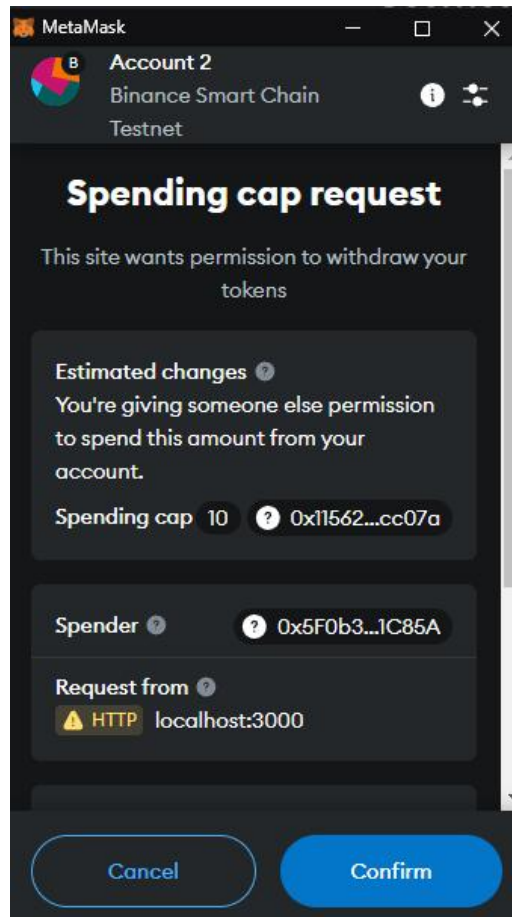


Рисунок 4.22 - Підтвердження транзакції

Для перегляду активних і завершених змагань користувач може використати відповідні елементи інтерфейсу (рис. 4.24).

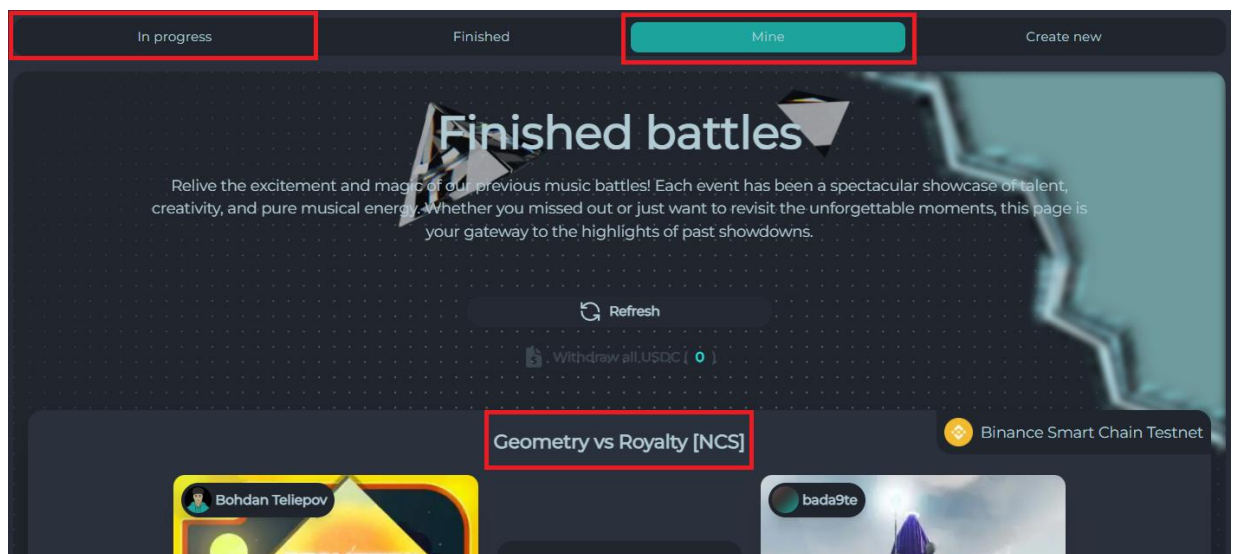


Рисунок 4.24 - Можливість перегляду завершених і активних змагань

Для входу в обліковий запис користувачу надано відповідну форму (рис. 4.25).

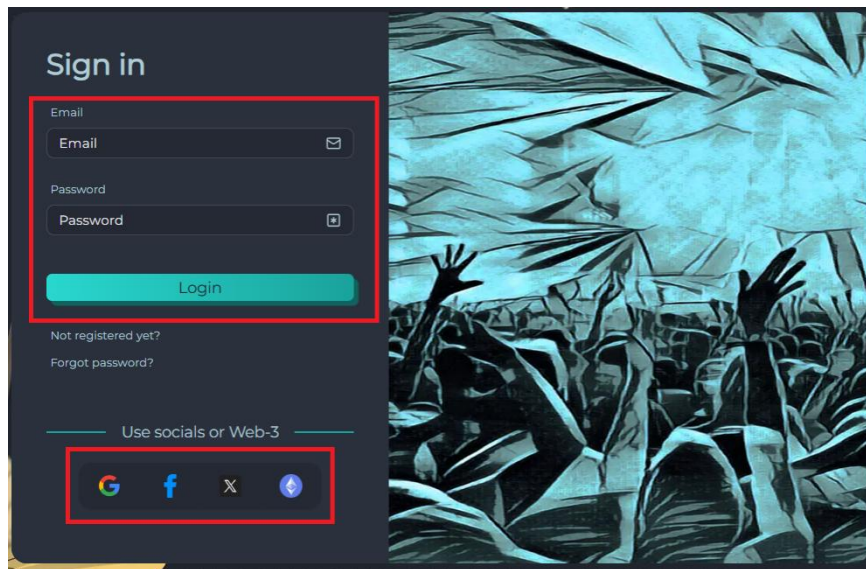
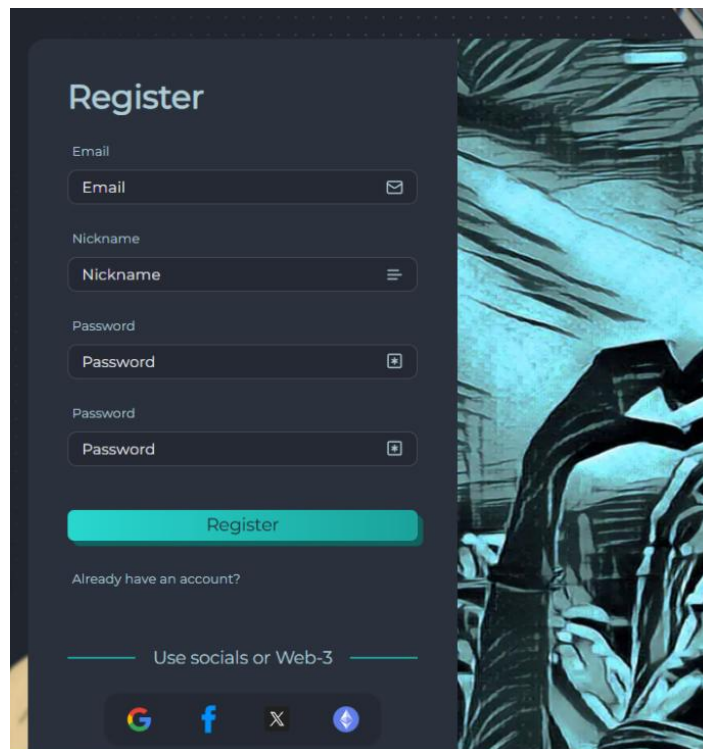
The image shows a 'Sign in' form on a dark background. The form is titled 'Sign in' in white. It contains two input fields: 'Email' and 'Password', both with placeholder text and icons (an envelope for email and a key for password). Below these fields is a blue 'Login' button. Under the button are two links: 'Not registered yet?' and 'Forgot password?'. At the bottom, there is a section titled 'Use socials or Web-3' with four icons: Google, Facebook, X, and a Web3 icon. The entire form is highlighted with a red border. To the right of the form is a stylized illustration of a person's hands reaching up towards a bright, abstract light source.

