

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ "ЖИТОМИРСЬКА ПОЛІТЕХНІКА"
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Веб-додаток для прослуховування подкастів»

Виконав студент групи ІПЗ-20-1
БІЛИЙ Олександр Васильович

Керівник роботи:
КУБРАК Юрій Олександрович

Консультант:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
МОРОЗОВ Андрій Васильович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: БІЛИЙ Олександр Васильович

Керівник роботи: КУБРАК Юрій Олександрович

Тема роботи: «Веб-додаток для прослуховування подкастів»,

затверджена Наказом закладу вищої освіти від «23» лютого 2024 р., №74/с

Вихідні дані для роботи: **методи та засоби розробки веб-додатків, аналіз проблематики додатку для прослуховування подкастів використання для розробки веб-додатку веб-фреймворку Next.JS.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Кубрак Ю.О., Єфремов Ю.М.	27.02.2024р.	27.02.2024р
2	Кубрак Ю.О., Єфремов Ю.М.	27.02.2024р	27.02.2024р
3	Кубрак Ю.О., Єфремов Ю.М.	27.02.2024р	27.02.2024р

РЕФЕРАТ

Випускна робота бакалавра складається з програмного комплексу додатка прослуховування подкастів і пояснювальної записки. Пояснювальна записка до випускної роботи містить 85 сторінок, 57 ілюстрацій, 16 таблиць, 27 використаних джерел та 16 сторінок додатків.

Метою роботи є розробка веб-додатку для прослуховування подкастів за допомогою фреймворку Next.js, що дозволить шукати, зберігати та додавати подкасти. У роботі сформульовано ключові завдання для створення системи, проведено аналіз аналогів існуючих аналогів, вибрано оптимальні інструменти для реалізації. Особлива увага приділена розробці структури бази даних та вибору сучасного стеку технологій. Описано концепцію програмного продукту, включаючи деталі його реалізації, результати випробувань та валідації розроблених рішень.

КЛЮЧОВІ СЛОВА: ВЕБДОДАТОК, ФРЕЙМБОРК, ПОДКАСТИ, ЗВУК, АУДИО, NEXT.JS, TYPESCRIPT, TAILWIND, POSTGRES, POSTGRESQL

					ІПЗ.КР.Б – 121 – 24 – ПЗ				
Змн.	Арк.	№ докум.	Підпис	Дата					
Розроб.		Білий О.В.			Веб-додаток для прослуховування подкастів.		Літ.	Арк.	Аркушів
Керівник		Кубрак Ю. О.						3	102
Консульт.		Єфремов Ю. М.					Житомирська політехніка, група ІПЗ-20-1		
Рецензент		Морозов А. В.							
Зав. каф.		Вакалюк Т.А.							

ABSTRACT

Bachelor's thesis consists of a software complex of a podcast listening application and an explanatory note. The explanatory note for the bachelor's thesis contains 85 pages, 57 illustrations, 16 tables, 27 cited sources, and 16 pages of appendices.

The purpose of the work is to develop a web application for podcast listening using the Next.js framework, which will allow searching, saving, and adding podcasts. The thesis outlines the key tasks for the system's creation, analyzes existing analogs, selects optimal tools for implementation. Special attention is given to developing the database structure and choosing a modern technology stack. The concept of the software product is described, including details of its implementation, test results, and validation of the developed solutions.

