Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Бакалавр
(назва освітнього ступеня)

на тему: Розробка веб-застосунку «музична платформа» з використанням Next.js та Nest.js

Виконав(ла): студент(ка) 4 курсу, групи СП-42
спеціальності 121 «Інженерія програмного Забезпечення»
(шифр і назва спеціальності)

_____ Тихович Б. П.
(підпис) (прізвище та ініціали)

Керівник _____ _____
(підпис) (прізвище та ініціали)

Нормоконтроль _____ _____
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ _____
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
2024

АНОТАЦІЯ

Кваліфікаційна робота бакалавра за спеціальністю 121 – Інженерія програмного забезпечення. Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-42, 2024 рік. . Пояснювальна записка до кваліфікаційної роботи на здобуття освітнього ступеню «бакалавр» містить: 61 с., 19 рис., 1 додаток, презентація, 1 таблиця.

Тема: "Розробка веб-застосунку музична платформа з використанням Next.js та Nest.js.

У кваліфікаційній роботі продемонстровано цикл розробки повноцінного веб-застосунку, також результати розробки та реалізації стратегії, обраної в програмі, а також шаблони та методології, що використовуються для реалізації. Вибір інструменту для реалізації ідеї в системному рішенні пояснюється і виводиться.

Ключові слова: веб-застосунок, Nest.js, Next.js, FSD, MVC, проєктування, розробка.

ABSTRACT

Bachelor's Qualification Paper in Specialty 121 – Software Engineering. Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, Group SP-42, 2024.

Explanatory note to the qualification paper for obtaining the educational degree "Bachelor" includes: 61 pages, 19 figures, 1 appendix, presentation, 1 table.

Topic: "Development of a Music Platform Web Application Using Next.js and Nest.js."

The qualification paper demonstrates the development cycle of a full-fledged web application, as well as the results of the development and implementation of the strategy chosen in the program, and the patterns and methodologies used for implementation. The choice of tools for implementing the idea in a system solution is explained and derived.

Keywords: web application, Nest.js, Next.js, FSD, MVC, designing, development.

ЗМІСТ

ВСТУП 7

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 8

1.1 Постановка задачі 8

1.2 Вибір технологій розробки 9

1.3 Аналіз вимог до ПЗ 11

2 ПРОЄКТУВАННЯ ТА КОНСТРУЮВАННЯ СИСТЕМИ 14

2.1 Опис акторів та прецедентів ПЗ 14

2.2 Проєктування архітектури 17

2.3 Проєктування БД та сховища файлів 25

2.4 Розробка серверної частини веб-застосунку 28

2.5 Розробка клієнтської частини веб-застосунка 34

3 ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЯ ВИМОГ 43

3.1 Тестування розробленої інформаційної системи 43

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ 48

4.1 Долікарська допомога при шоку 48

4.2 Розрахунок евакуаційних шляхів із виробничих приміщень дільниці (цеху) 54

ВИСНОВОК 56

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 57

ДОДАТКИ 58

Додаток А: Диск з розробленою програмою 59

ВСТУП

Сучасний світ характеризується безпрецедентною кількістю інформації та ресурсів в Інтернеті. Однією з ключових областей, яка активно використовується для вираження емоцій та творчого самовираження, є музика. У цьому контексті важливо мати зручний та ефективний інструмент для пошуку та відтворення музики відповідно до уподобань користувача.

Ця дипломна робота не тільки забезпечує швидкий доступ до різних музичних творів, але й надає інформацію для пошуку музики з урахуванням індивідуальних уподобань та уподобань кожного користувача. Мета цього проекту- створити інноваційний та ефективний інструмент пошуку музики, який враховує найсучасніші технології та тенденції в галузі інформаційних технологій.

Проект заснований на використанні сучасних методів обробки та аналізу даних. При розробці інформаційної системи ми враховуємо принципи зручності використання, високої продуктивності і безпеки, щоб забезпечити задоволеність і комфорт користувачів при використанні системи.

Розробка цієї музичної платформи буде спрямована на простоту використання та ефективність, адаптуючись до потреб сучасних споживачів звуку. Окрім створення практичних інструментів для прослуховування музики, ми прагнемо надати користувачам неперевершене відчуття комфорту та задоволення від використання нашої платформи.

Реалізація проекту не тільки розширить можливості доступу до музичного контенту, а й сприятиме розвитку сучасних інформаційних технологій і поліпшенню користувацького досвіду в області споживання музики.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка задачі

Музична платформа є веб-застосунком в якої на меті надавати користувачам можливість слухати музику, завантажувати музику, ділитися нею, створювати плейлисти, поширювати її та взаємодіяти з іншими користувачами платформи, поширювати власну музику. Платформа має за головну ціль є створення зручного у використанні та функціонального веб-застосунку. Який у собі використовує сучасні технології, які забезпечують високу продуктивність та безпеку.

В основі постановки задачі по створенню проєкту є:

- Проведення аналізу вимог до програмного забезпечення. Цей процес буде включати збір, аналіз та верифікацію вимог. Це дозволить забезпечити відповідність розробленої системи для потреб користувачів та бізнесу.
- Проєктування системи та опис акторів та прецедентів ПЗ. Забезпечить опис основних сценаріїв взаємодії між акторами та системою.
- Розробка архітектури веб-застосунку: Забезпечення модульності, масштабованості та підтримки сучасних підходів до розробки програмного забезпечення.
- Реалізація серверної частини: Використання Node.js та Nest.js для створення серверної логіки, обробки запитів та управління базою даних.
- Розробка клієнтської частини: Використання Next.js для побудови швидкого та інтерактивного інтерфейсу користувача, що забезпечує плавну взаємодію та приємний користувацький досвід.
- Інтеграція з базою даних: Створення ефективної та надійної бази даних для зберігання інформації про користувачів, музику, плейлисти та інші необхідні дані.
- Забезпечення безпеки: Реалізація сучасних методів автентифікації та авторизації користувачів, захист даних та забезпечення конфіденційності.

– Тестування та верифікація: Проведення всебічного тестування для виявлення та виправлення помилок, а також верифікація вимог для забезпечення відповідності розробленого продукту технічним та функціональним вимогам.

1.2 Вибір технології розробки

Для успішної реалізації музичної платформи важливо обрати відповідні технології, які забезпечать стабільність, масштабованість та зручність у розробці та підтримці системи. Після постановки задач проєкту та сучасних технологічних рішень, було прийнято рішення використовувати наступні інструменти та платформи:

Node.js — це середовище виконання JavaScript, яке дозволяє створювати серверні додатки. Воно побудоване на базі движка V8 від Google, що забезпечує високу продуктивність реалізацію неблокуючий ввід та вивід даних. Та за допомогою використання бібліотеки libuv відтворена багато поточність [4].

Основні переваги це - асинхронність, масштабованість, велика екосистема модулів та пакетів через пакетний менеджер(npm), активна спільнота розробників.

Nest.js — це фреймворк для побудови ефективних, надійних та масштабованих серверних додатків. Він використовує TypeScript та надбудовується на базі Express або Fastify [6].

Перевагами даного фреймворку є використання MVC архітектури, підтримка TypeScript, вбудовані механізми для роботи з базами даних, тестування та обробки запитів, великий спектр готових рішень, та добре складена документація для інтеграції з багатьма бібліотеками.

Next.js — це фреймворк для створення React-додатків з підтримкою серверного рендерингу (SSR) та статичної генерації (SSG). Дані технології надають такі переваги підтримка SSR, висока продуктивність, зручна робота з

маршрутизацією, інтеграція з різними API, можливість створення SEO-оптимізованих додатків, оптимізація та прискорення першого відображення сайту [5].

TypeScript — це мова програмування, яка є надбудовою над JavaScript та додає статичну типізацію.

Дані можливості надають підвищення надійності коду, покращення автодоповнення в IDE, полегшення процесу відкладки коду та виявлення помилок на етапі компіляції та написання програми.

PostgreSQL: Реляційна база даних, яка підтримує складні запити, підтримка бізлічі типів даних та робота з великими обсягами даних [7].

Переваги PostgreSQL: Підтримка ACID-транзакцій, висока надійність, потужні можливості для роботи з даними [7].

Архітектура Feature-Sliced Design є підходом до організації фронтенд-коду, який забезпечує модульність, зрозумілість і масштабованість проекту. Вона базується на поділі коду на окремі функціональні області, що дозволяє легко підтримувати і розширювати додаток [9].

Хмарне сховище для зберігання файлів Firebase Storage створене для обміну великими даними у веб-застосунках. Firebase Storage побудовано на базі Google Cloud Storage, що забезпечує надійність, масштабованість та безпеку. Також легко інтегрується з іншими сервісами Firebase, такими як Firebase Authentication та Firebase Firestore, що дозволяє створювати комплексні рішення без зайвих зусиль. Наявність SDK для різних платформ (веб, iOS, Android) спрощує розробку додатків для різних операційних систем.

Завдяки використанню цих технологій та практик, музична платформа буде мати високу продуктивність, надійність та безпеку, а також зручність у розробці та подальшій підтримці. Що дасть хороше підґрунтя для розвитку та просуванню бізнеса на міжнародний ринок для подальшого всеохопленя та залучення спонсорів та бізнес-партнерів, що дозвлить отримати фінансову вигоду.

1.3 Аналіз вимог до ПЗ

Функціональні вимоги

Веб-система повинна забезпечувати наступні функціональні можливості:

– реєстрація користувача;
– авторизація користувача;
– вихід з облікового запису;
– завантаження музики;
– завантаження фотографій;
– видалення музики;
– перегляд детальної інформації про музику;
– додавання музики в список вподобаних;
– створення плейлистів;
– пошук музики;
– можливість прослуховування музики;
– список популярних пісень;
– вибір зображення профілю;
– перегляд списку коментарів під музикою;
– збереження плейлистів;
– поділитись плейлистом;
– поширити плейлисти;
– адмін панель для контролю веб-застосунку;
– створення коментарів.

Вимоги до даних

Інформаційна система повинна зберігати дані в об’єктно-реляційній базі даних та зберігання файлів у хмарному сховищі.

Вимоги до інтерфейсу

Перелік вимог до інтерфейсу наступний:

– вебсторінки повинні однаково відображатись незалежно від виду та версії браузера;

– інтерфейс повинен бути простим;

– сторінки повинні використовувати кодування UTF-8;

– інтерфейс повинен бути адаптивним (різне відображення вебсторінок в залежності від роздільної здатності екрана пристрою).

Вимоги до продуктивності

Система повинна підтримувати мінімум 50 користувачів, що працюють одночасно, а час відгуку не повинен перевищувати 0,5 секунд для простих і 1,5 секунди для складніших задач.

Вимоги до безпеки для інформаційної системи.

– доступ до системи має здійснюватися лише за допомогою веббраузерів для різних пристроїв;

– доступ до вебсайту повинен бути здійснений за допомогою протоколу HTTPS;

– повинен бути реалізована JWT авторизація та аунтетинфікація на основі refresh та access токенів;

– Система повинна відповідати вимогам GDPR(загальний регламент про захист даних). Користувачі повинні знати які їхні особисті дані зберігаються, як вони обробляються, а також вони мають мати можливість видаляти, редагувати та заперечувати обробку даних.

Експлуатаційні вимоги

Перелік експлуатаційних вимог такий:

– браузери: Chrome, Safari, Edge, Mozilla Firefox;

– операційні системи: Windows 10/11, Linux, Mac OS.

Політичні та юридичні вимоги

Платформа повинна мати ліцензії на використання музичних творів, щоб уникнути порушення авторських прав. Це може включати договори з правовласниками, музичними лейблами, артистами або організаціями, які колективно управляють правами.

Інші обмеження

Немає.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ

2.1 Опис архітектури та технологій системи

В результаті аналізу вимог було виділено наступних акторів:

– guest: даний актор представляє неавторизованого користувача. Він має обмежений набір можливостей в системі, такі як перегляд головної сторінки для неавторизованих користувачів, можливість реєстрації та авторизації, а також вибір мови інтерфейсу;

– administrator: Особа або група осіб, відповідальна за адміністрування та підтримку роботи системи. Адміністратор має можливість керування користувачами, вирішення технічних питань, а також моніторинг безпеки системи;

– user: даний актор репрезентує авторизованого користувача. Він має весь спектр можливостей для роботи з інформаційною системою.

Для вищезгаданих акторів наведено відповідні варіанти використання в таблиці 1.1.

Таблиця 1.1 – Варіанти використання системи

Код	Актор	Найменування	Формування
1	2	3	4
G1	Guest	Створити акаунт	Передбачає можливість створення облікового запису для неавторизованого користувача.
G2	Guest	Ввійти в акаунт	Неавторизований користувач може здійснити вхід в обліковий запис за допомогою логіна та пароля або з використанням google акаунта.

Продовження таблиці 1.1

G3	Guest	Прослуховування музики	Передбачає можливість взаємодіяти з музикою вибирати з списку, включати, ставити на паузу, перемотувати.
G4	Guest	Перегляд опису музики	Можливість переглянути детальний опис музики з коментарями.
G5	Guest	Перегляд найпопулярнішої музики	Передбачає можливість перегляду списку музики, які найбільш відтворенні.
G6	Guest	Перегляд альбомів авторів	Можливість переглянути альбоми.
G7	Guest	Перегляд пісень, альбомів, плей листів у авторів	Можливість переглянути детальний опис усіх альбомів та усієї музики авторів.
U1	User	Збереження плей листів	Передбачає збереження плей листів.
U2	User	Завантажити музику	Передбачає можливість завантаження музики з текстом, картинкою альбома, назвою, автором.
U4	User	Видалення музики	Авторизований користувач може видалити завантажену ним музику.
U5	User	Створення альбому	Авторизований користувач має можливість створити альбом з вибраної музики.

Продовження таблиці 1.1

U6	User	Створення плей листів	Передбачає можливість створення плей листів та добавлення до них музики.
U7	User	Збереження музики	Передбачає можливість збереження музики.
A1	Administrator	Технічна підтримка	Адміністратор може забезпечувати технічну підтримку системи, вирішуючи можливі проблеми та відповідаючи на запитання користувачів.