Рисунок 4.25 - Форма входу в обліковий запис

Для створення нового облікового запису, користувачу надано відповідну форму (рис. 4.26).

The image shows a 'Register' form on a dark background. The form is titled 'Register' in white. It contains four input fields: 'Email', 'Nickname', 'Password', and 'Password' (repeated). The 'Email' field has an envelope icon, the 'Nickname' field has a person icon, and the 'Password' fields have key icons. Below these fields is a blue 'Register' button. Under the button is a link: 'Already have an account?'. At the bottom, there is a section titled 'Use socials or Web-3' with four icons: Google, Facebook, X, and a Web3 icon. The entire form is highlighted with a red border. To the right of the form is a stylized illustration of a person's hands reaching up towards a bright, abstract light source.

Риснок 4.26 - Форма реєстрації нового облікового запису

Для виходу із облікового запису необхідно натиснути відповідну кнопку в інтерфейсі (рис. 4.27).

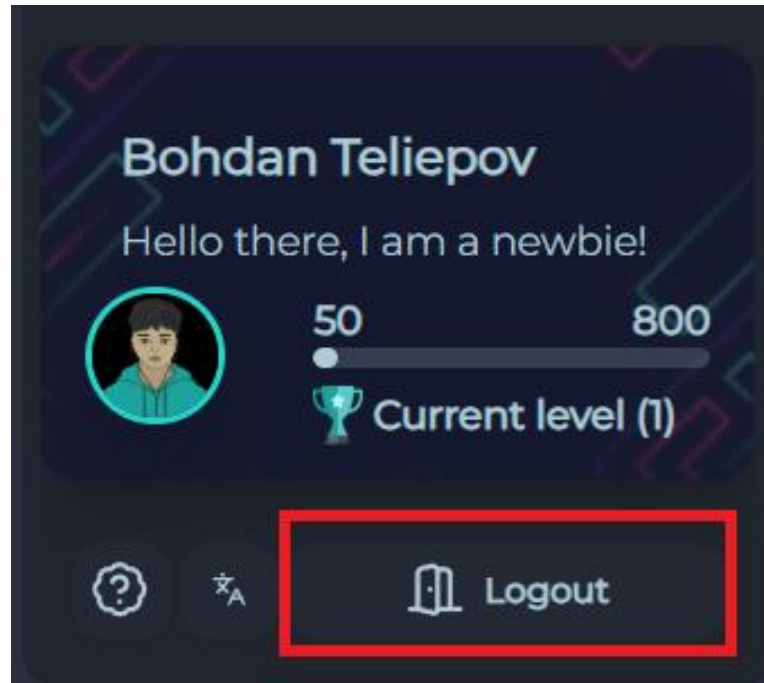


Рисунок 4.27 - Можливість виходу з облікового запису

4.2 Тестування серверної частини

Тестування працездатності серверної частини дуже важливе, адже саме серверна частина містить майже всю логіку функціонування всього застосунку, та напряму впливає на правильність роботи інших компонентів розроблюваного програмного забезпечення, за виключенням частини смарт-контрактів.

Для тестування серверної частини було розроблено 90 тест-кейсів для перевірки працездатності усіх можливих модулів, що напряму взаємодіють із базою даних та відповідають за повертання запитуваної інформації користувачу.

Водночас було перевірено правильність запису даних в БД, проаналізовано та відлагоджено її структуру для більш оптимального по швидкості і зручності варіанту взаємодії із даними.

На рисунку нижче (рис. 4.28), можна побачити результати тестування відповідних модулів системи.

```
PS C:\Users\bohdan\Documents\development\mfnsServer> yarn test
yarn run v1.22.22
$ jest
PASS src/entities/planned-tasks/test/planned-tasks.service.spec.ts (11.999 s)
PASS src/entities/moderations/test/moderations.service.spec.ts (12.026 s)
PASS src/entities/achievements/test/achievements.service.spec.ts (12.04 s)
PASS src/entities/notifications/test/notifications.service.spec.ts (12.058 s)
PASS src/entities/support-requests/test/support-requests.service.spec.ts (12.045 s)
PASS src/entities/playlists/test/playlists.service.spec.ts (12.343 s)
PASS src/entities/reports/test/reports.service.spec.ts (12.558 s)
PASS src/entities/battles/test/battles.service.spec.ts (12.581 s)
PASS src/entities/posts/test/posts.service.spec.ts (12.618 s)
PASS src/entities/users/test/users.service.spec.ts (12.662 s)

Test Suites: 10 passed, 10 total
Tests: 90 passed, 90 total
Snapshots: 0 total
Time: 13.351 s
Ran all test suites.
Done in 14.35s.
```

Рисунок 4.28 - Тестування серверної частини (90 тестів)

За результатами тестування серверної частини, можна зробити висновок, що програмне забезпечення працює вірно і не має помилок в логіці взаємодії із базою даних.

4.3 Тестування частини смарт-контрактів

Тестуючи смарт-контракти необхідно ретельно перевірити правильність роботи взаємодії із користувачами (їх web3-гаманцями) і логіки, що повинна відповідати принципам роботи розроблюваної платформи.

Для тестування смарт-контрактів було створено 2 групи тестів:

- для тестування логіки роботи із змаганнями на платформі - “Battles contract”;

– для перевірки правильності вивантаження смарт-контракту в мережу і перевірки роботи функцій, що доступні тільки власнику контракта - “MusicFromNothing contract”;

Результати тестування смарт-контрактів наведено нижче на рис. 4.29.

```
PS C:\Users\bohdan\Documents\development\mfncContracts> npx hardhat test --network hardhat

Battles contract
  Battle management
    ✓ Should create a battle (886ms)
  Battle voting
    ✓ Should allow users to vote in a battle
    ✓ Should emit a Voted event when a user votes
  Battle withdrawals
    ✓ Should allow users to withdraw tokens if they voted for the winning post (57ms)

MusicFromNothing Contract
  Deployment
    ✓ Should set the right owner (41ms)
    ✓ Should set the correct token address
  Admin Functions
    ✓ Should allow the owner to change the token address (74ms)
    ✓ Should prevent non-owners from changing the token address

8 passing (1s)
```

Рисунок 4.29 - Тестування смарт-контрактів (8 тестів)

Згідно результатам тестування частини самарт-контрактів, можна зробити висновок, що вся логіка відпрацьовує вірно.

4.4 Висновки з розділу

В даному розділі було проведено повне тестування програмного забезпечення потокової передачі музики. Для тестування було спеціально створено юніт-тести, в результаті роботи яких можна підтвердити правильність роботи застосунку.

Під час тестування клієнтської частини було приділено багато часу для відлаштовування інтерфейсу під різні моделі пристроїв і зауваження користувачів.

Для серверної частини було надважливим перевірити правильність оперування даними і роботи із клієнтською частиною.

Частина смарт-контрактів була перевірена на правильність відпрацювання логіки для частин інтерфейсу застосунку, що на пряму взаємодіє із web-3, читаючи дані зі смарт-контракту, або підписуючи транзакції.

Завдяки такому підходу було виявлено та усуното недоліки та програмні помилки, що забезпечило безвідмовну роботу кожної частини розроблюваного програмного забезпечення.