KEYWORDS: WEB APPLICATION, FRAMEWORK, PODCASTS, AUDIO, NEXT.JS, TYPESCRIPT, TAILWIND, POSTGRES, POSTGRESQL

					ПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБ-ДОДАТКУ ДЛЯ ПРОСЛУХОВУВАННЯ ПОДКАСТІВ	9
1.1. Постановка задачі	9
1.2. Аналіз аналогів програмного продукту	11
1.3. Визначення архітектури веб-додатку для прослуховування подкастів	16
Висновок до першого розділу	19
РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-ДОДАТКУ ДЛЯ ПРОСЛУХОВУВАННЯ ПОДКАСТІВ	20
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи	20
2.2. Розробка бази даних системи	29
2.3. Проектування та реалізація алгоритмів роботи системи	32
2.4. Реалізація функціоналу веб-додатку для прослуховування подкастів	37
Висновок до другого розділу	46
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ДОДАТКОМ ДЛЯ ПРОСЛУХОВУВАННЯ ПОДКАСТІВ	47
3.1. Порядок встановлення та налаштування параметрів додатку	47
3.2. Структура інтерфейсу веб-додатку для прослуховування подкастів	48
3.3. Тестування роботи веб-додатку	76
Висновок до третього розділу	80
ВИСНОВКИ	81
Джерела інформації	83
ДОДАТКИ	86

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

JSON – (JavaScript Object Notation) Запис об'єктів JavaScript

SQL – (Structured Query Language) Мова структурних запитів

UML – (Unified Modeling Language) Уніфікована мова моделювання

OMG – (Object Management Group) Група з управління об'єктами

JWT – (JSON Web Token) Токен на основі JSON

ISR – (Incremental Static Regeneration) Поступове статичне відтворення

SSR – (Server-Side Rendering) Обробка на стороні сервера

ВСТУП

Актуальність теми полягає у тому, що подкасти стають все більш популярним форматом споживання контенту завдяки своїй зручності та доступності. Люди можуть слухати подкасти під час поїздок, занять спортом або виконання домашніх справ, що робить їх ідеальним засобом для отримання інформації та розваг. Відсутність необхідності візуального контакту з контентом дозволяє слухачам багатозадачно виконувати інші завдання, що значно підвищує продуктивність і комфорт. Зростання інтернет-технологій та доступності мобільних пристроїв також сприяє поширенню подкастів. У світі, де люди все частіше обирають споживання контенту в цифровому форматі, подкасти стають невід'ємною частиною повсякденного життя. Це відображається у збільшенні кількості платформ, що спеціалізуються на подкастах, та появі нових, більш спеціалізованих жанрів і форматів аудіоконтенту. Більш того, сучасні користувачі прагнуть до мобільності і доступу до контенту в будь-який час і в будь-якому місці. Веб-додатки, які можуть працювати на різних пристроях, включаючи смартфони та планшети, забезпечують таку мобільність і відповідають потребам користувачів. Завдяки цьому, веб-додатки для прослуховування подкастів стають незамінними інструментами для багатьох людей. Крім того, такі додатки забезпечують інтеграцію з соціальними мережами, що дозволяє користувачам ділитися улюбленими епізодами та відкривати для себе новий контент через рекомендації друзів і знайомих.

Метою випускної кваліфікаційної роботи бакалавра є створення веб-додатка для прослуховування подкастів.

Для реалізації поставленої мети потрібно виконати наступні завдання:

1. Провести аналіз існуючих рішень, щоб визначити основні функції та переваги, які мають бути реалізовані.
2. Вибрати відповідні технології для розробки, що забезпечать продуктивність, безпеку та зручність використання.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						7

3. Розробити структуру системи, використовуючи методи об'єктно-орієнтованого проектування та діаграми для кращого розуміння цілей.
4. Створити та налаштувати базу даних для надійного зберігання та швидкого доступу до даних.
5. Реалізувати функціонал додатка для різних типів користувачів з урахуванням їхніх потреб.
6. Розробити інтерфейс користувача, який буде зручним та інтуїтивно зрозумілим.
7. Провести всебічне тестування додатка для виявлення потенційних помилок та забезпечення високої якості коду.

Об'єктом дослідження є процес створення та функціонування веб-додатка для прослуховування подкастів. Вивчення цього процесу включає розгляд різних технологій і підходів, що можуть бути використані при розробці веб-додатків, а також аналіз найкращих практик в цій галузі.