Діаграма варіантів використання, що показана на (рис. 2.1), є важливим інструментом для візуалізації взаємодії між різними учасниками системи та основних сценаріїв їхньої взаємодії. Діаграма є результатом застосування стандарту Unified Modeling Language (UML), який забезпечує уніфікацію та стандартизацію процесу моделювання програмних систем. На діаграмі виділено дві основні сутності: актори та прецеденти. Актори представляють ролі або сутності, які взаємодіють з системою, тоді як варіанти використання представляють конкретні взаємодії або сценарії, які можуть відбуватися між акторами та самою системою.

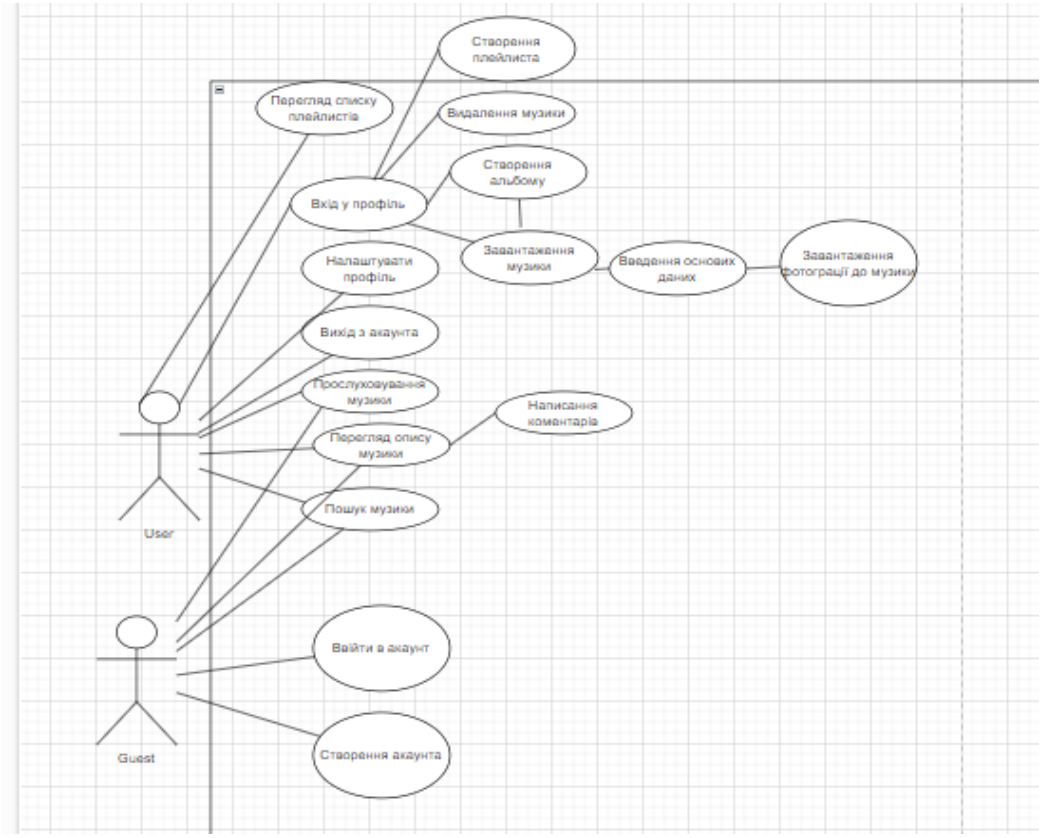

Рисунок 2.1 – Діаграма варіантів використання

2.2 Проєктування архітектури

Один з ключових етапів розробки програмного забезпечення є проєктування архітектури. Даний етап дає основоположення для застосунку який буде ефективним та легко масштабованим. Що в свою чергу пришвидшить розробку та покращить подальшу підтримку.

У даній роботі, яка має на меті розробку інтерактивного веб-застосунку з використанням Nest.js та Next.js. Даний застосунок користувачам надасть такі можливості, як прослуховування, завантаження, обміну музикою та взаємодії з музичним контентом. Користувачі матимуть змогу редагувати свій профіль, додавати аватар та змінювати особисту інформацію.

За допомогою клієнт-серверної архітектури можна гнучко розділити логіку клієнтом та сервером щоб зменшити навантаження клієнтську сторону. Дана модель працює по схемі яка зображена на (рис 2.2)

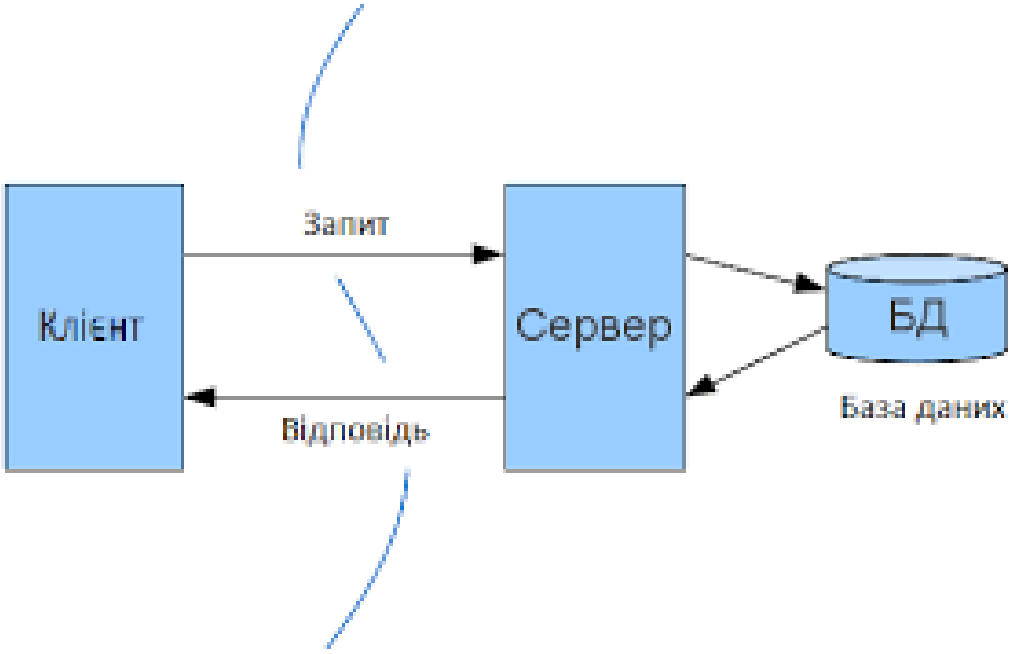

Рисунок 2.2 – Клієнт-серверна модель взаємодій

В даній моделі архітектури клієнт запитує віддаючи потрібні дані, а сервер надає по вказаних даних потрібні ресурси. При проєктуванні взаємодія між сервером та клієнтом відбувається згідно правил та принципів REST API. Одні з основних принципів, це клієнтська частина завжди прикріпляти тільки ті дані які потрібно тоді, як сервер має дати відповідь не зберігаючи інформацію про попередні запити. Що в свою чергу зробить взаємодію надійною та масштабованою. Також використання HTTP методів відповідно до їх чітко визначених функцій.

При проєктуванні для клієнтської та серверної частини веб-застосунку будуть побудовані на двох різних архітектурах таких, як MVC (Model-View-Controller) підхід(рис. 2.3) для розробки серверної частини та FSD (Feature-Sliced Design) методології для розробки клієнтської частини(рис. 2.4).

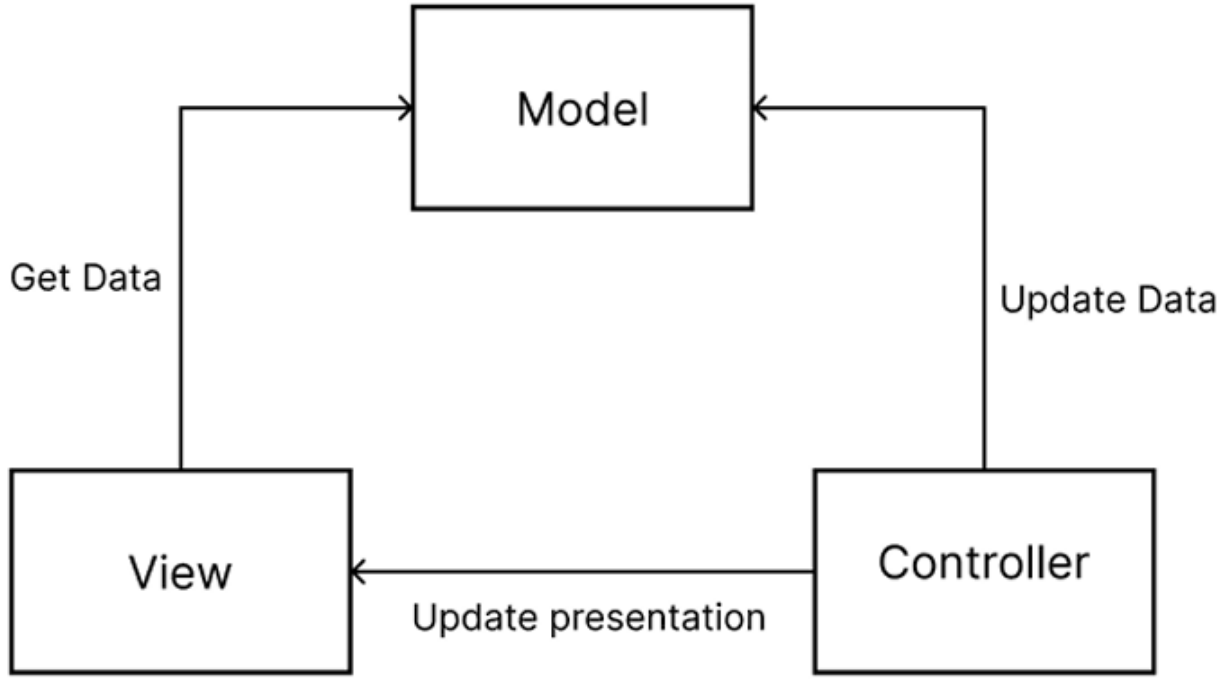

Рисунок 2.3 – Модель роботи MVC архітектури

Для розробки серверної частини за допомогою фреймворка Nest.js буде конструюватись на основі аріхітектури MVC.

– Model представляє структуру даних, формат і обмеження, з якими вони зберігаються. Він зберігає дані програми. По суті, це частина бази даних програми
– View– це те, що представлено користувачеві. Представлення використовують модель і представляють дані у формі, яку хоче користувач. Користувачеві також можна дозволити вносити зміни в дані, які йому надаються. Вони складаються зі статичних і динамічних сторінок, які відображаються або надсилаються користувачеві, коли користувач запитує їх.
– Controller: контролер контролює запити користувача, а потім генерує відповідну відповідь, яка передається глядачеві [9].

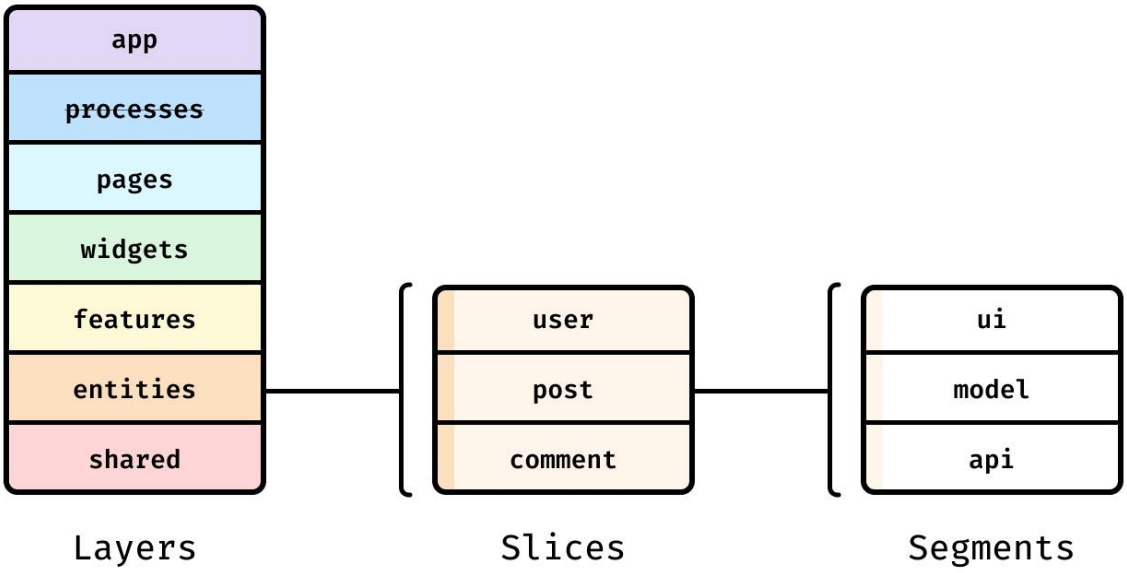

Рисунок 2.4 – Модель роботи MVC архітектури

Feature-Sliced Design (FSD) — це архітектурна методологія для створення зовнішніх додатків. Збірка правил і домовленостей щодо організації коду у React.js застосунках. Основна мета цієї методології це зробити проєкт більш зрозумілим і структурованим з огляду на постійні зміни бізнес-вимоги. Дана архітектура декларує створення та розподілення компонентів, що дозволяє гнучко перевикористовувати функції та елементи дизайну. Основна перевага FSD полягає в структурованій декомпозиції. У своїй найкращій формі він дає змогу майже детерміновано знайти будь-яку частину коду. Однак рівень декомпозиції є параметром і його можна налаштувати, щоб знайти баланс між простим впровадженням і кількістю переваг. Це створює стандартизовану архітектуру, яка легка для розуміння та підтримування [8].

На (рис 2.5) відображенна діаграма класів, яка використовується для опису структури системи на основі класів та описує їхні методи та атрибути, а також взаємозв’язки між класами. Що в свою чергу допомагає у розробці, розумінні та документуванні архітектури програмного забезпечення.