Використовуючи юніт-тестування, у ході розробки було мінімізовано шанси на прояв нових помилок і програмних збоїв в роботі застосунку.

Як результат, було отримано стабільний і відлагоджений застосунок, який відповідає потребам користувачів, правильно працює на пристроях із різною роздільною здатністю і не має недоліків в логіці виконання реалізованих функцій.

ВИСНОВКИ

Виконавши дану роботу, було розроблено вебзастосунок що виконує роль платформи для потокової передачі музики.

Для розробки було використано NestJS для побудови серверної частини і NextJS для клієнтської частини та з'язування логіки із сервером. Утилізацію Web-3 було досягнуто шляхом зберігання файлів користувачів в IPFS, та можливістю взаємодіяти із смарт-контрактами, комп'ютерними алгоритмами що слідкують за угодами в blockchain між користувачами.

При розробці було максимально дотримано MVC архітектурного шаблону проектування систем.

Спроектовано основні моделі для бази даних, що використовуються на серверній частині для маніпуляції контентом платформи та інформацією користувачів. Беручи до уваги той факт що на клієнтській частині можуть знадобитись не всі дані водночас, було інтегровано підтримку GraphQL для гнучкого оперування даними.

Розроблено інтерфейс застосунку, що відповідає стандартам сучасного інтерфейсу подібних програм по стримінгу музики і оперуванню користувачами. Також під час розробки клієнтської частини було створено спеціальні GraphQL запити для отримання і оперування даними з сервера.

Під час розробки смарт-контрактів, використавши мову програмування Solidity, було створено відповідні програмні компоненти для виконання усіх функціональних вимог з технічного завдання, що відносились до цієї частини програмного забезпечення.

Розроблене програмне забезпечення було добре протестовано, завдяки використанню юніт-тестів і відгукам реальних користувачів при взаємодії із інтерфейсом застосунку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Apple Music. New [Electronic resource]. - Access mode: <https://music.apple.com/ua/new>.
2. Spotify. Слухайте мільйони пісень і подкастів безкоштовно [Електронний ресурс]. - Режим доступу: <https://www.spotify.com/ua-uk/free/>.
3. SoundCloud. Discover. Get Discovered [Electronic resource]. - Access mode: <https://soundcloud.com/>.
4. NodeJS. Run JavaScript everywhere [Electronic resource]. - Access mode: <https://nodejs.org/en>.
5. Express. Fast, unopinionated, minimalist web framework for Node.js [Electronic resource]. – Access mode: <https://expressjs.com/>.
6. How to Authenticate and Authorize Users in a Node.js Express App [Electronic resource]. - Access mode: <https://supertokens.com/blog/user-authentication-in-node>.
7. NestJS. Hello, nest! A progressive NodeJS framework for building efficient, reliable and scalable server-side applications. [Electronic resource]. – Access mode: <https://nestjs.com/>.
8. Middleware. How to Build Middleware for Node.js: A Complete Guide [Electronic resource]. – Access mode: <https://www.turing.com/kb/building-middleware-for-node-js>.
9. NextJS. The React Framework for the Web [Electronic resource]. – Access mode: <https://nextjs.org/>.
10. ReactJS. The library for web and native user interfaces [Electronic resource]. - Access mode: <https://react.dev/>.
11. Вступ до JSX [Електронний ресурс]. - Режим доступу: <https://uk.legacy.reactjs.org/docs/introducing-jsx.html>.
12. GraphQL. A query language for your API [Electronic resource]. - Access mode: <https://graphql.org/>.

13. IPFS. Building better web [Electronic resource]. - Access mode: <https://ipfs.tech/>.

14. MongoDB. Non-relational database [Electronic resource]. - Access mode: <https://www.mongodb.com/resources/basics/databases/non-relational>.

15. NextJS. Testing [Electronic resource]. - Access mode: <https://nextjs.org/docs/app/building-your-application/testing>.

16. NestJS. Testing [Electronic resource]. - Access mode: <https://docs.nestjs.com/fundamentals/testing>.

17. Hardhat. Testing contracts [Electronic resource]. - Access mode: <https://hardhat.org/hardhat-runner/docs/guides/test-contracts>.

ДОДАТОК А
Технічне завдання

A.1 Повна назва проекту

Платформа для потокової передачі музики із інтеграцією Web-3 технологій.

A.2 Мета розробки і призначення

A.2.1 Мета розробки

Основною метою є створення стримінгового сервісу музики із підтримкою кросплатформності та інтеграцією Web-3 (наприклад при зберіганні файлів користувача через IPFS, включаючи аудіо).

A.2.2 Функціональне призначення

Функціональним призначенням майбутньої програми буде потокове відтворення аудіо, виступаючи в ролі музикального програвача та інтерактивної платформи шерингу аудіо та профілів музикантів.

A.2.3 Експлуатаційне призначення

Програма не має експлуатуватися в особистих інтересах користувача.

A.3 Посилки до розробки

A.3.1 Стислий зміст основних теоретичних посилок до розробки на основі проведених пошуків.

Зміст основних теоретичних посилок до розробки наведено нижче:

– вебтехнології - використання HTML, CSS та JavaScript для реалізації інтерфейсу та логіки додатка;

- асинхронність та AJAX - застосування асинхронних запитів для оптимізації продуктивності та покращення відгуків;
- архітектурні принципи - використання паттернів проектування, в нашому випадку MVC (Model – View - Controller) для організації коду та розділення логіки;
- інтерфейс користувача та дизайн - розробка інтуїтивно зрозумілого та зручного інтерфейсу на ReactJS для полегшення навігації та використання.

На мою думку існує багато аналогів платформ потокової передачі музики, побудованих на інших технологіях (наприклад .NET), але усі вони не мають певної унікальності, яку у нашому випадку можна досягти шляхом інтеграції застосунку із Web-3, додаючи інтерактивності нового рівня (Blockchain).

А.4 Вимоги до програми

А.4.1 Вимоги до функціональних характеристик

Групи користувачів

Платформа має використовуватись тільки одним типом користувача - даний користувач має змогу програвати треки, вивантажувати треки на платформу, завантажувати треки з платформи, оперувати плейлистами, створювати музикальні змагання із Web-3, редагувати власний профіль, переглядати профілі інших користувачів, додавати треки у обране, та закріплювати найкращі у власному профілі.

Вимоги до складу виконуваних функцій

Програмне забезпечення має задовільняти можливість виконання визначених нижче функцій:

- функція прослуховування треків. Потрібно надати користувачу можливість прослуховувати треки;

- функція додавання треків у обране. Користувач повинен мати можливість зберігати улюблені треки;
- функції оперування плейлистами. У користувача повинен бути функціонал по створенню і редагуванню плейлистів;
- функція редагування власних треків. Користувач повинен мати можливість редагувати власні треки (наприклад назву, опис);
- функція створення постів. У користувача повинен бути доступ до створення постів (вивантаження треків у IPFS);
- функція редагування профілю. Користувач повинен мати можливість видозмінювати власну сторінку у застосунку;
- функція додавання треків у закріплене. Користувач повинен мати можливість закріплювати на його думку найкращі треки;
- функції пошуку треків і користувачів. Необхідно надати інтерфейс для пошуку треків і користувачів;
- функція створення музичних змагань (із можливістю прив'язки до Blockchain). Користувач повинен мати можливість створювати музичні змагання;
- функція голосування за найкращий трек у музичних змаганнях (як звичайне голосування, так і через Web-3 USDC якщо змагання прив'язано до Blockchain). Користувач повинен мати можливість голосувати у музичних змаганнях за обраний трек;
- функція перегляду активних і завершених змагань. Користувач повинен мати можливість переглядати завершені змагання, які певним чином асоційовані із ним;
- функції входу до системи. Необхідно надати функціонал для виконання входу до системи як через логін-пароль, так і через соціальні мережі;
- функція створення облікового запису. У користувача має бути можливість створення акаунта;

– функція виходу із системи. Необхідно надати можливість виходжу із облікового запису.