Предметом дослідження є функціональність та ефективність веб-додатка для прослуховування подкастів, зокрема його інтерфейс, швидкість роботи, надійність та можливості управління контентом. Важливими характеристиками додатка є його здатність забезпечувати безперебійне відтворення аудіо та адаптивність до різних розмірів екрану та операційних систем. Дослідження цих аспектів дозволить визначити ключові фактори успіху веб-додатків для подкастів та шляхи їхнього вдосконалення.

Для розробки веб-додатку було застосовано веб-фреймворк Next.js та платформу Node.js у поєднанні з об'єктно-реляційною системою керування базами даних PostgreSQL.

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБ-ДОДАТКУ ДЛЯ ПРОСЛУХОВУВАННЯ ПОДКАСТІВ

1.1. Постановка задачі

Метою кваліфікаційної роботи бакалавра є створення веб-додатка, який дозволить користувачам зручно прослуховувати та керувати подкастами. Додаток буде включати функціональність потокового відтворення аудіо, пошук та відображення подкастів за категоріями, можливість додавання подкастів у свої закладки, а також інші корисні функції.

На початку 2000-х років з'явилась технологія RSS, або ж Really Simple Syndication, яка дає змогу отримувати оновлення від улюблених веб-сайтів безпосередньо на свої пристрої, забезпечуючи зручний спосіб перегляду новин та іншого контенту. Вона сприяла поширенню аудіофайлів, допомагаючи користувачам розповсюджувати власні аудіоблоги в маси. Проте, на той момент, через технічні обмеження та недостатню обізнаність користувачів, про аудіоблоги мало хто говорив, і їх мало хто слухав [1].

Вперше термін «подкаст» з'явився у британському журналі «The Guardian» у 2004 році і став популярним завдяки об'єднанню популярного на той час музичного програвача компанії Apple iPod та слова broadcast, що перекладається з англійського як трансляція, мовлення.

У 2014 році відбувся випуск аудіо-серіалу під назвою «Serial», який став справжнім проривом у світі аудіо-контенту. Цей серіал здобув величезну популярність і зацікавив широке коло слухачів своєю захоплюючою історією та унікальним форматом. «Serial» відкрила нові можливості для подкастів, демонструючи їх потенціал як засобу розповіді цікавих історій та провокативних дискусій. З цього моменту почався справжній бум в інтересі до подкастів, і кількість

прослуховувань зростала з кожним днем, відкриваючи нові горизонти для аудіо-контенту в Інтернеті.

На сьогоднішній день світ аудіо-контенту переживає справжню експансію та розмаїття. Подкасти стали не лише популярним джерелом інформації та розваги, але й важливим інструментом для навчання, розвитку та спілкування. Існує безліч тематик і жанрів подкастів, що задовольняють різноманітні інтереси та потреби аудиторії - від новин і наукових досліджень до розважальних шоу та історичних розповідей.

У кваліфікаційній роботі необхідно створити веб-додаток для прослуховування подкастів, який відповідатиме сучасним вимогам до зручності, швидкості роботи і безпеки, забезпечуючи інтуїтивно зрозумілий інтерфейс для ефективної взаємодії з контентом.

Для реалізації поставленого завдання необхідно:

1. Детальний аналіз потреб цільової аудиторії. Визначення характеристик та вимог користувачів щодо функціоналу додатка, їхні уподобання та очікування.
2. Встановлення вимог до функціоналу. Розробка переліку основних і додаткових функцій додатка, які мають бути реалізовані для задоволення потреб користувачів і відповідності сучасним стандартам
3. Розробка інтуїтивно зрозумілого інтерфейсу користувача. Створення дизайну інтерфейсу, який буде зручним у використанні для широкого кола користувачів, з урахуванням адаптивності для різних типів пристроїв.
4. Побудова архітектури клієнтської та серверної частин додатка. Проектування і реалізація ефективної архітектури для фронтенд- і бекенд-частин додатка
5. Розробка системи автентифікації та авторизації. Впровадження системи реєстрації та входу для користувачів з різними рівнями доступу та правами.
6. Реалізація основного функціоналу. Розробка основних функцій додатка, такі як пошук, закладки, завантаження та потокове відтворення подкастів та інші.