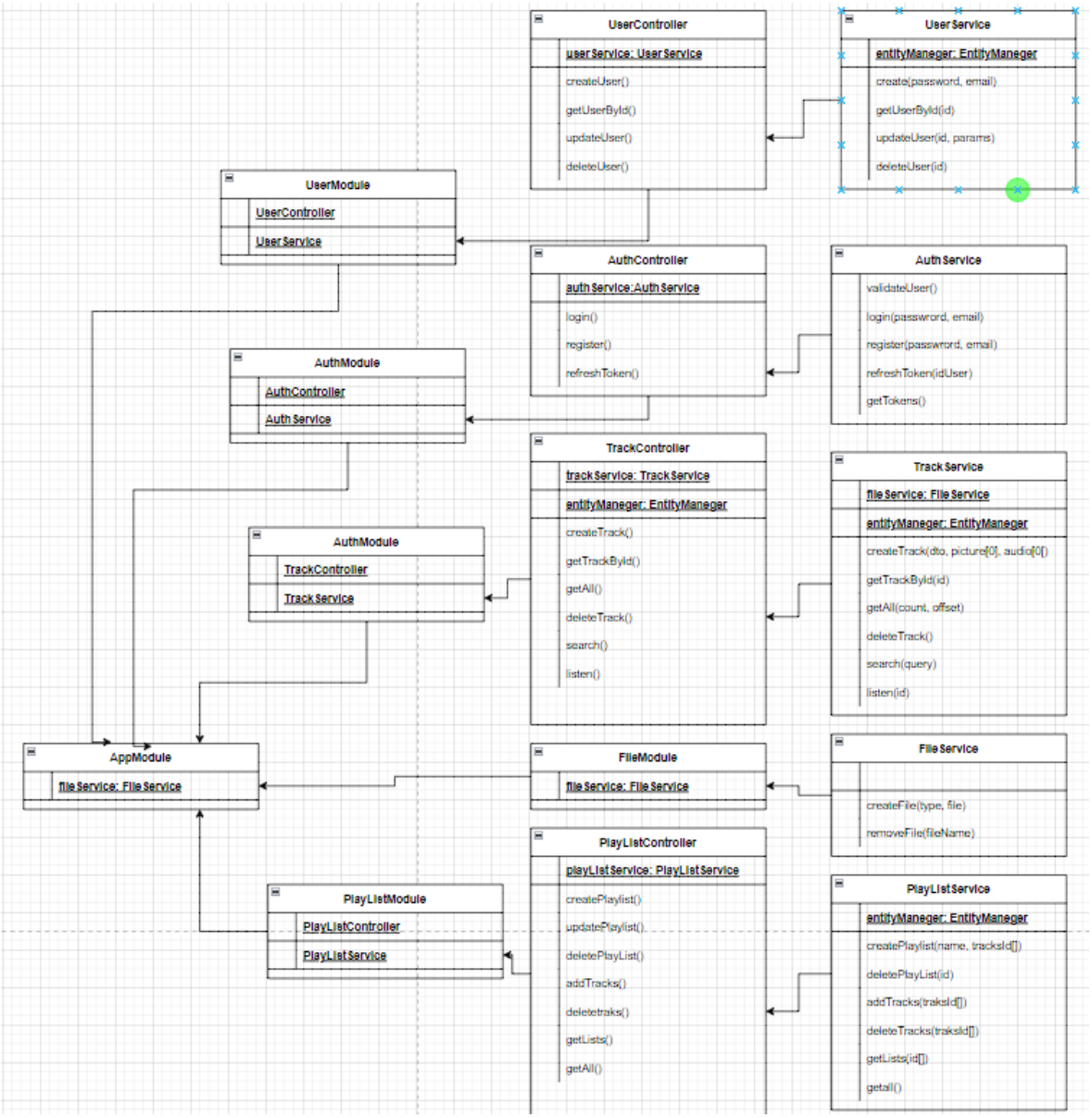

Рисунок 2.5 – Основна Діаграма Класів

Діаграма класів на (рис 2.5) представляє архітектуру музичної платформи, розробленої за допомогою Nest.js. Вона відображає модулі, контролери та сервіси, а також взаємозв'язки між ними.

Опис кожного класу та його методів

Модуль AppModule – є головний модулем, що імпортує інші модулі такі, як (UserModule, AuthModule, TrackModule, FileModule, PlayListModule).

– Модуль UserModule імпортує UserController і UserService.
– Модуль AuthModule імпортує AuthController і AuthService.

- Модуль TrackModule імпортує TrackController і TrackService.
- Модуль FileModule імпортує FileService.
- Модуль PlayListModule імпортує PlayListController і PlayListService.

Контролер – UserController використовує UserService для виконання CRUD операцій над користувачами.

Методи:

- createUser(): Створює нового користувача.
- getUserById(): Отримує користувача за ID.
- updateUser(): Оновлює дані користувача.
- deleteUser(): Видаляє користувача.

AuthController використовує AuthService для аутентифікації та авторизації користувачів.

Методи:

- login(): Авторизація користувача.
- register(): Реєстрація нового користувача.
- refreshToken(): Оновлення токена доступу.

TrackController використовує TrackService для управління треками.

Методи:

- createTrack(): Створює новий трек.
- getTrackById(): Отримує трек за ID.
- getAll(): Отримує всі треки.
- deleteTrack(): Видаляє трек.
- search(): Шукає треки за запитом.
- listen(): Відтворює трек.

PlayListController використовує PlayListService для управління плейлистами.

Методи:

- createPlaylist(): Створює новий плейлист.
- getPlayListById(): Отримує плейлист за ID.
- updatePlayList(): Оновлює дані плейлиста.
- deletePlayList(): Видаляє плейлист.
- addTrack(): Додає трек до плейлиста.
- deleteTrack(): Видаляє трек з плейлиста.
- getAll(): Отримує всі плейлисти.

UserService використовує EntityManager для взаємодії з базою даних.

Методи:

- create(password, email): Створює нового користувача.
- getUserById(id): Отримує користувача за ID.
- updateUser(id, params): Оновлює дані користувача.
- deleteUser(id): Видаляє користувача.

AuthService використовує EntityManager для взаємодії з базою даних і UserService для валідації користувачів.

Методи:

- validateUser(): Перевіряє автентичність користувача.
- login(email, password): Авторизація користувача.
- register(password, email): Реєстрація нового користувача.
- getTokens(): Отримує токени доступу для користувача.

TrackService використовує EntityManager для взаємодії з базою даних і FileService для роботи з файлами такими, як аудіо та фото.

Методи:

- createTrack(name, artistId, albumId, pictureId, audioId): Створює новий трек.
- getTrackById(id): Отримує трек за ID.
- getAll(count, offset): Отримує всі треки з пагінацією.
- deleteTrack(id): Видаляє трек.

- search(query): Шукає треки за запитом.
- listen(id): Відтворює трек.

FileService використовує EntityManager для взаємодії з базою даних.

Методи:

- createFile(type, file): Створює новий файл.
- removeFile(fileName): Видаляє файл.

PlayListService використовує EntityManager для взаємодії з базою даних.

Методи:

- createPlayList(name, trackIds): Створює новий плейлист.
- getPlayListById(id): Отримує плейлист за ID.
- updatePlayList(id, name, trackIds): Оновлює дані плейлиста.
- deletePlayList(id): Видаляє плейлист.
- addTrack(playListId, trackId): Додає трек до плейлиста.
- deleteTrack(playListId, trackId): Видаляє трек з плейлиста.
- getAll(): Отримує всі плейлисти.

Зв'язки між класами

- Контролери (UserController, AuthController, TrackController, PlayListController) використовують сервіси (UserService, AuthService, TrackService, PlayListService) для виконання бізнес-логіки і взаємодії з базою даних.
- Сервіси (TrackService, FileService, UserService, AuthService, PlayListService) використовують EntityManager для взаємодії з базою даних і виконання CRUD операцій.
- TrackService додатково використовує FileService для обробки файлів треків.

2.3 Проєктування БД та сховища файлів

Основними принципами роботи вибраної нами хмарної бази даних Firestore та PostgreSql є:

– таблиці: дані у PostgreSql організовані у таблиці. Колекція – це група документів, а документ – це JSON-подібний об'єкт, що містить дані. Кожен таблиця може мати унікальний ідентифікатор, який ідентифікує його в таблиці;

– реальний час: однією з ключових особливостей Firestore та PostgreSql є можливість синхронізації даних у реальному часі між усіма пристроями, що взаємодіють з базою даних. Якщо дані в базі даних змінюються, всі підключені пристрої негайно отримують оновлення;

– доступ до даних: PostgreSql дозволяє отримувати доступ до даних за допомогою різноманітних запитів; можна отримувати, оновлювати, додавати та видаляти дані з таблиць та колонок [7].

Опис Полів Таблиць Бази Даних у PostgreSQL з типами даних

Таблиця: PlayLists, призначена для зберігання інформації про плейлисти, створені користувачами або системою.

id: SERIAL PRIMARY KEY - унікальний ідентифікатор плейліста.

name: VARCHAR(255) - назва плейліста.

tracks_id: INTEGER[] - масив ідентифікаторів треків, що входять до плейліста.

rated: FLOAT - оцінка плейліста.

picture: TEXT - шлях до зображення плейліста.

Таблиця: Tracks, призначена для зберігання інформації про музичні треки.

– id: SERIAL PRIMARY KEY - унікальний ідентифікатор треку.
– name: VARCHAR(255) - назва треку.
– artist_id: INTEGER - ідентифікатор артиста, який створив трек.
– text: TEXT - текст пісні або лірика.
– rated: FLOAT - оцінка треку.

- picture: TEXT - шлях до зображення треку.
- audio: TEXT - шлях до аудіофайлу треку.
- albums_id: `INTEGER` - ідентифікатор альбому, до якого належить трек.

Таблиця: Albums, призначена для зберігання інформації про музичні альбоми.

- id: SERIAL PRIMARY KEY - унікальний ідентифікатор альбому.
- name: VARCHAR(255) - назва альбому.
- author: VARCHAR(255) - автор або виконавець альбому.
- picture: TEXT - шлях до зображення альбому.
- rated: FLOAT - оцінка альбому.
- artist_id: INTEGER - ідентифікатор артиста, який створив альбом.

Таблиця: Artist, призначена для зберігання інформації про артистів.

- id: SERIAL PRIMARY KEY - унікальний ідентифікатор артиста.
- nickname: VARCHAR(255) - псевдонім артиста.
- rated: FLOAT - оцінка артиста.
- listens: INTEGER - кількість прослуховувань треків артиста.
- picture: TEXT - шлях до зображення артиста.
- account_id: INTEGER - ідентифікатор облікового запису, пов'язаного з артистом.

Таблиця: Users, призначена для зберігання інформації про користувачів платформи.

- id: SERIAL PRIMARY KEY - унікальний ідентифікатор користувача.
- email: VARCHAR(255) UNIQUE NOT NULL - електронна адреса користувача.
- password: VARCHAR(255) NOT NULL - зашифрований пароль користувача.
- isActivated: BOOLEAN - статус активації облікового запису.
- activationLink: TEXT - посилання для активації облікового запису.
- owner_id: INTEGER - ідентифікатор власника, посилання на таблицю Account.

Таблиця: `Account, призначена для зберігання інформації про облікові записи користувачів.

- id: SERIAL PRIMARY KEY - унікальний ідентифікатор облікового запису.
- savedPlayList_id: `INTEGER[]` - масив ідентифікаторів збережених плейлістів.
- user_id: INTEGER - ідентифікатор користувача, до якого належить обліковий запис.
- savedTracks_id: INTEGER[] - масив ідентифікаторів збережених треків.
- savedAlbums_id: INTEGER[] - масив ідентифікаторів збережених альбомів.

Таблиця: PersonalInfo, призначена для зберігання особистої інформації користувачів.

- id: SERIAL PRIMARY KEY - унікальний ідентифікатор особистої інформації.
- secondName: VARCHAR(255) - друге ім'я або прізвище користувача.
- lastName: VARCHAR(255) - прізвище користувача.
- fullName: VARCHAR(255) - повне ім'я користувача.
- firstName: VARCHAR(255) - ім'я користувача.
- phoneNumber:`VARCHAR(20) - номер телефону користувача.
- avatar: TEXT - URL або шлях до аватара користувача.

Таблиця: JWTs, призначена для зберігання JSON Web Token для автентифікації користувачів.

- id: SERIAL PRIMARY KEY - унікальний ідентифікатор JSON Web Token.
- user_id: INTEGER - Ідентифікатор користувача, до якого належить токен.
- refreshToken: TEXT - Токен для оновлення доступу (refresh token).