Вимоги до організації вхідних даних

Вхідні дані в контексті платформи - це ті, які користувачі власноруч додають до платформи і потім взаємодіють із ними.

Вимоги до організації вихідних даних

Вихідні дані це та інформація, яку бачить користувач на екрані монітора; згруповану та виведену за допомогою інтерфейсу платформи.

Вимоги до тимчасових характеристик

До тимчасових характеристик відносяться такі параметри (види даних): треки - вивантажені користувачами аудіо, плейлісти та музичні змагання, даний перелік даних може бути змінено у ході роботи застосунку.

A.4.2 Вимоги до надійності

Вимоги до забезпечення надійного функціонування програми

Надійне функціонування програми повинно бути забезпечено командою розробників, або власником платформи у сукупності організаційно-технічних пунктів, описаних нижче:

- підтримка працездатності і доступності серверів;
- підтримка доступності інтерфейсу (клієнтської частини) у всіх країнах світу;
- регулярне виконання вимог ДСТУ 3396.0-96 захисту інформації;
- виконання перевірок серверів на наявність вірусів і бекдорів.

Збійні ситуації

Наступні збійні ситуації мають оброблятися програмою:

- відсутність зв'язку із сервером (включаючи обробку усіх неуспішних запитів до сервера і виводу повідомлення про помилку);
- обробка невалідних даних.

A.4.3 Вимоги до апаратно-технічних засобів

Для якісної роботи програми рекомендується персональний комп'ютер до складу якого входять наступні компоненти та технічні засоби:

- процесор AMD Bristol Ridge A6-9500 3.5GHz/1MB (AMD Radeon R5 Series);
- оперативна пам'ять DDR3 - 4 Гб;
- об'єм логічної пам'яті HDD - 64 Гб;
- інструменти вводу інформації - клавіатура та миша;
- засоби виводу інформації - монітор;
- рекомендована операційна система – Ubuntu 22.04.

A.5 Основні обмеження на установку і використання програми

Для функціонування програми необхідно мати будь-який сучасний браузер із підтримкою JS (es8), додатково встановити NodeJS – платформу для виконання високопродуктивних мережевих застосунків та самого JS коду поза межами браузера, і NPM (Node Package Manager).

ДОДАТОК Б**Код програми (основні компоненти)**

Б.1 Серверна частина

У наведеному коді нижче, показано точку входу в серверну частину застосунку.

Main.ts - головний виконуваний файл серверної частини.

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import * as cookieParser from 'cookie-parser';
import * as compression from 'compression';
import * as session from 'express-session';
import * as express from "express";
import * as path from "path";
require("dotenv").config();
```

```
async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  app.use(compression());

  app.use(
    session({
      secret: process.env.SESSION_SECRET,
      resave: false,
      saveUninitialized: false,
      cookie: { secure: false },
    })
  );

  app.use(cookieParser());

  app.enableCors({
    origin: process.env.CLIENT_BASE,
```

```

    credentials: true,
  });

  app.use(express.static(path.join(__dirname, 'public')));

  await app.listen(8000);
}

bootstrap();

```

App.module.ts - файл імпорту і ініціалізації усіх модулів.

```

import { Module } from '@nestjs/common';
import { UsersModule } from './entities/users/users.module';
import { GraphQLModule } from '@nestjs/graphql';
import { ApolloDriver, ApolloDriverConfig } from '@nestjs/apollo';
import { upperDirectiveTransformer } from './common/directives/upper-case-directive';
import { ModerationsModule } from './entities/moderations/moderations.module';
import { SupportRequestsModule } from './entities/support-requests/support-requests.module';
import { ReportsModule } from './entities/reports/reports.module';
import { PostsModule } from './entities/posts/posts.module';
import { PlaylistsModule } from './entities/playlists/playlists.module';
import { NotificationsModule } from './entities/notifications/notifications.module';
import { BattlesModule } from './entities/battles/battles.module';
import { MongooseModule } from '@nestjs/mongoose';
import { AuthModule } from './auth/auth.module';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { EmailModule } from './utils/email/email.module';
import { MailerModule } from '@nestjs-modules/mailer';
import { ScheduleModule } from '@nestjs/schedule';
import { TasksModule } from './utils/tasks/tasks.module';
import { PlannedTasksModule } from './entities/planned-tasks/planned-tasks.module';
import { join } from 'path';
import { EjsAdapter } from '@nestjs-modules/mailer/dist/adapters/ejs.adapter';
import { MulterModule } from '@nestjs/platform-express';
import { AchievementsModule } from './entities/achievements/achievements.module';

```

```

import { OnApplicationBootstrapService } from './utils/init/on-bootstrap';

@Module({
  imports: [
    // ENV
    ConfigModule.forRoot({
      isGlobal: true,
    }),
    // MONGODB
    MongooseModule.forRootAsync({
      imports: [ConfigModule],
      useFactory: async(configService: ConfigService) => ({
        uri: configService.get('MONGO_URL')
      }),
      inject: [ConfigService]
    }),
    // SCHEDULE
    ScheduleModule.forRoot(),
    TasksModule,
    PlannedTasksModule,
    // GQL
    GraphQLModule.forRoot<ApolloDriverConfig>({
      driver: ApolloDriver,
      typePaths: ['./**/*.graphql'],
      transformSchema: schema => upperDirectiveTransformer(schema, 'upper'),
      installSubscriptionHandlers: true,
      playground: true,
    }),
    // FILE UPLOAD
    MulterModule.registerAsync({
      imports: [ConfigModule],
      useFactory: async(configService: ConfigService) => ({
        dest: configService.get('MULTER_DEST'),
      }),
      inject: [ConfigService]
    )
  ]
})

```

```

    }),
    // EMAILS
    MailerModule.forRootAsync({
      imports: [ConfigModule],
      useFactory: async(configService: ConfigService) => ({
        transport: {
          host: configService.get('EMAIL_HOST'),
          secure: true,
          tls: {
            ciphers: "SSLv3",
          },
          requireTLS: true,
          port: 465,
          debug: true,
          connectionTimeout: 10000,
          auth: {
            user: configService.get('EMAIL_USERNAME'),
            pass: configService.get('EMAIL_PASSWORD'),
          },
        },
        template: {
          dir: join(__dirname, 'utils', 'email', 'templates'),
          adapter: new EjsAdapter(),
          options: {
            strict: false,
          },
        },
      }),
      inject: [ConfigService]
    }),
    // ENTITIES
    UsersModule,
    ModerationsModule,
    SupportRequestsModule,
    ReportsModule,

```

```

    PostsModule,
    PlaylistsModule,
    NotificationsModule,
    BattlesModule,
    AuthModule,
    EmailModule,
    AchievementsModule,
  ],
  providers: [OnApplicationBootstrapService]
})
export class AppModule {}

```

Б.2 Клієнтська частина

У наведеному коді нижче показано головний файл клієнтської частини.