7. Тестування і оптимізація. Проведення ретельного тестування додатка на різних пристроях та операційних системах для виявлення та усунення потенційних проблем.

Результатом реалізації кваліфікаційної роботи є веб-додаток, в якому користувачі зможуть легко та зручно прослуховувати подкасти, додавати подкасти в списки закладок та ділитися контентом у соціальних мережах.

1.2. Аналіз аналогів програмного продукту

Для розробки якісного веб-додатка для прослуховування подкастів важливо провести аналіз існуючих аналогів. Це дозволить визначити сильні та слабкі сторони конкурентів, а також виявити функції, які можуть бути корисними для користувачів і зробити додаток більш конкурентоспроможним.

Одним з таких аналогів є платформа Spotify (рис. 1.1) [2]

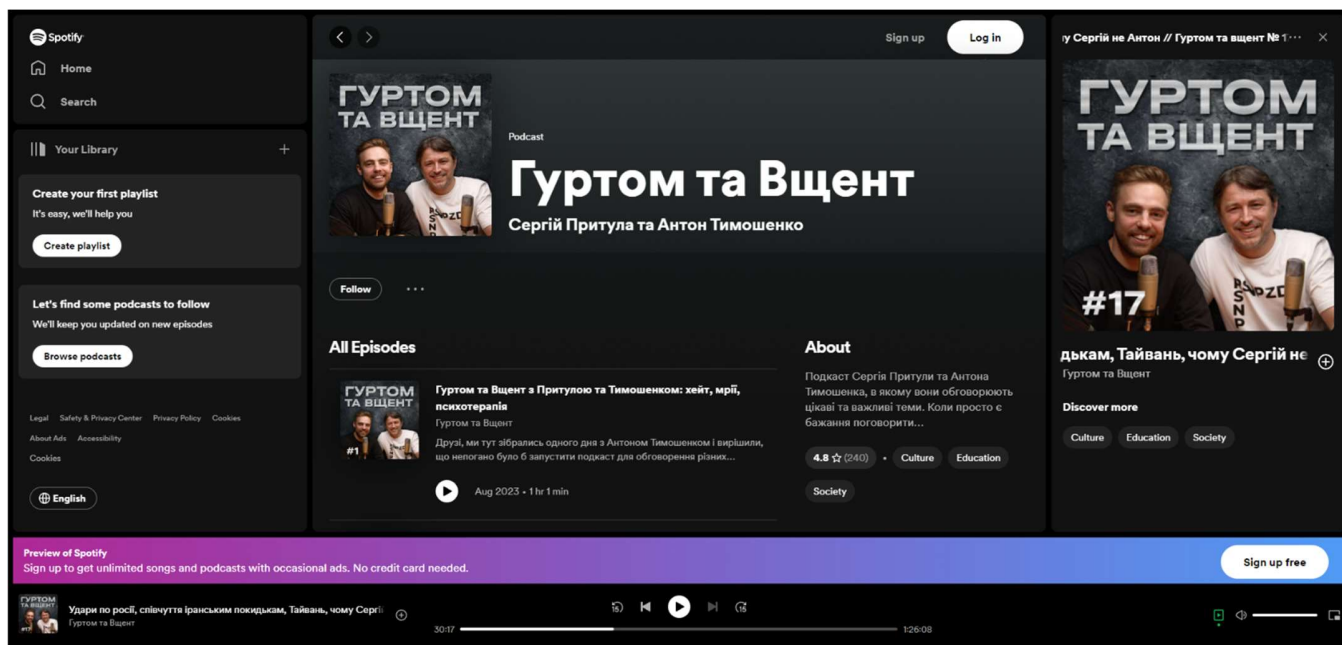


Рис. 1.1. Платформа Spotify

Spotify – це одна з найпопулярніших платформ для стрімінгу музики і подкастів. Вона пропонує великий каталог подкастів, зручний інтерфейс і можливість створювати плейлисти.