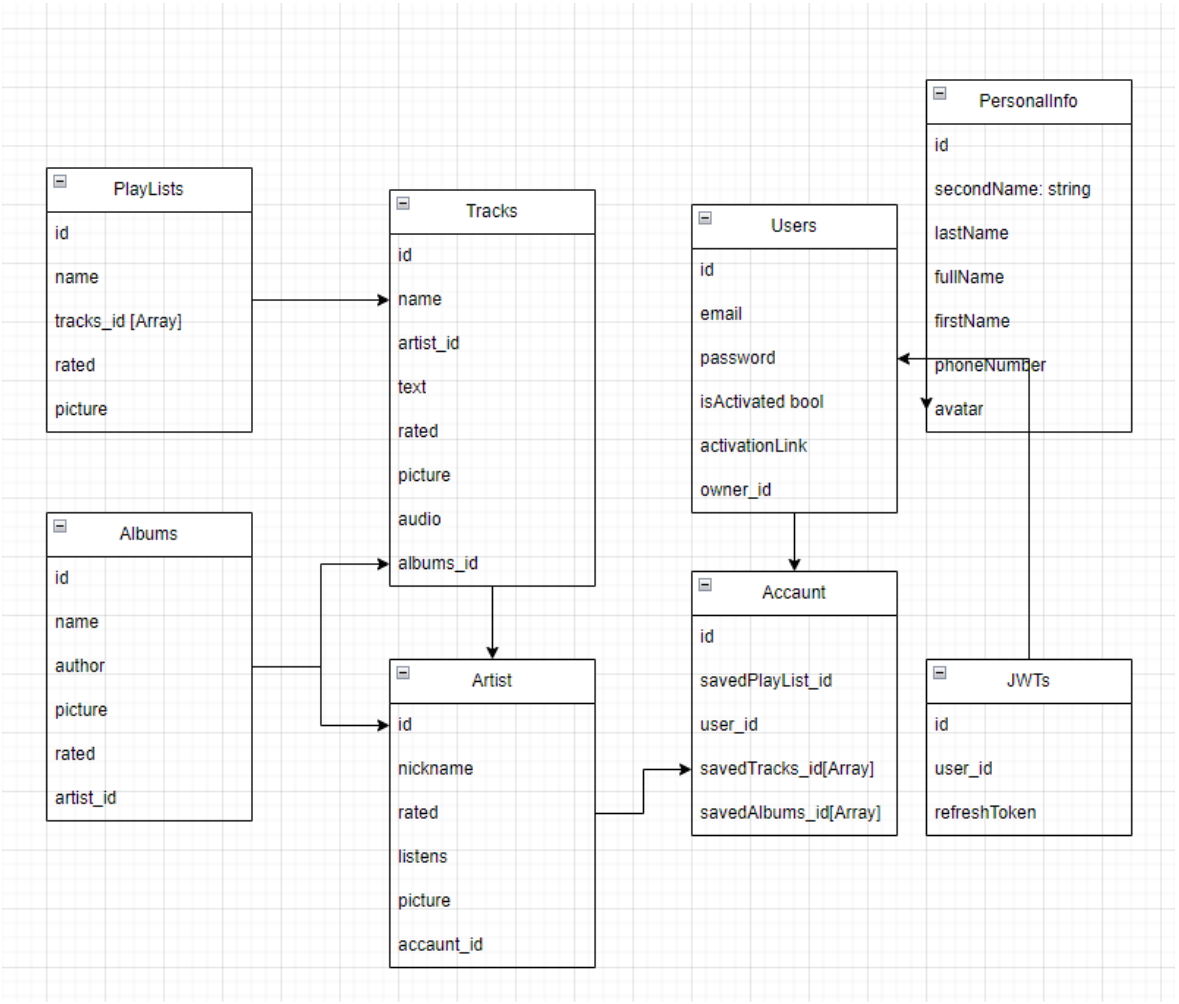

Рисунок 2.6 – схема базиданих у PosgerSQL

Ця структура бази даних забезпечить організоване зберігання інформації про плейлисти, треки, альбоми, артистів, користувачів та їх облікові записи, персональну інформацію та токени автентифікації. Кожна колонка має чітко визначений тип даних, що дозволяє ефективно зберігати та обробляти інформацію. За допомогою зв’язків можна гнучко шукати та маніпулювати даними у таблицях.

2.4 Розробка серверної частини веб-застосунку

Розробка ендпоїнтів для REST API є ключовим етапом у створенні сучасних веб-застосунків. REST (Representational State Transfer) визначає архітектурний стиль для взаємодії між компонентами системи. Ендпоїнти представляють собою

URI-ресурси, до яких можна здійснювати HTTP-запити (GET, POST, PUT, DELETE) для виконання різних операцій.

Лістинг 2.1 – файл track.controller.ts

```
import {Body, Controller, Delete, Get, Param, Post, Query, UploadedFiles, UseInterceptors} from "@nestjs/common";
import {TrackService} from "./track.service";
import {CreateTrackDto} from "./dto/create-track.dto";
import {CreateCommentDto} from "./dto/create-comment.dto";
import {FileFieldsInterceptor} from "@nestjs/platform-express";


@Controller('/tracks')
export class TrackController {
    constructor(private trackService: TrackService) {    }

    @Post()
    @UseInterceptors(FileFieldsInterceptor([
        { name: 'picture', maxCount: 1 },
        { name: 'audio', maxCount: 1 },
    ]))
    create(@UploadedFiles() files, @Body() dto: CreateTrackDto) {
        const {picture, audio} = files
        return this.trackService.create(dto, picture[0], audio[0]);
    }

    @Get()
    getAll(@Query('count') count: number,
           @Query('offset') offset: number) {
        return this.trackService.getAll(count, offset)
    }

    @Get('/search')
    search(@Query('query') query: string) {
        return this.trackService.search(query)
    }
```

```
    @Get(':id')
    getOne(@Param('id') id: ObjectId) {
        return this.trackService.getOne(id);
    }

    @Delete(':id')
    delete(@Param('id') id: ObjectId) {
        return this.trackService.delete(id);
    }

    @Post('/comment')
    addComment(@Body() dto: CreateCommentDto) {
        return this.trackService.addComment(dto);
    }

    @Post('/listen/:id')
    listen(@Param('id') id: ObjectId) {
        return this.trackService.listen(id);
    }
}
```

У лістингу 2.1 у коді визначено контролер TrackController для обробки HTTP-запитів, пов'язаних із музичними треками. Реалізовані наступні функції:

- @Controller('/tracks') – що цей контролер обробляє запити, які починаються з шляху '/tracks'.
- constructor(private trackService: TrackService) { } - Конструктор контролера, який приймає сервіс `TrackService` як залежність.
- @Post() create() – обробляє POST-запит для створення нового музичного треку. Використовує `FileFieldsInterceptor`, щоб обробити файли (зображення та аудіо), які завантажуються разом із запитом.
- @Get() getAll() – обробляє GET-запит для отримання списку усіх музичних треків. Приймає параметри `count` (кількість елементів) та `offset` (зсув).

- @Get('/search') search() – обробляє GET-запит для пошуку музичних треків за заданим запитом.
- @Get(':id') getOne() – обробляє GET-запит для отримання інформації про конкретний музичний трек за його ідентифікатором.
- @Delete(':id') delete() – обробляє DELETE-запит для видалення музичного треку за його ідентифікатором.
- @Post('/comment') addComment() – обробляє POST-запит для додавання коментаря до музичного треку.
- @Post('/listen/:id') listen() – обробляє POST-запит для відзначення того, що користувач прослуховує конкретний музичний трек.

Кожен метод використовує сервіс TrackService, який має відповідні методи для обробки логіки, пов'язаної з музичними треками, від створення до видалення та інших дій. Використання декораторів дозволяє легко налаштовувати маршрути та обробляти дані від клієнта відповідно до визначеної логіки.

Лістинг 2.2 – файл track.service.ts

```
import {Injectable} from "@nestjs/common";
import {Track, TrackDocument} from "./schemas/track.schema";
import {Comment, CommentDocument} from "./schemas/comment.schema";
import {CreateTrackDto} from "./dto/create-track.dto";
import {CreateCommentDto} from "./dto/create-comment.dto";
import {FileService, FileType} from "../file/file.service";


@Injectable()
export class TrackService {

    constructor(@InjectModel(Track.name)        private        trackModel:
Model<TrackDocument>,
                @InjectModel(Comment.name)      private      commentModel:
Model<CommentDocument>,
                private fileService: FileService) {}

    async create(dto: CreateTrackDto, picture, audio): Promise<Track> {
```

```
        const audioPath = this.fileService.createFile(FileType.AUDIO, audio);
        const picturePath = this.fileService.createFile(FileType.IMAGE, picture);
        const track = await this.trackModel.create({...dto, listens: 0, audio: audioPath, picture: picturePath})
        return track;
    }

    async getAll(count = 10, offset = 0): Promise<Track[]> {
        const tracks = await this.trackModel.find().skip(Number(offset)).limit(Number(count));
        return tracks;
    }

    async getOne(id: ObjectId): Promise<Track> {
        const track = await this.trackModel.findById(id).populate('comments');
        return track;
    }

    async delete(id: ObjectId): Promise<ObjectId> {
        const track = await this.trackModel.findByIdAndDelete(id);
        return track._id
    }

    async addComment(dto: CreateCommentDto): Promise<Comment> {
        const track = await this.trackModel.findById(dto.trackId);
        const comment = await this.commentModel.create({...dto})
        track.comments.push(comment._id)
        await track.save();
        return comment;
    }

    async listen(id: ObjectId) {
        const track = await this.trackModel.findById(id);
        track.listens += 1
        track.save()
    }
```

```
    async search(query: string): Promise<Track[]> {
        const tracks = await this.trackModel.find({
            name: {$regex: new RegExp(query, 'i')}
        })
        return tracks;
    }
}
```

У лістингу 2.2 у цьому коді описано сервіс TrackService, який використовується для взаємодії з базою даних та обробки логіки, пов'язаної з музичними треками:

- @Injectable() декоратор, який позначає клас TrackService як сервіс, який може бути вбудований у інші компоненти NestJS.
- async create(dto: CreateTrackDto, picture, audio): Promise<Track> – метод для створення нового музичного треку. Викликається метод `this.fileService.createFile`, щоб зберегти зображення та аудіофайл, та потім створюється новий запис у колекції Track з вказаними параметрами.
- async getAll(count = 10, offset = 0): Promise<Track[]> – метод для отримання списку музичних треків з можливістю вказати кількість та зсув. Викликається метод find для отримання відповідних записів.
- async getOne(id): Promise<Track> – метод для отримання інформації про конкретний музичний трек за його ідентифікатором. Використовується метод.
- async delete(id): Promise<ObjectId> – метод для видалення музичного треку за його ідентифікатором. Викликається метод findByIdAndDelete для видалення запису.
- async listen(id) – метод для відзначення того, що трек був прослуханий. Знаходить трек за його ідентифікатором та збільшує значення поля listens на одиницю.
- async search(query: string): Promise<Track[]> – метод для пошуку музичних треків за заданим запитом. Використовує регулярний вираз та метод find для отримання відповідних записів.

Цей сервіс забезпечує логіку створення, отримання, видалення та взаємодії з музичними треками у системі. Він також використовує сервіс FileService для обробки файлів (зображень та аудіо).

2.5 Розробка клієнтської частини веб-застосунка

React.js-компоненти є основними будівельними блоками у світі Next.js, фреймворка для розробки веб-додатків. Кожен Next.js-компонент - це самодостатній модуль інтерфейсу, який використовується для конструювання високопродуктивних та ефективних веб-додатків.

Роль Next.js-компонентів полягає в тому, щоб структурувати додаток, роблячи його більш зрозумілим та легко управляємі. Кожен компонент може містити власний стан, а також логіку, пов'язану зі своєю областю відповідальності. Це дозволяє створювати масштабовані та перевикористовувані елементи інтерфейсу [5].

Next.js підтримує як класові, так і функціональні компоненти. Класові компоненти можуть мати внутрішній стан та використовувати методи життєвого циклу, в той час як функціональні компоненти базуються на функціях та можуть використовувати хуки для управління станом та додатковою логікою[5].

Реактивна програма Next.js-компонентів забезпечує автоматичне оновлення інтерфейсу при змінах, що дозволяє легко створювати динамічні та інтерактивні веб-додатки.

Дана концепція компонентів допомагає зробити розробку веб-додатків на Next.js більш модульною, розширюваною та підтримуваною, що є важливим для успішного створення високоякісних веб-застосунків на основі Next.js.

Реалізовані наступні компоненти:

– RegisterPage: компонент, що репрезентує сторінку для реєстрації;

- HomePage: головна сторінка для авторизованих користувачів, що відображає список музики;
- LoginPage: сторінка для здійснення авторизації;
- TracksPage: сторінка для перегляду списку музики, що містить віджет для налаштування параметрів пошуку;
- ProfilePage: компонент, що представляє сторінку профілю користувача. На цій сторінці авторизований користувач може завантажити зображення та вибрати поточне зображення профілю, створити плей лист, завантажити музику, створити альбом, видалити завантажену музику;
- TrackPage: сторінка для відображення детальної інформації про музику;
- WelcomePage: головна сторінка для неавторизованих користувачів, що містить інформацію про інформаційну систему.

Також створено набір компонентів, що використовуються в інформаційній системі повторно, а саме:

- FileUpload: компонент для завантаження файлів;
- Navbar: компонент, головного навігаційного меню з усіма головними сторінками;
- Player: компоненти який створює плеєр для прослуховування музики;
- StepWrapper: компонент меню;
- TrackTime: компонент, що відображає час музики;
- TrackList: компонент, що містить список всієї музики або список музики у аьбомі або у плейисті;
- TrackProgress: компонент для відображення прогресу;
- MainLayout: компонент обгортка для закріплення Navbar;
- Login: компонент для здійснення авторизації. Використовується компонентом LoginPage;
- Register: компонент для здійснення реєстрації. Використовується компонентом RegisterPage;

Використання бібліотеки Material UI у поєднанні з Next.js є потужним засобом для прискорення розробки веб-додатків. Material UI надає набір готових до

використання UI компонентів, які відповідають дизайну згідно із концепцією матеріального дизайну від Google. Це важливо для створення сучасного та консистентного інтерфейсу [5].

Завдяки Material UI можна уникнути написання власних стилів та компонентів для багатьох базових елементів інтерфейсу. Замість цього можна використовувати готові компоненти, які вже мають вбудовані стилі та функціонал. Це приводить до значного зменшення часу, витраченого на розробку, та сприяє швидшій імплементації проекту.

Material UI також надає можливість кастомізації компонентів за допомогою тем та стилів, що дозволяє забезпечити відповідність дизайну вашого проєкту унікальним вимогам. Це робить інтеграцію Material UI у Next.js більш гнучкою та адаптованою до конкретних потреб команди розробників.

У підсумку, використання Material UI з Next.js прискорює розробку, полегшує підтримку коду і дозволяє зосередитися на бізнес-логіці та функціоналі додатку, замість повторного винахідництва колеса у сфері дизайну та інтерфейсу.

Створення та конфігурація Redux store

Лістинг 2.3

```
import { Context, createWrapper, MakeStore } from "next-redux-wrapper";
import { AnyAction, applyMiddleware, createStore } from "redux";
import { reducer, RootState } from "./reducers";
import thunk, { ThunkAction, ThunkDispatch } from "redux-thunk";

const makeStore: MakeStore<RootState> = (context: Context) =>
  createStore(reducer, applyMiddleware(thunk));

export const wrapper = createWrapper<RootState>(makeStore, { debug: true });

export type NextThunkDispatch = ThunkDispatch<RootState, void, AnyAction>;
```

Висновок: makeStore створює Redux store з використанням редюсера та middleware (thunk). wrapper визначає обгортку для Next.js, щоб легко налаштувати та ініціалізувати Redux store.