```

"use client"
import "./globals.css";
import { TLang } from "@app/types/language";
import ReduxProvider from "./lib/redux/provider";
import { SnackbarProvider } from 'notistack';
import { ApolloWrapper } from "./lib/apollo/apollo-wrapper";
import RainbowkitAppProvider from "./lib/rainbowkit/provider";
import { Montserrat } from "next/font/google";

import { MFNSuccess } from "./lib/notistack/success";
import { MFNDefault } from "./lib/notistack/default";
import { MFNError } from "./lib/notistack/error";
import { MFNWarning } from "./lib/notistack/warning";

const font = Montserrat({
  subsets: ['latin', 'cyrillic'],
  weight: '400'

```

```

});

export default function RootLayout({
  children,
  params
}: Readonly<{
  children: React.ReactNode;
  params: { lang: TLang }
}>) {
  return (
    <html lang={params.lang} data-theme="dim">
      <body className={font.className}>
        <RainbowkitAppProvider>
          <ApolloWrapper>
            <SnackbarProvider
              anchorOrigin={{
                vertical: 'bottom',
                horizontal: 'center',
              }}
              maxSnack={5}
              Components={{
                default: MFNDefault,
                success: MFNSuccess,
                error: MFNError,
                warning: MFNWarning,
              }}
            >
              <ReduxProvider>
                {children}
              </ReduxProvider>
            </SnackbarProvider>
          </ApolloWrapper>
        </RainbowkitAppProvider>
      </body>
    </html>
  );
}

```

```
);
}
```

Б.3 Частина смарт-контрактів

У наведеному коді нижче показано головний файл частини смарт-контрактів.

```
// SPDX-License-Identifier: MIT
// Compatible with OpenZeppelin Contracts ^5.0.0

pragma solidity ^0.8.24;

import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
import "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

import "./content/Battles.sol";

contract MusicFromNothing is Initializable, OwnableUpgradeable, UUPSUpgradeable, Battles {
    IERC20 public tokenUSDC;

    event EtherWithdrawn(address indexed owner, uint256 amount);
    event TokenAddressChanged(address newAddress);

    function initialize(address initialOwner, address tokenAddress) initializer public {
        __Ownable__init(initialOwner);
        tokenUSDC = IERC20(tokenAddress);
    }

    function _authorizeUpgrade(address newImplementation)
        internal
        onlyOwner
```

```

    override
    {}

    function _getToken() internal view override(Battles) returns (IERC20) {
        return tokenUSDC;
    }

    function depositUSDCTokens(uint256 amount) public {
        require(tokenUSDC.balanceOf(msg.sender) >= amount, "Insufficient token balance");
        tokenUSDC.transferFrom(msg.sender, address(this), amount);
        emit TokensDeposited(msg.sender, amount);
    }

    function withdrawAllEther() public onlyOwner {
        uint256 contractBalance = address(this).balance;
        require(contractBalance > 0, "No Ether available to withdraw");

        payable(owner()).transfer(contractBalance);
        emit EtherWithdrawn(owner(), contractBalance);
    }

    function withdrawUSDCTokens(uint256 amount) public onlyOwner {
        require(tokenUSDC.balanceOf(address(this)) >= amount, "Insufficient token balance");
        tokenUSDC.transfer(owner(), amount);
    }