Основними перевагами Spotify є:

- Величезна база подкастів і музики;
- Зручний інтерфейс користувача;
- Можливість створення та обміну плейлистами;
- Алгоритми рекомендацій, що допомагають відкривати новий контент.

З недоліків можна виділити:

- Висока вартість преміум-підписки;
- Обмежені можливості для створення подкастів;

Apple Podcasts є стандартним додатком для прослуховування подкастів на пристроях Apple. Вона має великий каталог подкастів і інтеграцію з екосистемою Apple (рис. 1.2) [3].

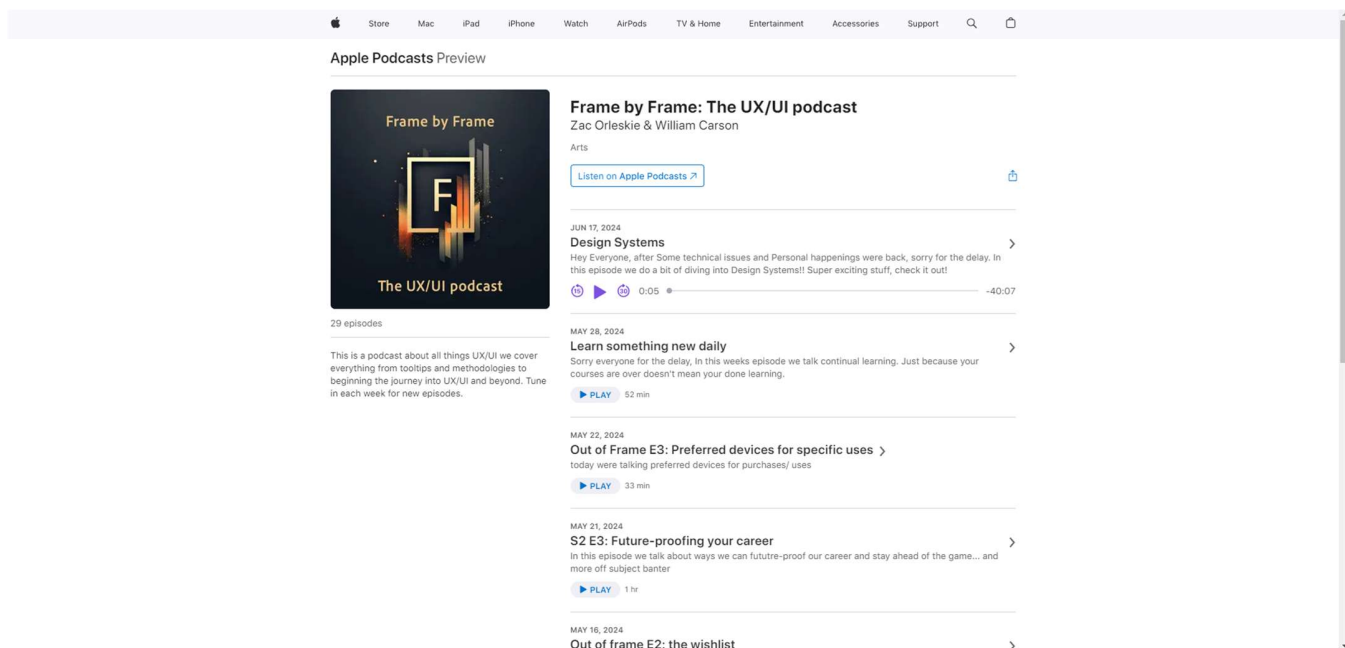


Рис. 1.2. Платформа Apple Podcasts

Основними перевагами Apple Podcasts є:

- Інтеграція з іншими сервісами Apple
- Простий та зрозумілий інтерфейс
- Висока якість аудіо
- Безкоштовне використання

З недоліків можна виділити:

- Обмежені можливості налаштування
- Недоступність для користувачів інших операційних систем (без сторонніх додатків)

Google Podcasts пропонує доступ до великої кількості подкастів і інтеграцію з іншими сервісами Google. Вона підтримує як Android, так і iOS (рис. 1.3) [4].