Лістинг 2.4

```
import { combineReducers } from "redux";
import { playerReducer } from "./playerReducer";
import { HYDRATE } from "next-redux-wrapper";
import { trackReducer } from "./trackReducer";

const rootReducer = combineReducers({
  player: playerReducer,
  track: trackReducer,
});

export const reducer = (state, action) => {
  if (action.type === HYDRATE) {
    const nextState = {
      ...state,
      ...action.payload,
    };
    return nextState;
  } else {
    return rootReducer(state, action);
  }
};

export type RootState = ReturnType<typeof rootReducer>;
```

Висновок: rootReducer об'єднує різні редюсери (playerReducer та trackReducer) за допомогою combineReducers. При використанні HYDRATE, створюється новий стан на основі попереднього стану та даних гідрації.

Лістинг 2.5

```
import { PlayerAction, PlayerActionTypes, PlayerState } from "../../types/player";

const initialState: PlayerState = {
  currentTime: 0,
  duration: 0,
  active: null,
  volume: 50,
  pause: true,
```

```
};

export const playerReducer = (state = initialState, action:
PlayerAction): PlayerState => {
  switch (action.type) {
  }
};
```

Висновок: playerReducer обробляє різні дії для стану гравця, такі як PAUSE, PLAY, SET_CURRENT_TIME.

Лістинг 2.6

```
import { TrackAction, TrackActionTypes, TrackState } from
"../../types/track";

const initialState: TrackState = {
  tracks: [],
  error: "",
};

export const trackReducer = (state = initialState, action:
TrackAction): TrackState => {
  switch (action.type) {
  }
};
```

Висновок: trackReducer обробляє різні дії для стану треку, такі як FETCH_TRACKS_ERROR, FETCH_TRACKS і т. д.

Загалом, Redux-код розділений на кілька частин, кожна з яких відповідає за конкретний аспект стану застосунку (гравець, трек і т. д.). Redux допомагає управляти станом і діями в додатку, забезпечуючи більшу структурованість і контроль.

Вигляд головної сторінки інформаційної системи з пошуку фільмів за трейлерами для авторизованих користувачів можна побачити на (рис. 2.7). Візуальну розмітку компонентів, що використовуються компонентом HomePage, показано на (рис. 2.8).

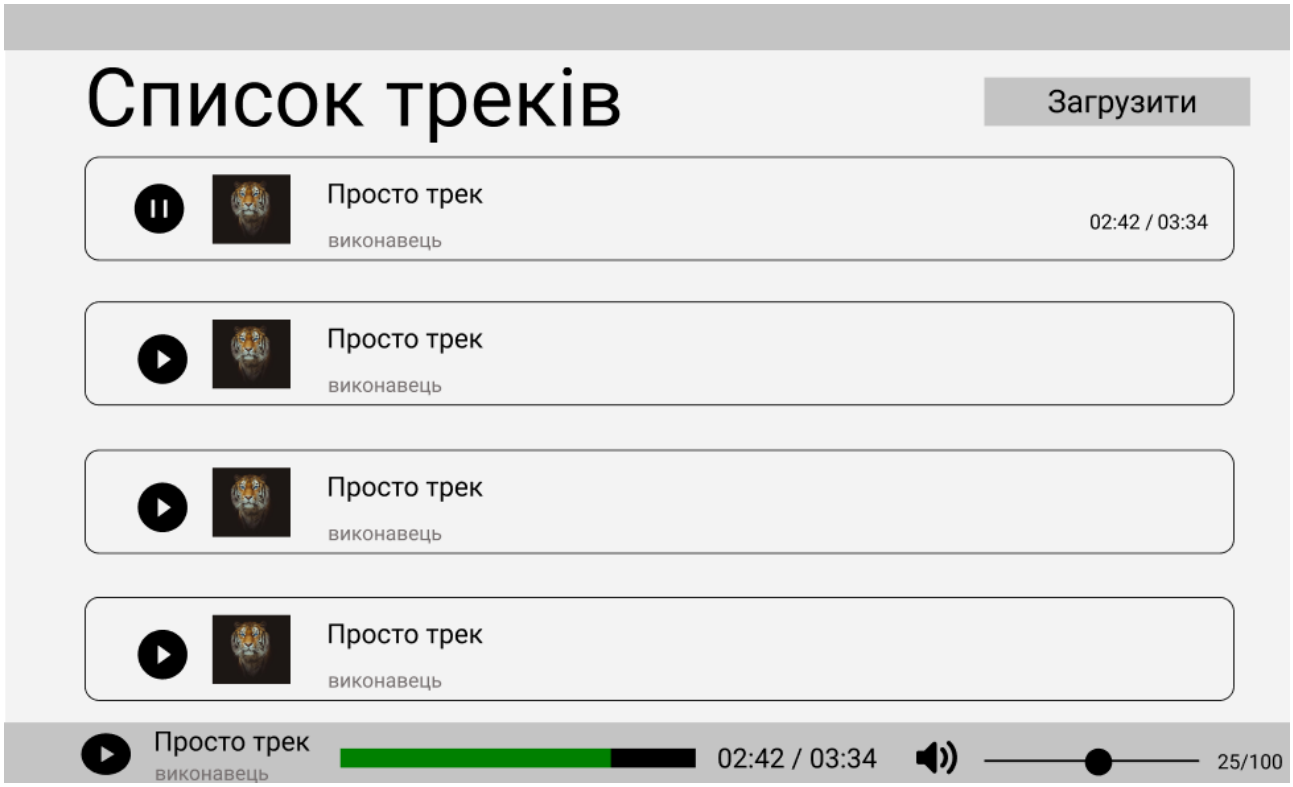

Рисунок 2.7 – Головна сторінка для користувачів

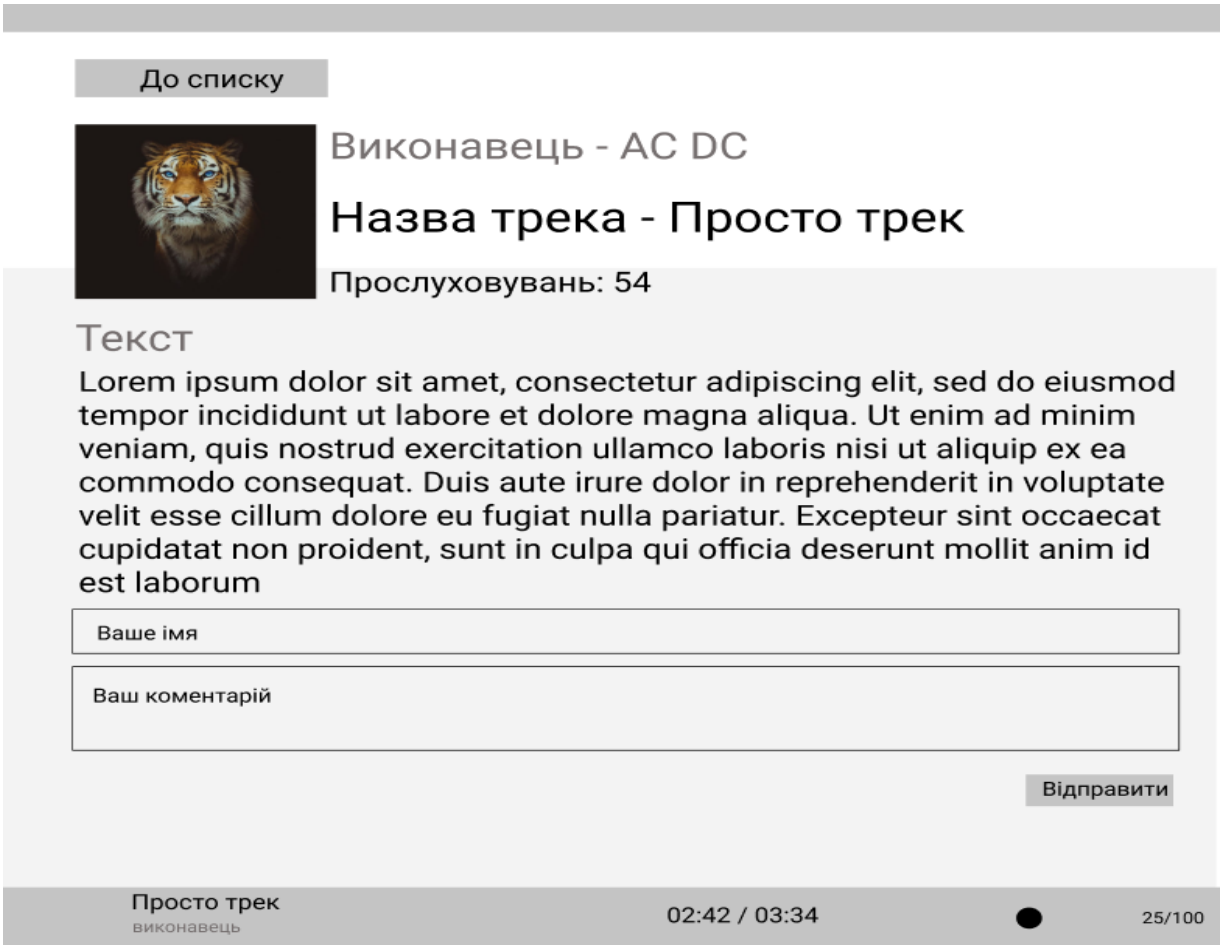

Рисунок 2.8 – Структура компонента Головне меню

На (Рис. 2.8) відображено інформацію про музику з списком коментарів, кількістю прослуховувань. Також на цій сторінці є можливість створення коментаря до музики.

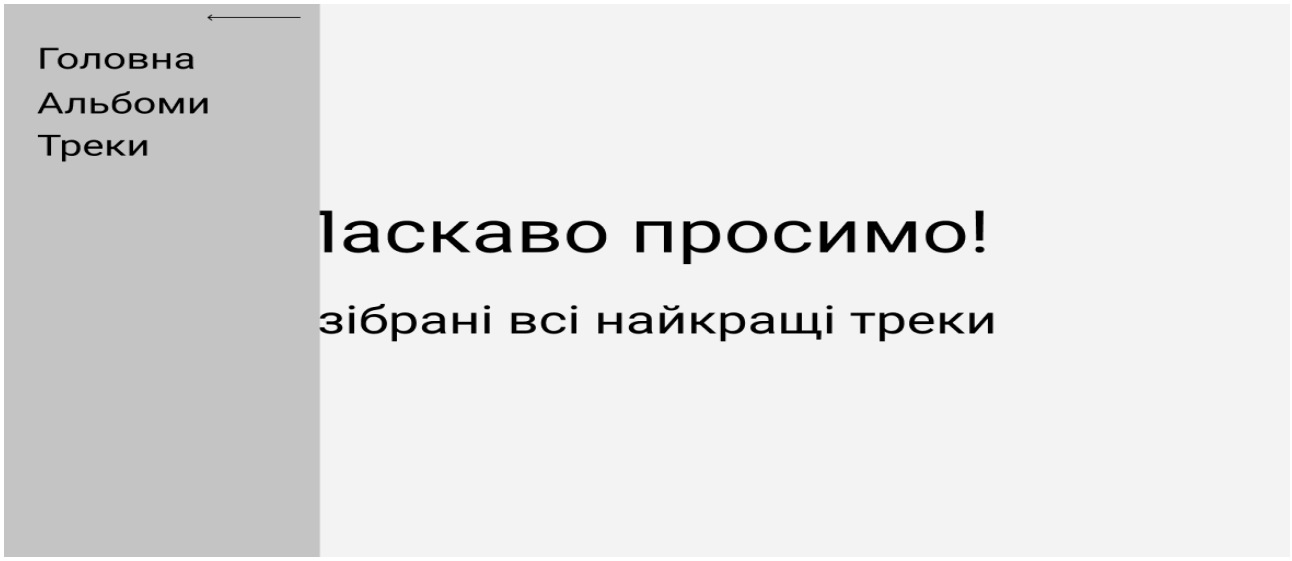

Рисунок 2.9 – Структура компонента Головне меню

Вхідна сторінка на (рис. 2.9) при заходженні на веб-застосунок після якої можна перейти на інші сторінки по посиланнях.

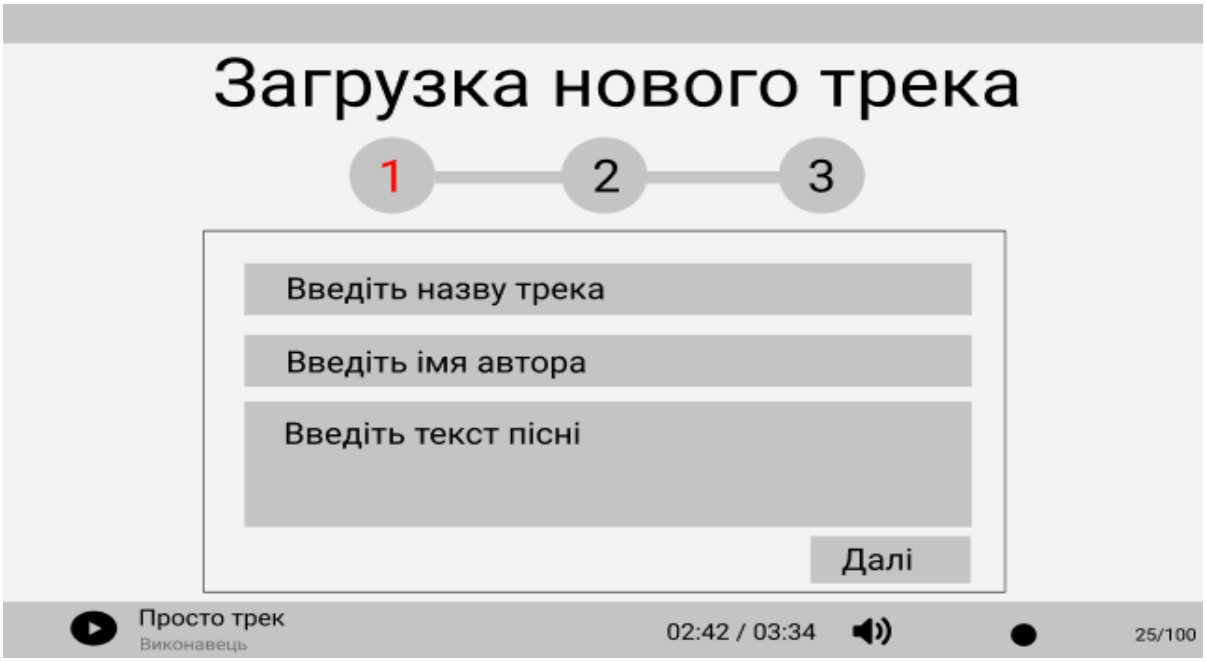

Рисунок 2.10 – Сторінка для завантаження музики

На (рис. 2.10) зображена сторінка для завантаження музики. Форма для заповнення основної інформації про музику.