    function setUSDCToken(address tokenAddress) public onlyOwner {
        tokenUSDC = IERC20(tokenAddress);
        emit TokenAddressChanged(tokenAddress);
    }
}

```

ДОДАТОК В
Слайди презентації



Рисунок В.1 - Титульний слайд

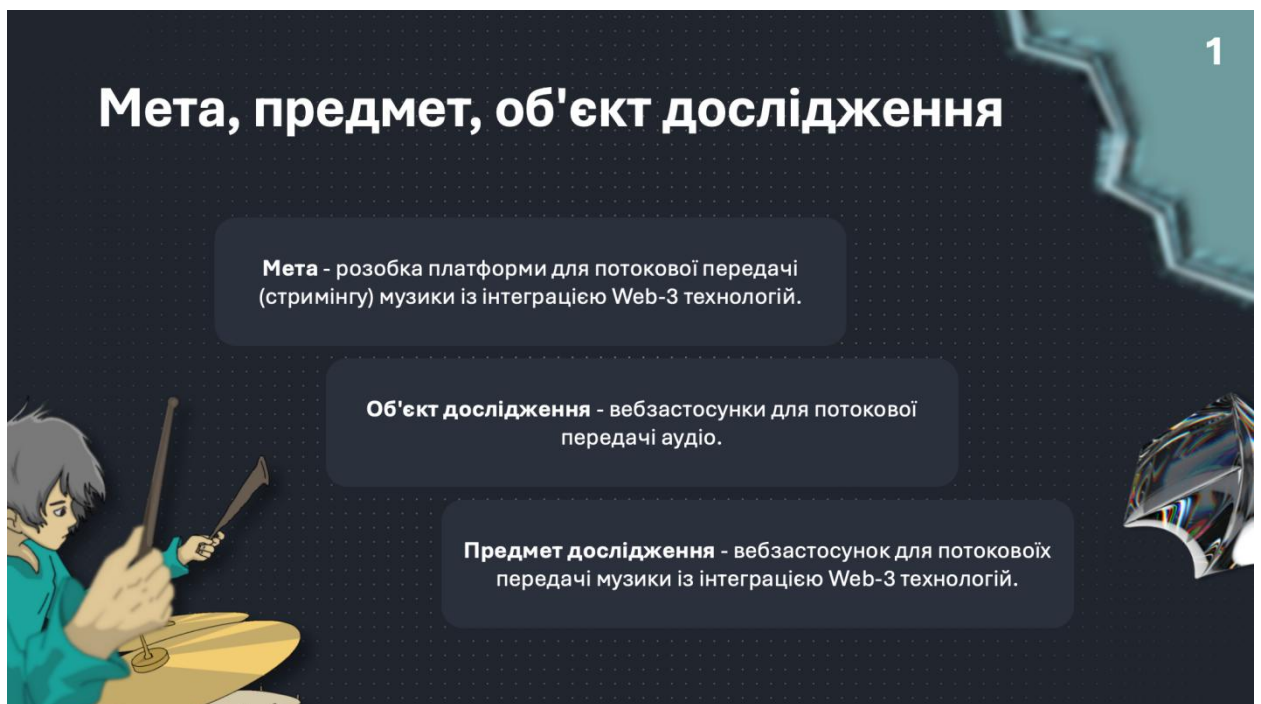


Рисунок В.2 - Слайд 1

2

Огляд і порівняння аналогів

Назва застосунку	Apple Music	Spotify	Sound Cloud
Зручність використання	4	4	3
Кросплатформність	+	+	+
Тип підписки	paid	free	free
Розмір застосунку	>300 MiB (Windows)	>100 MiB	-
Актуальність інтерфейсу	4	5	3




Рисунок В.3 - Слайд 2

3

Постановка задачі

Спроекувати програмне забезпечення для потокової передачі (стримінгу) музики