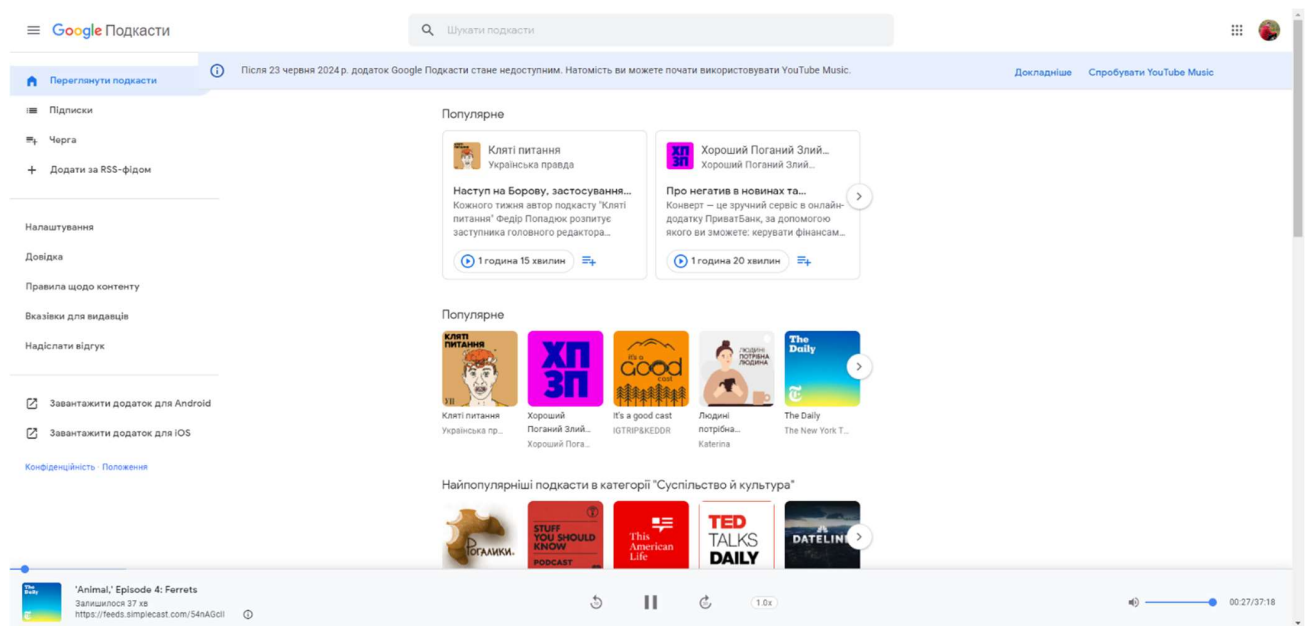


Рис. 1.3. Платформа Google Podcasts

Основними перевагами Apple Podcasts є:

- Безкоштовне використання
- Інтеграція з іншими сервісами Google
- Легкість у використанні
- Підтримка завантаження подкастів для офлайн-прослуховування

З недоліків можна виділити:

- Обмежені можливості персоналізації
- Менше функцій у порівнянні з конкурентами

Слід зауважити, що після 23 червня 2024 року Google Podcasts більше не будуть доступні користувачам і будуть закриті. Усі подкасти будуть інтегровані в один додаток – YouTube Music.

Overcast – це популярний додаток для прослуховування подкастів, орієнтований на користувачів iOS. Відомий своїм зручним інтерфейсом і розширеними функціями для управління подкастами (рис. 1.4) [5].