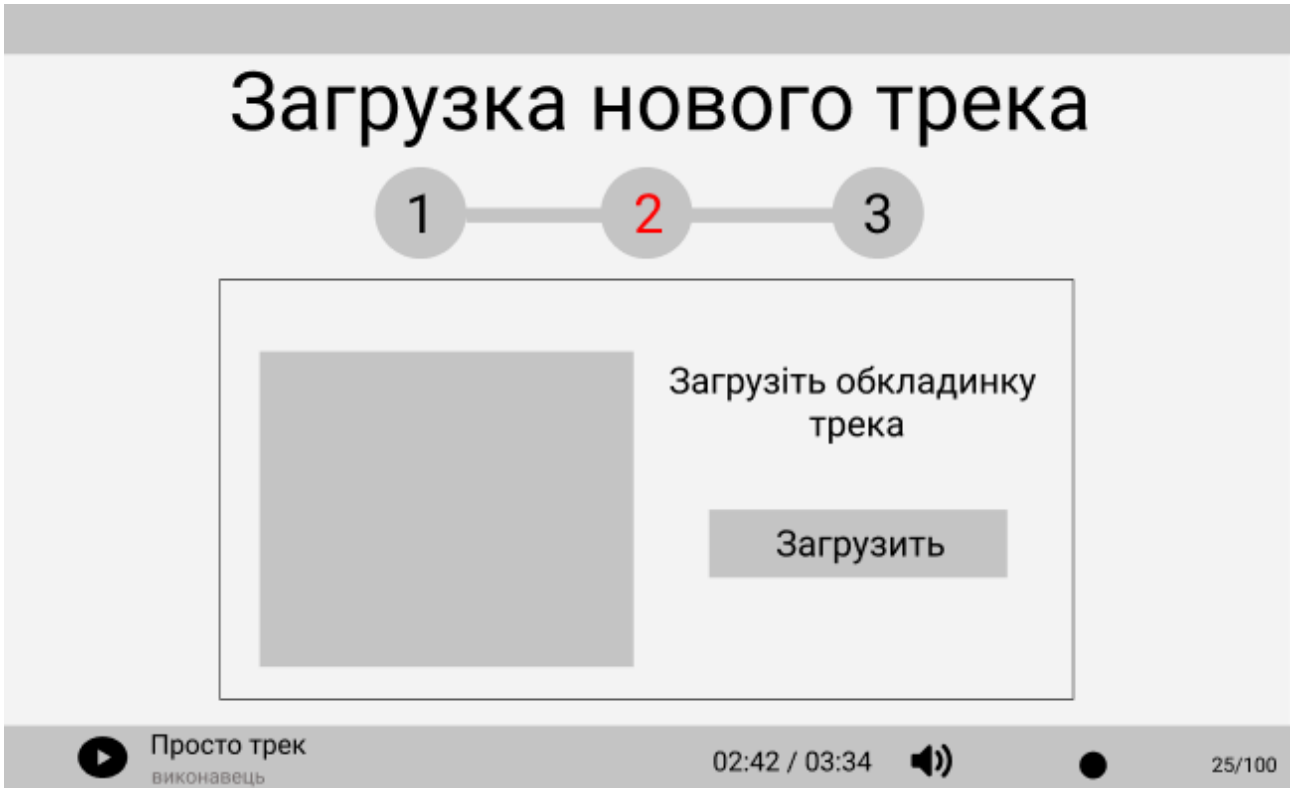

Рисунок 2.11 – Етап форми для завантаження музики

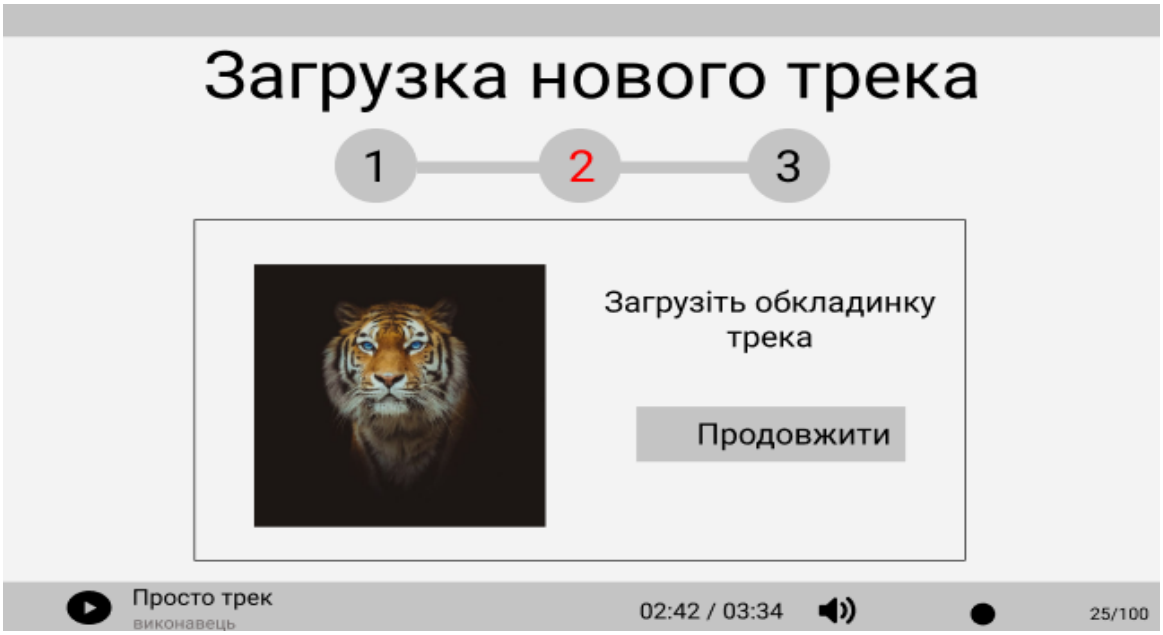

Рисунок 2.12 – Етап форми для завантаження музики

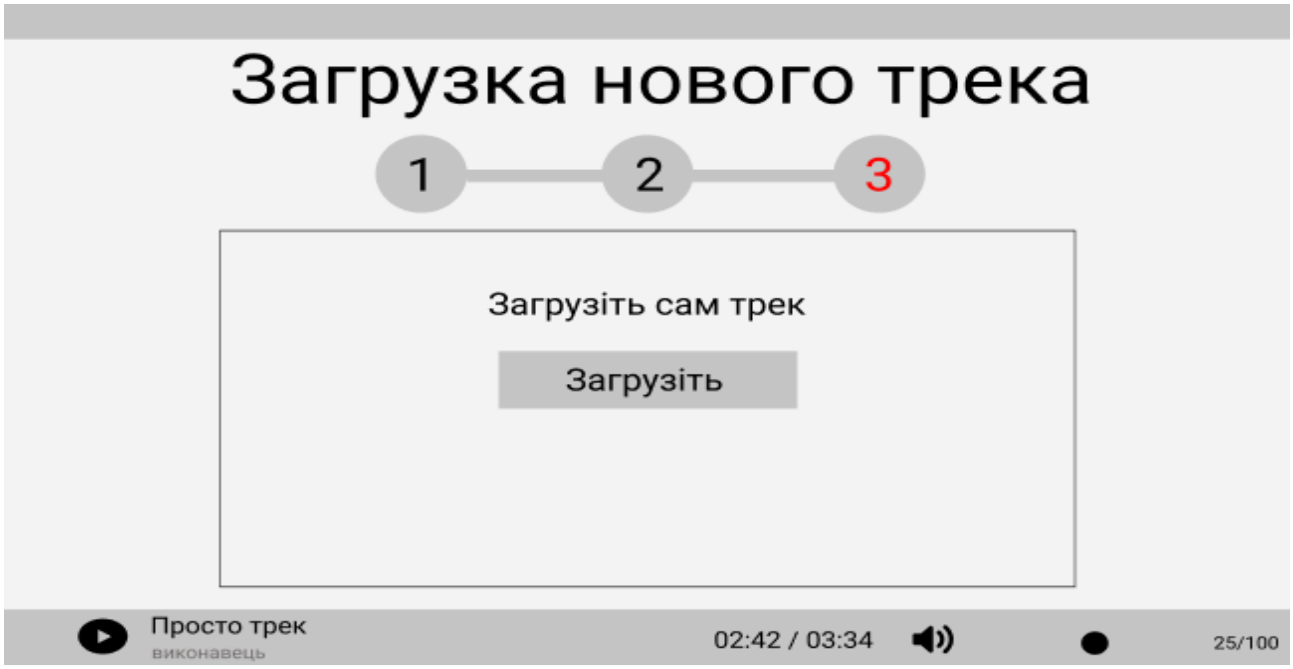

Рисунок 2.13 – Етап форми для завантаження музики

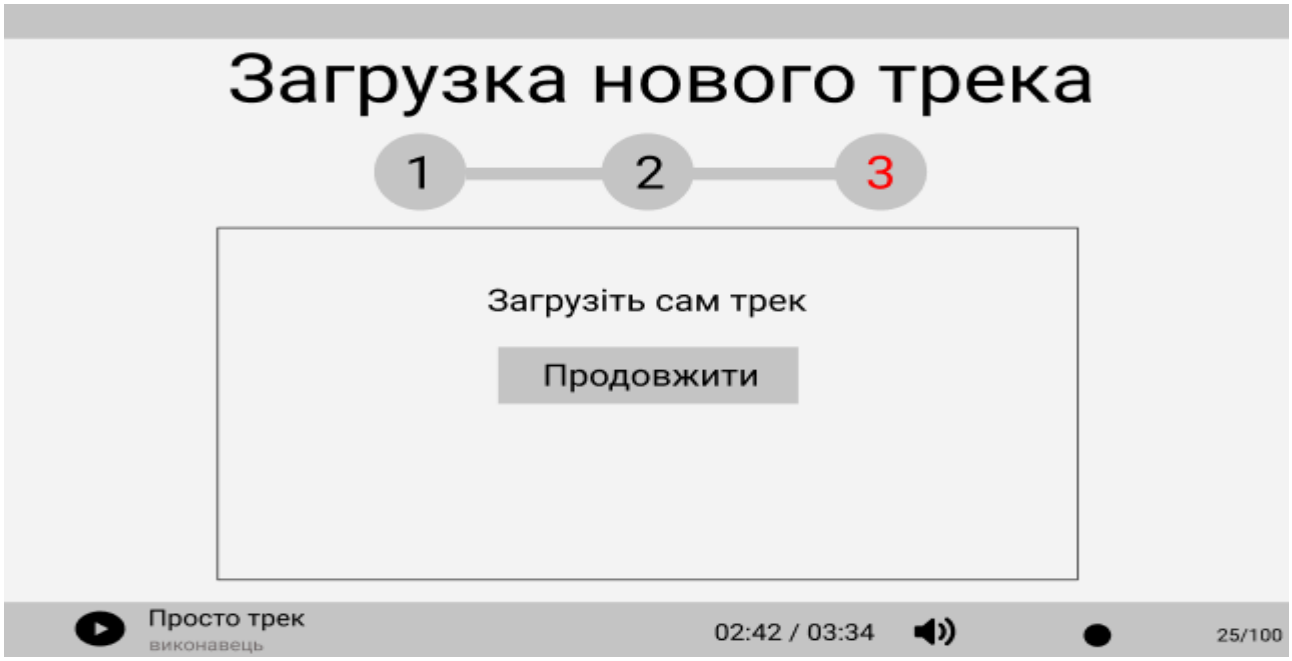

Рисунок 2.14 – Етап форми для завантаження музики

На (Рис. 2.10, 2.11, 2.12, 2.13, 2.14) зображена сторінка для завантаження музики з етапами. Перший етап заповнення форми для полів назва музики, ім’я автора та текст пісні. Другий етап завантаження зображення до музики та відображення стадій завантаження. Третій етап завантаження музики до музики та відображення стадій завантаження.

3 ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЯ ВИМОГ

3.1 Тестування розробленої інформаційної системи

Функціональне тестування ПЗ охоплює стратегію, що спрямована на перевірку, чи задовольняє програмне забезпечення вимогам та очікуванням користувачів, зокрема в контексті музикальної платформи. У цьому виді тестування будуть розглянуті основні принципи та підходи, що застосовуються для перевірки різних функціональних аспектів музикальної платформи.

Додатково будуть розроблені тест-кейси, які дозволять провести систематичну перевірку роботи функціональних компонентів цього веб-застосунку.

Тест-кейс - це мінімальний компонент тесту, як правило, він націлений тільки на один елемент об'єкта тестування. Чим менше у тест-кейса покриття функціональності, тим чіткіше область пошуку причини у випадку знайденої помилки.