Розробити програмне забезпечення (платформу) для потокової передачі аудіо

Протестувати розроблене програмне забезпечення



Рисунок В.4 - Слайд 3

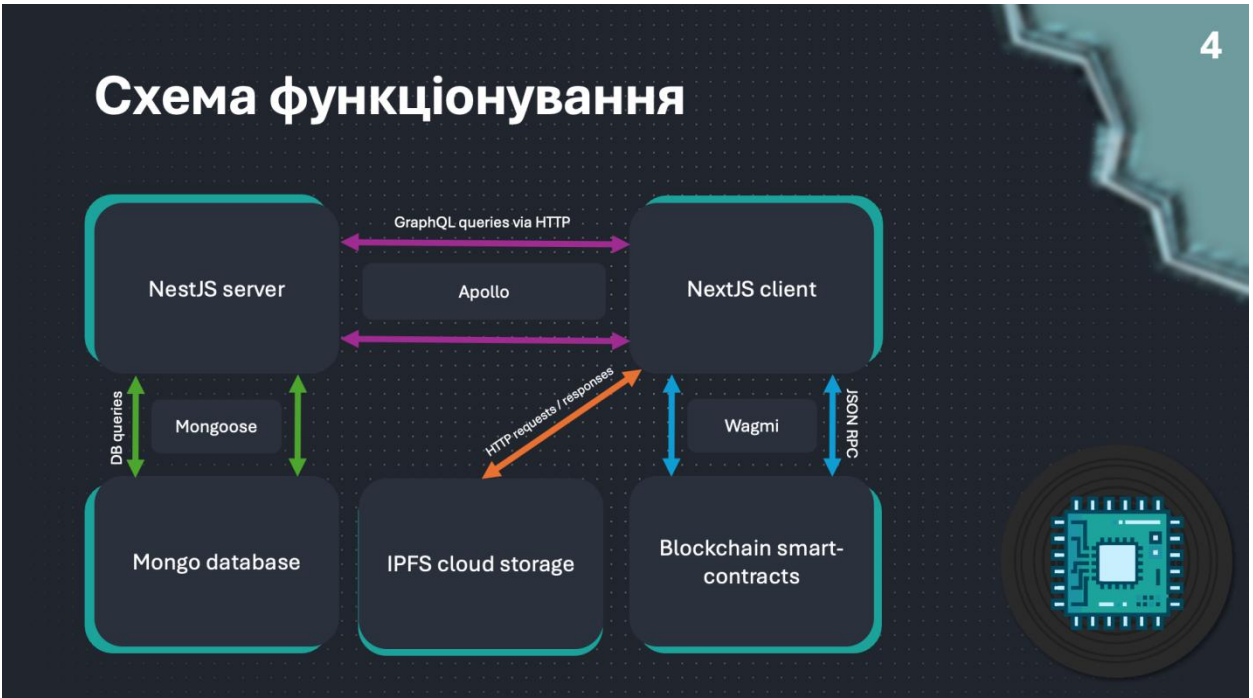


Рисунок В.5 - Слайд 4

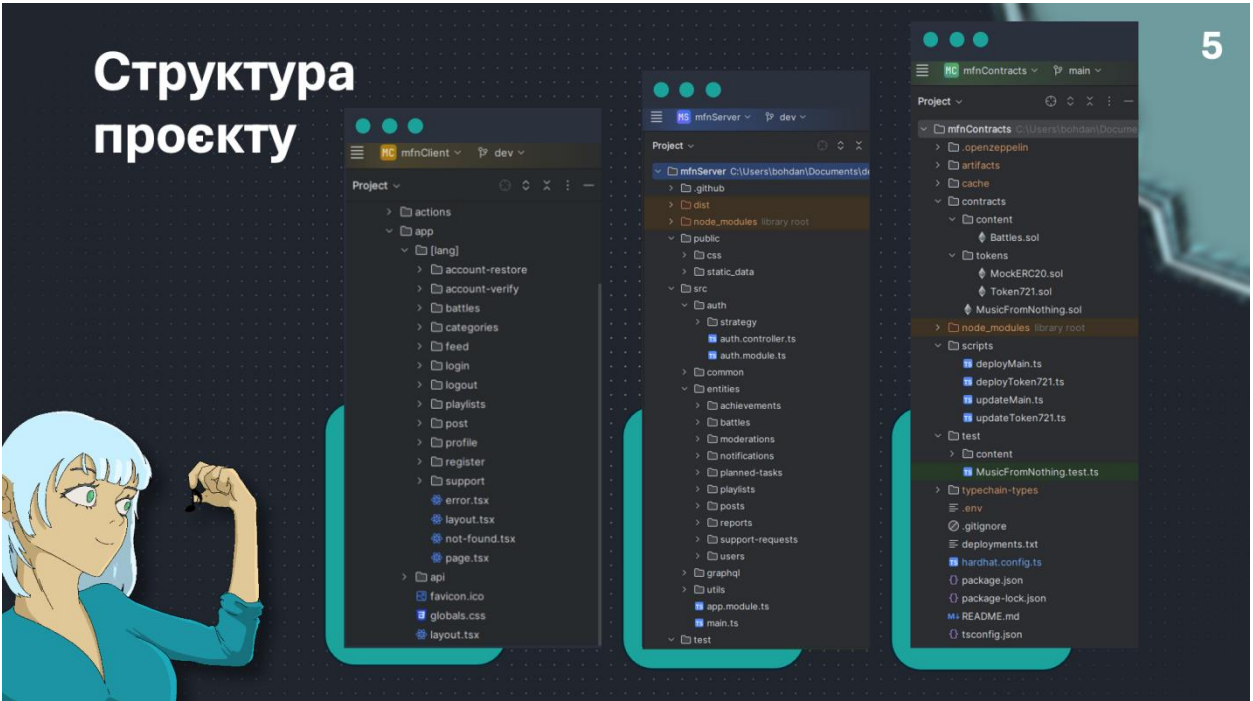


Рисунок В.6 - Слайд 5

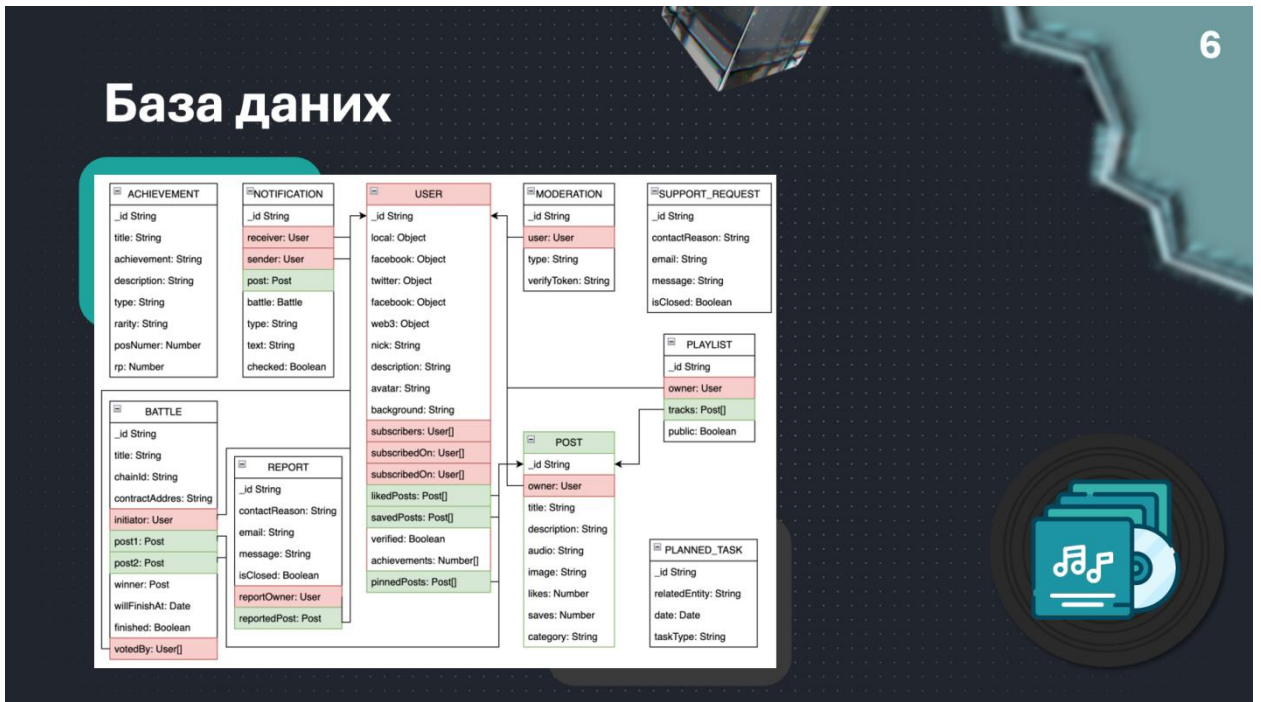


Рисунок В.7 - Слайд 6

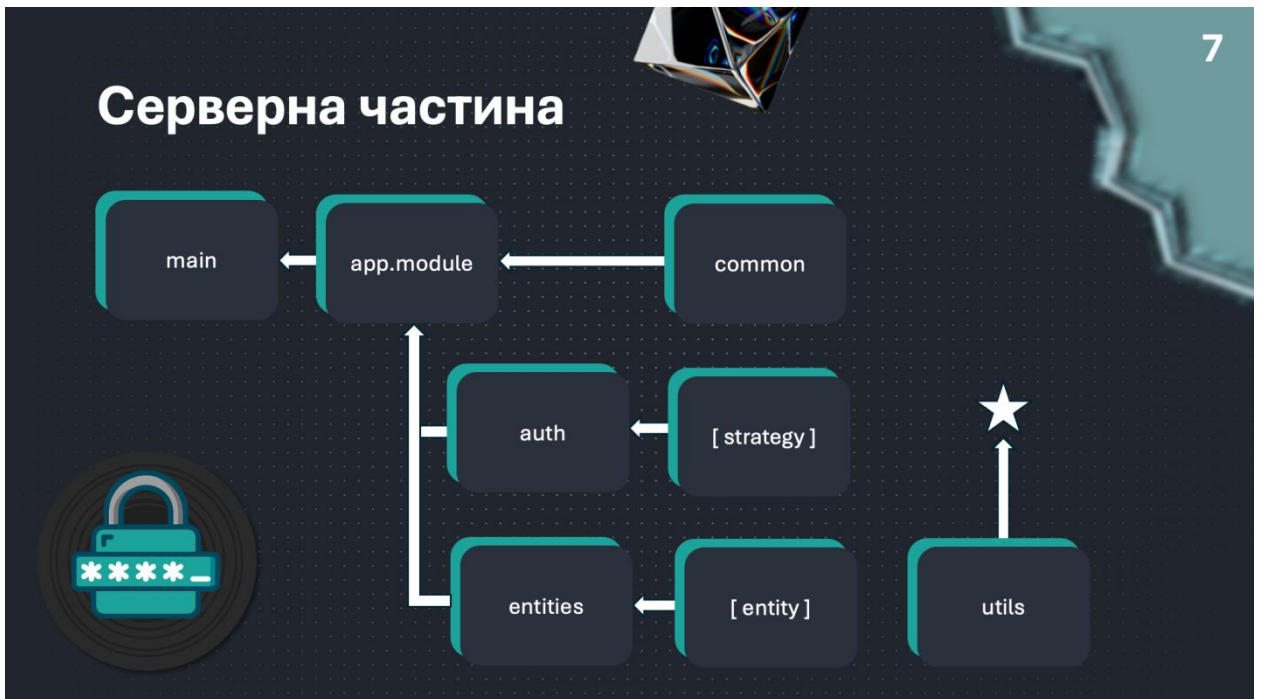


Рисунок В.8 - Слайд 7



Рисунок В.9 - Слайд 8

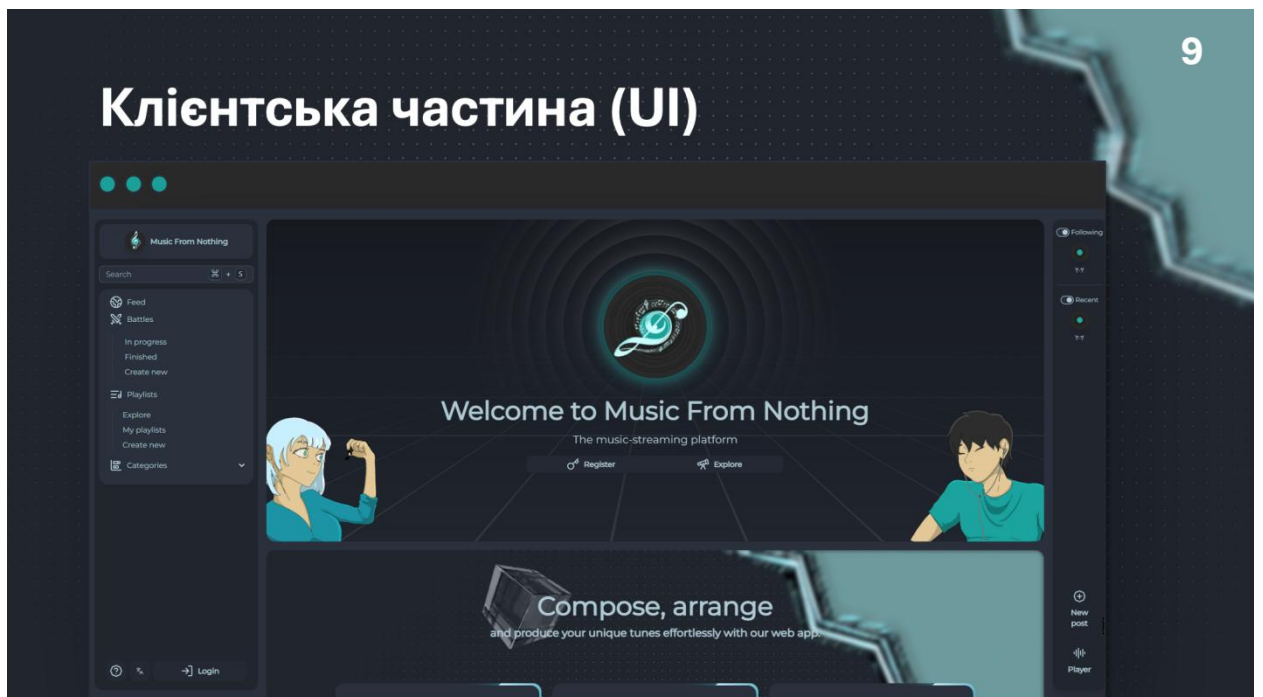


Рисунок В.10 - Слайд 9

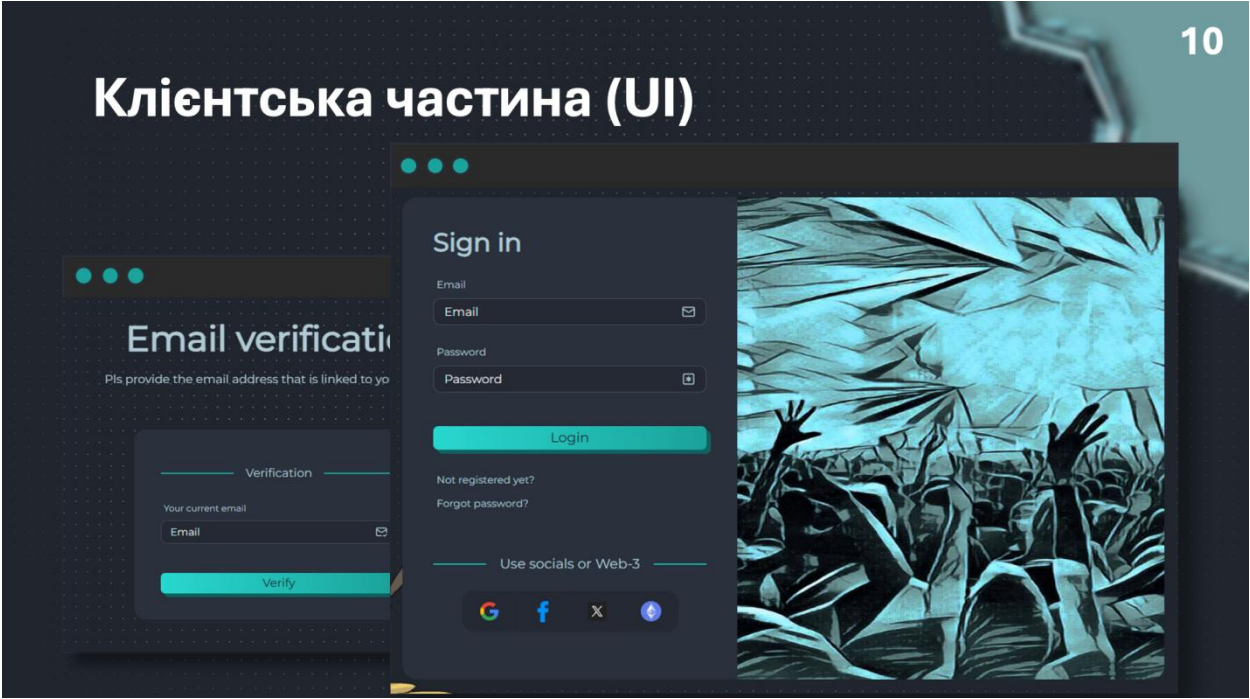


Рисунок В.11 - Слайд 10

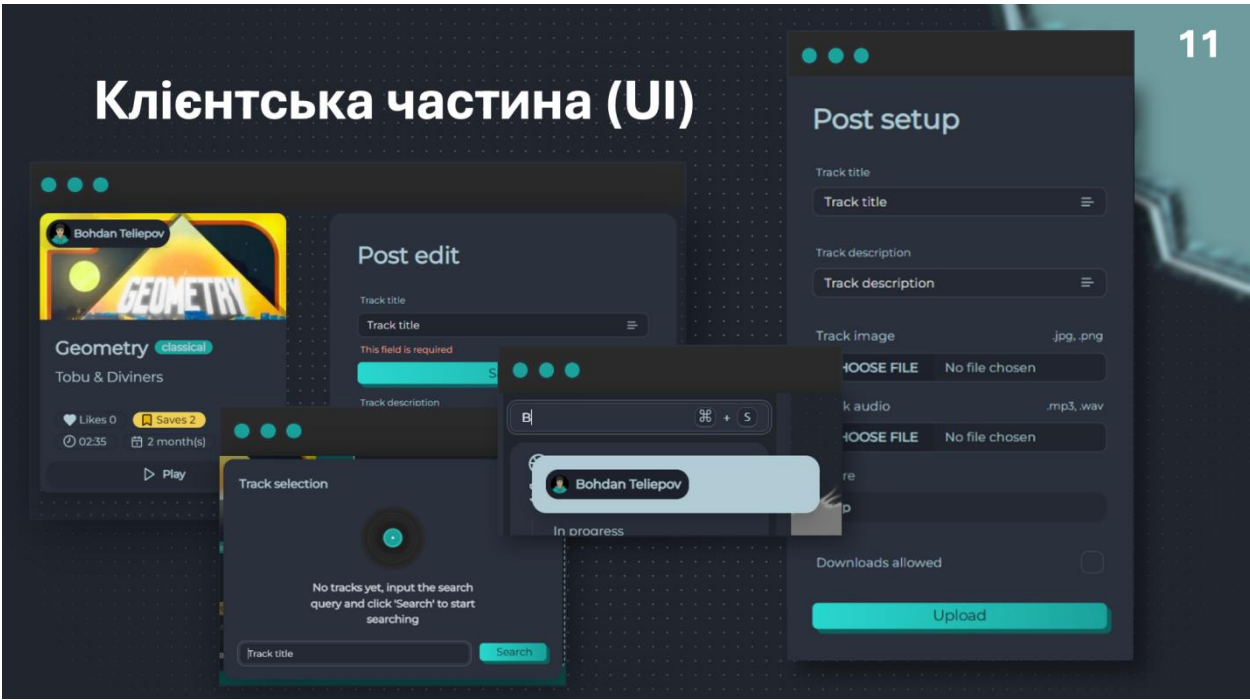


Рисунок В.12 - Слайд 11