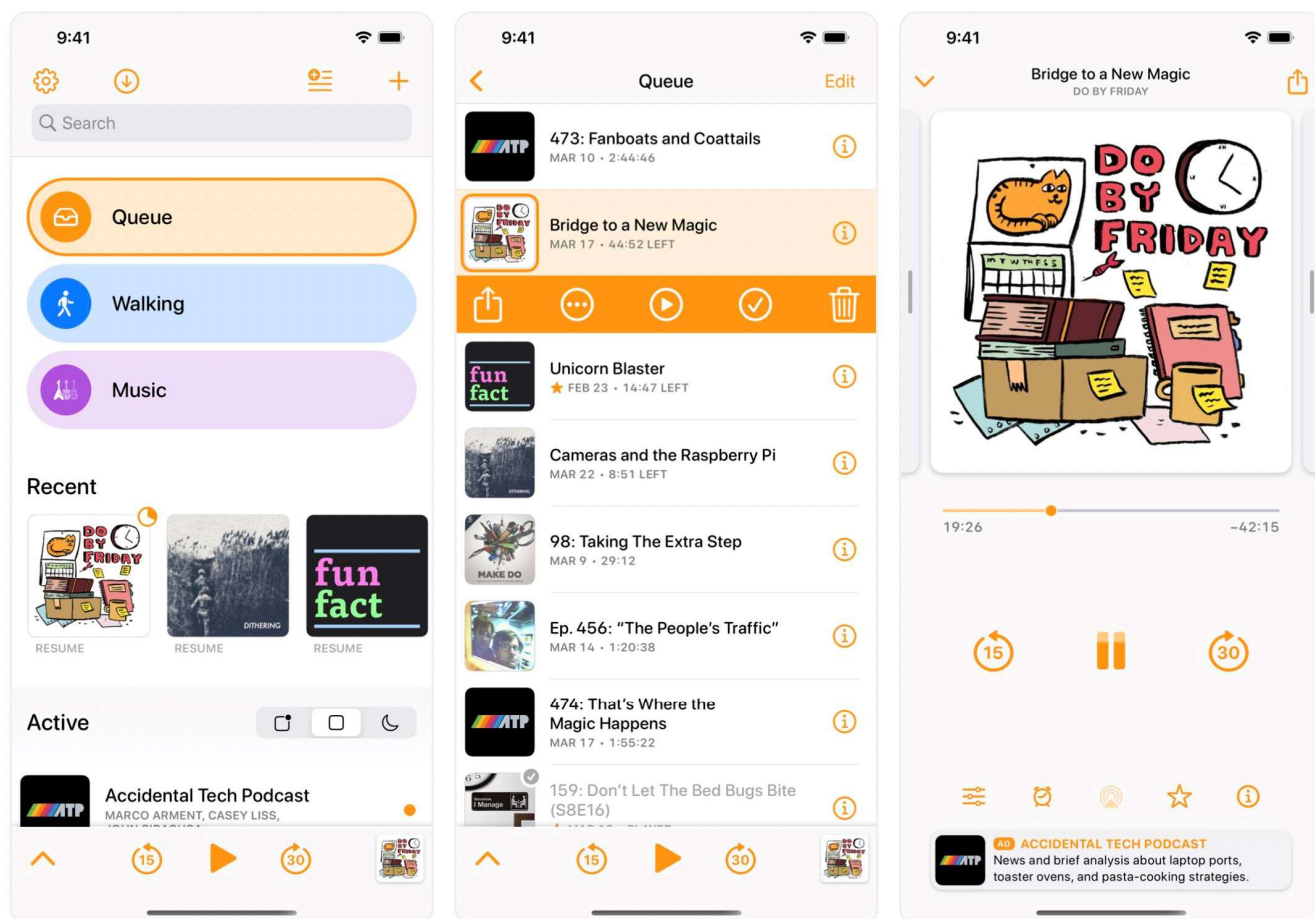


Рис. 1.4. Мобільна версія платформи Overcast

З переваг можна виділити:

- Зручний і чистий інтерфейс
- Підтримка розширених функцій, таких як Smart Speed та Voice Boost
- Можливість завантаження подкастів для офлайн-прослуховування
- Безкоштовна версія з мінімальною рекламою

Основними недоліками є:

- Доступний лише для користувачів iOS

– Відсутність деяких функцій, що є в конкурентів, таких як музичний стрімінг

– Платна підписка для доступу до додаткових функцій

Проаналізувавши основні аналоги програмного продукту, ми змогли краще зрозуміти, які особливості слід врахувати при розробці веб-додатку, щоб забезпечити його конкурентоспроможність та відповідність потребам користувачів. Виходячи з цього, наведемо таблицю порівняння основних функцій та характеристик популярних платформ для прослуховування подкастів.