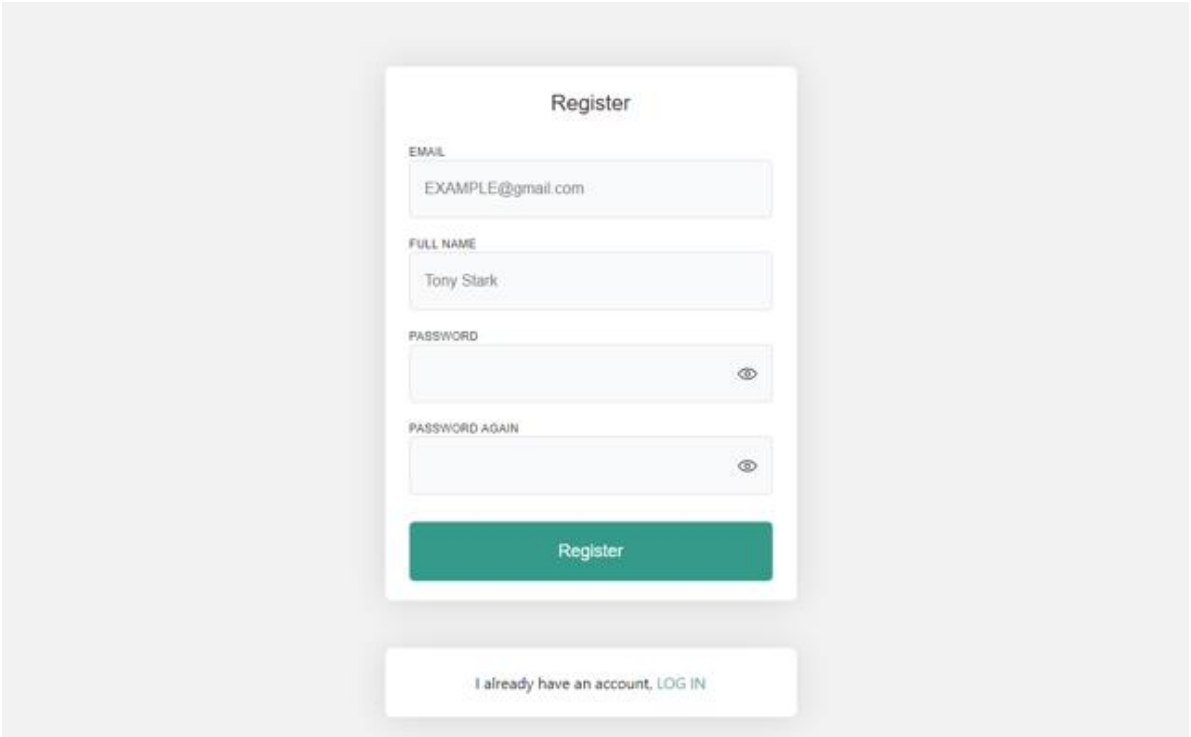

Рисунок 3.1- сторінка реєстрації

Test Case №1

Summary: перевірити, що система перенаправляє користувача на Головну сторінку, якщо всі обов’язкові поля заповнені та “Register” кнопка записує та реєструє нового користувача на сторінці реєстрації в систему.

Preconditions: користувач знаходиться на сторінці реєстрації.

Steps to reproduce:

1. Ввести пошту в “Email” поле.
2. Ввести повне ім’я користувача в “FullName” поле.
3. Ввести пароль у “Password” поле.
4. Ввести пароль у “Confirm Password” поле.
5. Натиснути “Register” кнопку.

Expected Result: система перенаправляє користувача на сторінку продуктів.

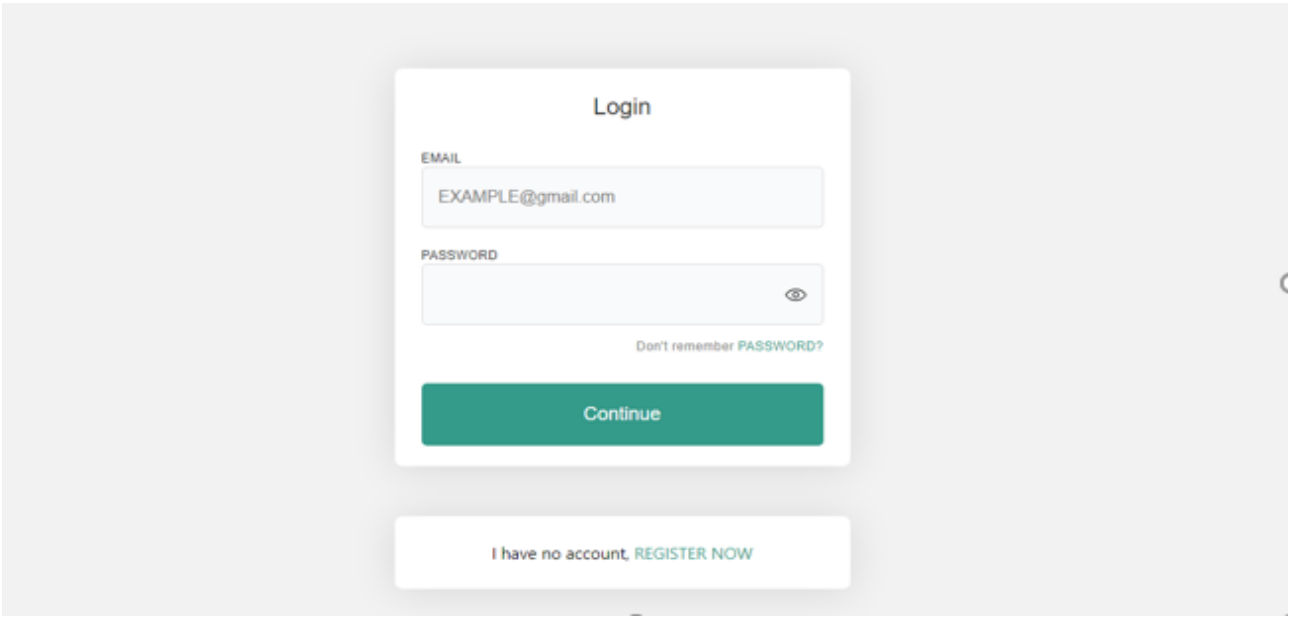

Рисунок 3.2- сторінка входу в профіль

Test Case №2

Summary: перевірити, що система правильно сортує список продуктів, якщо необхідні поля заповнені і кнопка “Search” шукає вказаний елемент в списку.

Preconditions: користувач знаходиться на сторінці списку продуктів.

Steps to reproduce:

1. Ввести ключове слово в “Title” поле.
2. Натиснути “ Search ” кнопку.

Expected Result: система відображає відсортований список продуктів.

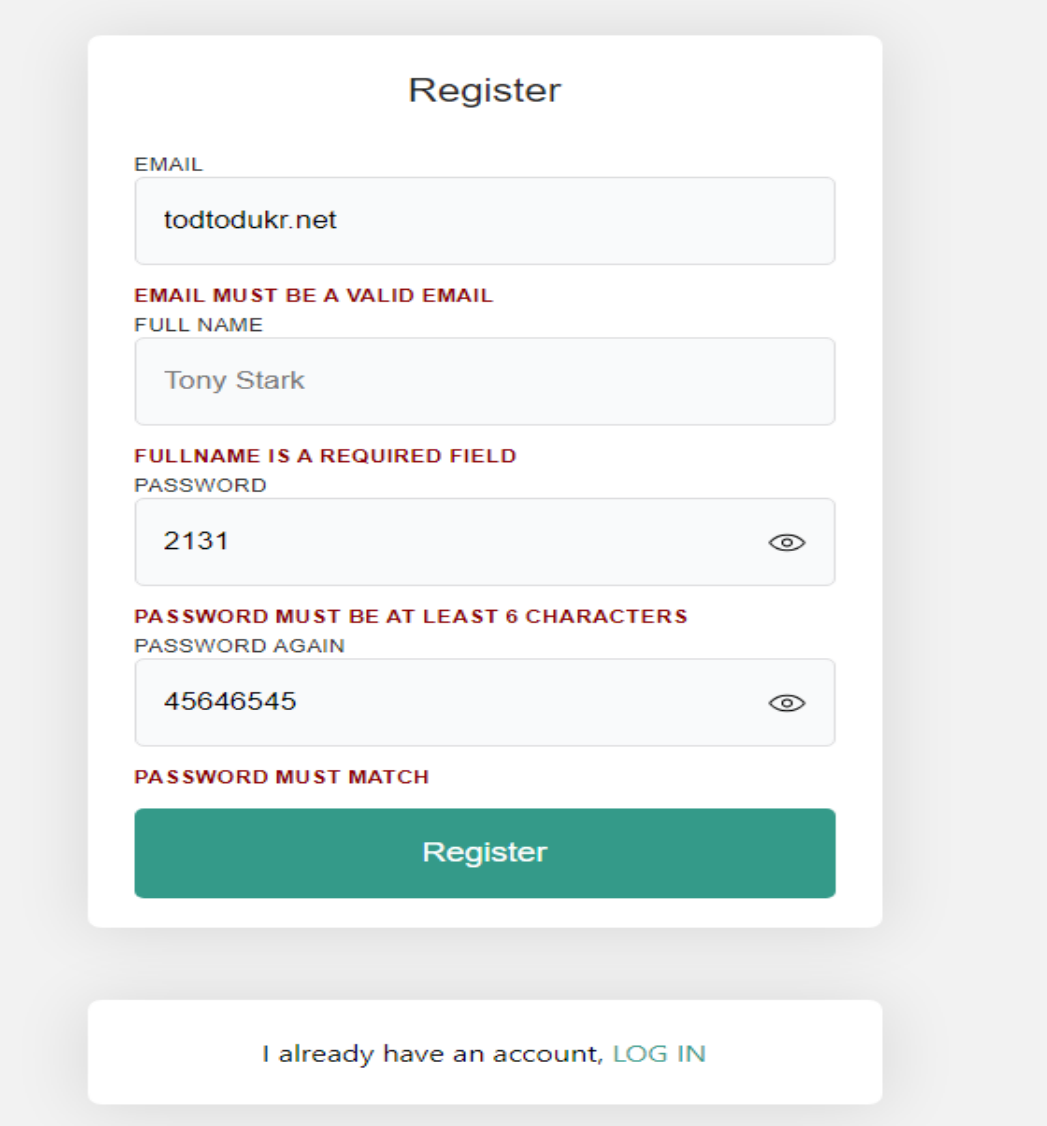

Рисунок 3.3- сторінка реєстрації

Test Case №3

Summary: перевірити, що система перенаправляє користувача на Головну сторінку, якщо всі обов’язкові поля заповнені та “Register” кнопка записує та реєструє нового користувача на сторінці реєстрації в систему.

Preconditions: користувач знаходиться на сторінці реєстрації.

Steps to reproduce:

1. Ввести емейл в “Email” поле.
2. Ввести повне ім’я користувача в “FullName” поле.
3. Ввести пароль у “Password” поле.
4. Ввести пароль у “Confirm Password” поле.
5. Натиснути “Register” кнопку.

Expected Result: система відображає помилки при вводі даних.

Рисунок 3.4- плеєр

Test Case №4

Summary: перевірити, чи коректно працює плеєр та після закінчення музики чи переходить до наступної.

Preconditions: користувач знаходиться на сторінці головній сторінці.

Steps to reproduce:

1. Включити музику з списку.
2. Збільшити звук у плеєрі.
3. Зменшити звук у плеєрі.
4. Натиснути кнопку “Pause”.
5. Натиснути кнопку “Start”.
6. Перемотати музику.
7. Зачекати закінчення музики.

Expected Result: система збільшує, зменшує звук, ставить на паузу, продовжує грати музика, перемотує музику, переключає на наступну музику.

Висновок: Цими "Тест кейсами" було розглянуто важливий аспект тестування програмного забезпечення - створення та виконання тестових кейсів. Тест кейси є документованою процедурою, яка описує кроки для виконання певного тесту, його очікувані результати та вхідні дані. За допомогою покриття програми Тест кейси дозволило забезпечити покриття функціональності програмного забезпечення. Шляхом створення різних тестових сценаріїв та

варіацій, можна переконатися, що програма працює правильно в різних умовах. Використання тест-кейсів допомогло стандартизувати процес тестування, що робить його більш систематичним і ефективним. Тест-кейси надали чіткі інструкції для виконання тестів, що сприяє однаковому підходу до тестування і забезпечує консистентність результатів. Тест-кейси допомогли виявити дефекти та проблеми в роботі програмного забезпечення. Чіткі критерії прийняття дозволяють ідентифікувати неправильну поведінку програми та сприяють виправленню проблем.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Долікарська допомога при шоку

Травматичний шок - складний патогенний процес, що виникає внаслідок важкої механічної травми, опіку і характеризується порушенням функцій життєво важливих органів та систем організму [1].

При комбінованих хімічно-радіаційних ушкодженнях, опіках тканин і органів травматичний шок спостерігається у 30% потерпілих [1].

В розвитку травматичного шоку першочергову роль мають наступні фактори: втрата крові і біль, розлад дихання, порушення процесів мета-болізму, інтоксикація організму недоокисленими продуктами обміну речовин внаслідок руйнування тканин [1].

При пораненнях зміни виникають у підкіркових утвореннях великого мозку та в системі периферійного кровообігу (перерозподіл крові, яка забезпечує життєдіяльність органів, передусім серця і мозку). Розвивається циркулярна гіпонія, судоми посткапілярних венул (випотіває плазма в позаклітинний простір), набряк і згущення крові. Знижується венозний тиск, слабнуть нирки, печінка, легені, відбувається тромбоутворення, розвиток незворотних змін в органах [1].

Травма кишечника призводить до інтоксикації організму, ускладнюється стан пораненого, спостерігається розлад дихання і кровообігу. Порушується функція нервової системи [1].

Шок має дві фази: еретильну і торпідну.

Еретильна фаза шоку - фаза збудження. Спостерігається надмірна рухливість, мова уривчаста, погляд неспокійний, шкірний покрив блідий, іноді виникає гіперемія (різке потовиділення), пульс відхилений від норми сповільнений або прискорений. Дихання часте, поверхневе [1].

Торпідна фаза шоку - фаза пригнічення; розрізняють 4 ЇЇ ступеня:

- І ступінь (легка форма шоку). Це результат ізольованих поранень се-редньої важкості та втрати 500-1000 мл крові. Стан помірного психічного

гальмування, блідий шкірний покрив, артеріальний тиск 100-95 мм. рг. ст. Прогноз сприятливий [1].