Рисунок В.13 - Слайд 12

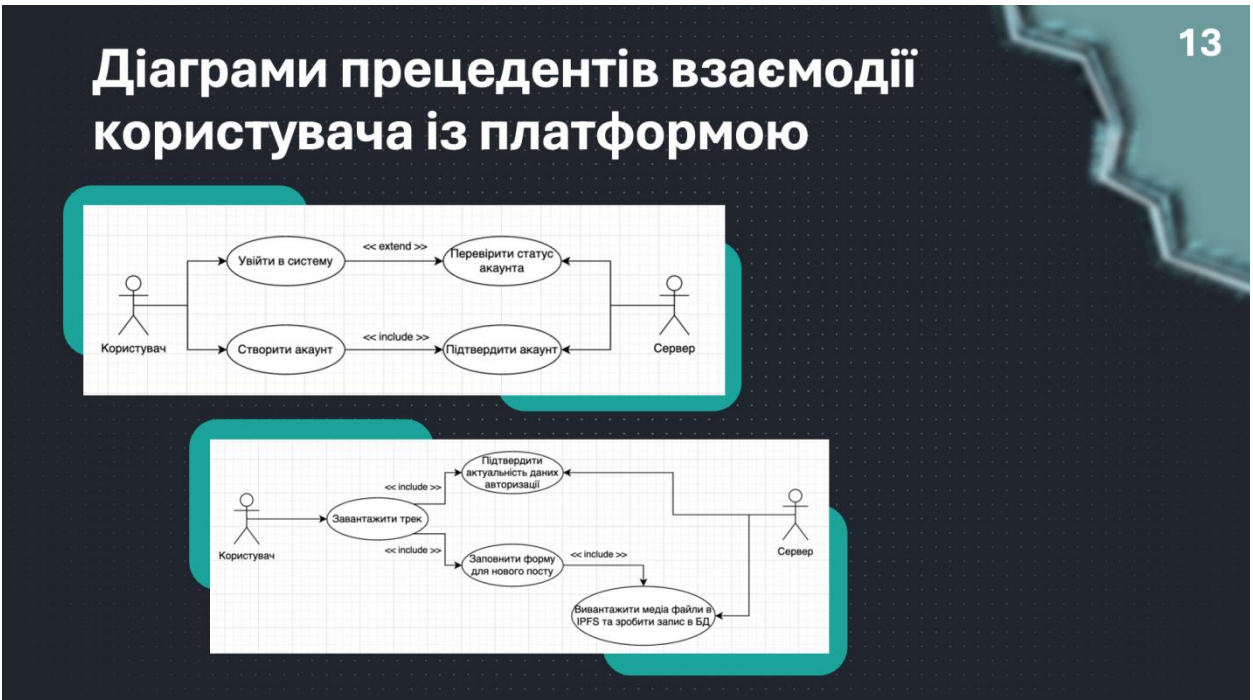


Рисунок В.14 - Слайд 13

16

Висновки

- Оглянуто аналоги програмних реалізацій сервісів (платформ) по потоковій передачі аудіо
- Спроектовано та створено базу даних для роботи із контентом платформи
- Фінальну версію застосунку протестовано через unit-тестування
- Розроблено платформу "Music From Nothing"

MUSIC FROM NOTHING

The slide features a dark blue background with a light blue dotted pattern. A jagged, light blue line runs along the top right and bottom right edges. The text is in white, and the platform name is in a bold, light blue font.

Рисунок В.17 - Слайд 16