- II ступінь (шок середньої важкості). Численні ушкодження тіла, втра-та крові до 1000-1500 мл, стан важкий, хоч орієнтація і свідомість не втра-чені, шкіра бліда, губи ціанотичні, психічна загальмованість, пульс -110-130 пульсацій за хвилину, тиск - 90-75 мм. рт. ст. нестійкий. Прогноз сприятливий при проведенні протишокової терапії[1].
- III ступінь (важкий шок). Виникає при важких ушкодженнях грудної клітки, черевної порожнини. Крововтрата - до 2000 мл, стан важкий, ви-ражена психічна загальмованість, іноді ступор. Шкіра бліда ціанотична, пітніє, слизові оболонки сухі, гіпотермія, зниження сухожильних рефлексів, розлад нирок, сечовиділення. Пульс - 120-160 пульсацій за хвилину, тиск - 75 мм.рт.ст., дихання поверхневе. Без протишокових заходів прогноз несприятливий[1].
- VI ступінь - термінальний стан (передагональний, агональний та клінічна смерть), - вкрай важкий для потерпілого. Втрата свідомості, шкіра холодна, трупна, ціанотична, вкрита липким холодним потом, зіниці розширені, не реагують на світло, пульс не промацується, крайній ступінь шоку призводить до клінічної смерті [1].

Діагностика шоку ґрунтується на визначенні показників, які характеризують загальний стан потерпілого. Найважливіший показник - рівень артеріального тиску. Що нижчим він є, то глибший розлад функцій організму, його життєдіяльності. Величина крововтрати - найоб'єктивніший показник ступеня важкості шоку [1].

Перебіг клінічного шоку в залежності від локалізації поранення чи опіку має такі особливості: проникаючі поранення черевної порожнини до 80% спричиняють шок; проникаючі поранення грудної клітки зумов-люють гемоторакс, відкритий пневмоторакс. При пораненнях і ушкоджен-нях тазу - крововтрата - до 2,5 літра. При ушкодженнях кінцівок крововтрата до 2 літрів, біль, інтоксикація. Велика крововтрата (до 50%) веде до смерті. Крововтрата організмом до 700 мл

компенсується самостійно, за рахунок інфузії плазми крові, введення сольових багатокомпонентних розчинів. Рівень гемоглобіну повинен бути 65% [1].

Профілактична і долікарська допомога при шоку. Під час шоку усувають дію травмуючих факторів і факторів розвитку шоку, зупиняють кровотечу, перев'язують рани, усувають загрозу асфікції; вводять S-подібну трубку (повітропровід); при порушенні зовнішнього дихання в долікарську допомогою входить очищення порожнини рота і но-соглотки, усунення западання язика, відновлення прохідності дихальних шляхів; при пневмотораксі накладається пов'язка; проводиться інгаляція киснем, зупинка зовнішньої кровотечі; вводяться серцево-судинні і аналектичні засоби (виконує фельдшер); здійснюється іммобілізація кінцівок. Ввівши повторно знеболювальні засоби, дають гарячий чай та інші напої [1].

Синдром тривалого стискання тканин, зокрема тканин верхніх і нижніх кінцівок, буває внаслідок землетрусів, коли люди опиняються під уламками споруд і будинків. Синдром тривалого стискання може спостерігатися поряд з переломами, опіками та іншими ушкодженнями організму. При розтрощенні і роздавлюванні тканин різко погіршується кровообіг в м'язах, виникає анемія, гіпонія тканин, інтоксикація, нервово-рефлекторний розлад, судоми капілярів, артерій, гостра серцево-судинна недостатність, набряки. Плазма крові пропотіває в міжклітинний простір (обсяг циркулюючої плазми зменшується на 50%), зменшується артеріальний тиск, може настати гостра ниркова недостатність і порушення сечовиділення [1].

Синдром тривалого стискання тканин характеризується трьома періодами:

1 період (ранній). Спостерігаються набряки тканин і гострий розлад гемодинаміки. Триває 1-3 доби.

2 період (проміжний). Гостра ниркова недостатність тривалістю від 5 діб до 1,5 місяців.

3 період (пізній). Супроводжується гангреною, флегмонами, абсцесами[1].

Кінцівки потерпілого набрякають, шкіра багряно-синя, іноді виникають пухирі з бурштиново-жовтою рідиною, пульсація послаблена або відсутня,

чутливість шкіри знижена або втрачена. Відбувається згущення крові. Погіршується загальний стан організму. Холодний піт на шкірі, різкий біль на місці травми, нудота і блювання. Пульс - 100-120 пульсацій за 1 хвилину, тиск 60 мм. рт. ст. Сечовиділення червоного кольору. Тип клініки торпідної фази травматичного шоку. Наростає загальна інтоксикація організму, гостра ниркова недостатність, іноді гангрена кінцівки, абсцеси і флегмони, може виникнути атрофія м'язів. Ускладнюється рухливість суглобів, пошкоджуються нервові стовбурих [1].

Існує 4 ступеня прояву синдрому стискання:

I ступінь (дуже важкий). Стискання м'яких тканин або кінцівок протягом 6-8 годин. Потерпілі, як правило гинуть через 2-3 доби;

II ступінь (важкий). Стискання рук або ніг протягом 4-7 годин, потерпілі можуть загинути;

III ступінь (середньої важкості). Стискання рук або ніг до 6 годин. Лікування до 3 місяців;

IV ступінь (легкий). Стискання рук або ніг до 2 годин. Порушення помірні. Прогноз сприятливий [1].

Накладається джгут (вище від місця стискання). Вводяться знеболювальні, антигістомічні та серцево-судинні препарати, призначаються антибіотики, проводять правцеве щеплення [1].

Кваліфікована медична допомога надається в лікарні.

а) Цей Порядок визначає механізм надання домедичної допомоги постраждалим при підозрі на шок особами, які не мають медичної освіти, але за своїми службовими обов’язками повинні надавати домедичну допомогу.

б) Ознаки шоку:

1) бліда, холодна і волога шкіра;

2) загальна слабкість;

3) неспокій, роздратованість;

4) сухість в роті, відчуття спраги;

5) часте або повільне дихання;

6) порушення свідомості, непритомність.

в) Причинами виникнення шоку можуть бути:

1) масивна зовнішня кровотеча;

2) внутрішня кровотеча;

3) травми різного ґенезу;

4) анафілаксія;

5) серцевий напад тощо.

г) Послідовність дій при наданні домедичної допомоги постраждалим при підозрі на шок:

1) перед наданням допомоги переконатися у відсутності небезпеки та за її відсутності перейти до наступного кроку;

2) заспокоїти постраждалого та пояснити свої подальші дії;

3) здійснити виклик екстреної медичної допомоги та дотримуватись вказівок диспетчера прийому виклику;

4) за можливості виявити та усунути причину виникнення шоку;

5) надати постраждалому протишокове положення:

i) перевести постраждалого в горизонтальне положення, якщо це не погіршує його дихання;

ii) покласти під ноги постраждалого ящик, валик з одягу тощо таким чином, щоб ступні ніг знаходились на рівні його підборіддя;

iii) підкласти під голову постраждалого одяг/подушку, якщо це не погіршує його дихання;

д) вкрити постраждалого термопокривалом/ковдрою;

е) забезпечити постійний нагляд за постраждалим до приїзду бригади екстреної (швидкої) медичної допомоги;

ж) при погіршенні стану постраждалого до приїзду бригади екстреної (швидкої) медичної допомоги, повторно здійснити виклик екстреної медичної допомоги;

з) за можливості зібрати у постраждалого максимально можливу інформацію, стосовно обставин травми та механізму її отримання. Всю отриману інформацію передати працівникам бригади екстреної (швидкої) медичної допомоги або диспетчеру служби екстреної медичної допомоги [2].

Враховуючи сказане вище, можна сказати, що основні етапи надання долікарської допомоги при шоку є такі пункти:

а) Переконатись у відсутності небезпеки;

б) Провести огляд постраждалого, визначити наявність свідомості, дихання;

в) якщо у постраждалого відсутнє дихання, розпочати проведення серцево-легеневої реанімації;

г) усунути причину виникнення шокового стану: зупинити кровотечу, іммобілізувати перелом тощо;

д) надати постраждалому протишокове положення:

1) перевести постраждалого в горизонтальне положення;
2) покласти під ноги постраждалого ящик, валик з одягу тощо таким чином, щоб ступні ніг знаходились на рівні його підборіддя;
3) підкласти під голову постраждалого одяг/подушку;
4) вкрити постраждалого термопокривалом/покривалом;

е) забезпечити постійний нагляд за постраждалим до приїзду бригади екстреної (швидкої) медичної допомоги;

ж) при погіршенні стану постраждалого до приїзду бригади екстреної (швидкої) медичної допомоги повторно зателефонувати диспетчеру екстреної медичної допомоги [2].

4.2 Розрахунок евакуаційних шляхів із виробничих приміщень дільниці (цеху)

Аналіз об’ємно-планувальних рішень, що впливають на визначення розрахункової схеми евакуації людей Для проведення аналізу об’ємно-планувальних рішень здійснюється збирання даних:

- про кількість і місця ймовірного розміщення людей;
- про евакуаційні шляхи і виходи;
- про можливі напрямки евакуації людей при пожежі. На підставі аналізу об’ємно-планувальних рішень визначають напрямки евакуації людей та розробляють розрахункові схеми евакуації [3].

Вибір розрахункового методу Вибір розрахункового методу тривалості евакуації визначається з урахуванням специфічних особливостей об’ємно-планувальних рішень будинку, особливостей контингенту людей, що знаходяться в ньому, характеристик людських потоків, параметрів шляхів евакуації [3].

Визначення розрахункової тривалості евакуації людей із приміщень будівель і споруд із застосуванням спрощеної аналітичної моделі руху людського потоку Розрахункова тривалість евакуації людей tp із приміщень і будівель встановлюється за розрахунком тривалості руху одного чи декількох людських потоків через евакуаційні виходи від найвіддаленіших місць розташування людей. Під час розрахунку весь шлях руху людського потоку поділяється на ділянки (прохід, коридор, дверний проріз, сходовий марш, тамбур) завдовжки li і завширшки bi. Початковими ділянками є проходи між робочими місцями, обладнанням, рядами крісел тощо. Під час визначення розрахункового часу евакуації людей довжину і ширину кожної ділянки шляху евакуації для будівель,

які проектуються, приймають згідно з проектом, а для побудованих за фактичним значенням. Довжину шляху сходовими маршами, а також по пандусах вимірюють по довжині маршу. Довжину шляху в дверному прорізі приймають рівній нулю. Проріз, розташований у стіні завтовшки 11 більше ніж 0,7 м, а також тамбур слід вважати самостійними ділянками горизонтального шляху, що мають кінцеву довжину li [3].

Рисунок 4.1 - Порядок визначення розрахункового часу евакуації людей із будівлі (моделювання процесу евакуації)

На (Рис 4.1) зображена схема, що описує процес моделювання евакуації людей з будівлі. Отже процес продовжується до завершення евакуації коли усі люди залишають будівлю, або продовжується, якщо евакуація ще не завершена.

ВИСНОВОК

Результатом курсової роботи та в детальний аналіз вимог до програмного забезпечення ,що охоплює, як функціональні, так і різні нефункціональні вимоги. Цей аналіз дозволив нам визначити основні аспекти, необхідні для успішного впровадження інформаційної системи.

Основні варіанти використання системи визначені відповідно до встановлених вимог і представлені у вигляді повної схеми випадків використання з використанням мови UML. Це сприяло кращому розумінню взаємодії між Користувачем та системою та функціональності.

В процесі розробки була обрана Архітектура веб-системи, що включає технології та інструменти для реалізації продукту. Важливим фактором є розробка бази даних для зберігання інформації користувача та включення медіа-сховища для ефективного зберігання зображень та аудіо файлів, завантажених користувачами.

Клієнтська і серверна частини веб-додатки були створені і взаємодіяли один з одним. Цей модуль інтегрований з компонентами React за допомогою Next.js відображає, як всю сторінку, так і окремі компоненти, забезпечуючи високу модульність та повторне використання коду.

На заключному етапі розробки інформаційної системи функціональність всієї системи була успішно протестована і перевірена для забезпечення високої якості, стабільності та відповідності функціональним вимогам. Такий підхід дозволяє створювати високопродуктивні і масштабовані системи, що відповідають потребам користувачів і забезпечують надійну роботу в реальних умовах експлуатації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Долікарська допомога при шоку [Електронний ресурс] Тернопільський національний технічний університет ім. І. Пулюя, Серт. №0273 Гурик О.Я., Окіпний І.Б.. – 2018. – Режим доступу до ресурсу: https://dl.tntu.edu.ua/content.php?cid=299870.

2. ПОРЯДОК надання домедичної допомоги постраждалим при підозрі на шок [Електронний ресурс] Міністерства охорони здоров'я України. – 2022. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0356-22#n543.

3. НАЦІОНАЛЬНИЙ СТАНДАРТ УКРАЇНИ ДСТУ 8828:2019 ПОЖЕЖНА БЕЗПЕКА Загальні положення [Електронний ресурс] Київ ДП «УкрНДНЦ». – 2020. – Режим доступу до ресурсу: https://zakon.isu.net.ua/sites/default/files/normdocs/dstu_8828_2019.pdf.

4. Node.js documentation [Електронний ресурс] // OpenJS Foundation. – 2021. – Режим доступу до ресурсу: https://nodejs.org/docs/latest/api/.

5. Next.js Documentation [Електронний ресурс] Vercel, Inc. – 2024. – Режим доступу до ресурсу: https://nextjs.org/docs.

6. Nest.js documentation [Електронний ресурс] Kamil Mysliwiec. – 2023. – Режим доступу до ресурсу: https://docs.nestjs.com/.

7. PostgreSQL Documentation [Електронний ресурс] The PostgreSQL Global Development Group. – 2023. – Режим доступу до ресурсу: https://www.postgresql.org/docs/current/.

8. Feature-Sliced Design [Електронний ресурс] Feature-Sliced Design. – 2023. – Режим доступу до ресурсу: https://feature-sliced.design/docs/reference.

9. MVC (Model-View-Controller) [Електронний ресурс] mozilla.org. – 2024. – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Glossary/MVC.

ДОДАТКИ

ДОДАТОК А

Диск з розробленою програмою