Таблиця 1.1.

Критерії / Додатки	Spotify	Apple Podcasts	Google Podcasts	Overcast
<i>Бібліотека</i>	Велика, включаючи музику	Велика, але обмежена новими випусками	Велика	Велика
<i>Дизайн</i>	Інтуїтивний, сучасний	Інтуїтивний, простий	Простий, мінімалістичний	Чистий, мінімалістичний
<i>Відтворення і управління</i>	Стандартне	Стандартне	Стандартне	Розширене з додатковими функціями
<i>Зручність використання</i>	Висока	Висока для користувачів Apple	Висока для користувачів Google	Висока
<i>Синхронізація між пристроями</i>	Синхронізується	Обмежена (тільки Apple-пристрої)	Обмежена	Синхронізується
<i>Рекомендації та відкриття нових подкастів</i>	Так	Обмежено	Так	Мінімальні
<i>Функції спільного користування</i>	Так	Немає	Немає	Немає
<i>Інтеграція з іншими сервісами</i>	Так (музика)	Так (Apple Music)	Так (Google сервіси)	Немає
<i>Можливості завантаження</i>	Так	Так	Так	Так
<i>Зменшення швидкості відтворення</i>	Так	Немає	Немає	Так

Можливості налаштування еквалайзера	Так	Немає	Немає	Так
-------------------------------------	-----	-------	-------	-----

Переглянувши таблицю 1.1 з порівнянням основних критеріїв популярних додатків для прослуховування подкастів, можна зробити кілька важливих висновків. Всі чотири додатки мають великі бібліотеки подкастів, однак Spotify виділяється також включенням музики, що робить його більш універсальним. Дизайн додатків варіюється від сучасного та інтуїтивного у Spotify до мінімалістичного у Overcast, зручного для користувачів, які цінують простоту. Що стосується відтворення та управління контентом, Overcast вигідно виділяється своїми розширеними функціями. Зручність використання висока у всіх додатків, проте синхронізація між пристроями повноцінно підтримується лише у Spotify та Overcast. Кажучи про рекомендаційні алгоритми, вони найкраще всього реалізовані у Spotify, тоді як, Google Podcasts, Apple Podcasts і Overcast мають обмежені можливості в цій сфері.

Для створення конкурентоспроможного веб-додатка слід врахувати сильні сторони кожного аналога: забезпечити широкий вибір контенту, зручний та інтуїтивно зрозумілий інтерфейс, а також розширені можливості управління відтворенням. Інтеграція з іншими сервісами та функції спільного користування також можуть додати цінності, роблячи додаток більш привабливим для широкої аудиторії. Важливо створити функціонал, що відповідає сучасним вимогам та очікуванням користувачів. Такий підхід дозволить створити продукт, який задовольнить потреби широкої аудиторії та забезпечить його конкурентоспроможність на ринку.

1.3. Визначення архітектури веб-додатку для прослуховування подкастів.

Веб-додаток буде використовувати клієнт-серверну архітектуру для забезпечення розділення відповідальності між клієнтською та серверною частинами, що сприяє більшій гнучкості та можливості масштабування. Для

клієнтської частини було обрано Next.js, який є фреймворком для React, що забезпечує серверний рендеринг і генерацію статичних сторінок. Це покращує продуктивність та SEO, що є важливим для веб-додатку. Також, це дозволить легко створювати динамічні сторінки та забезпечує високу швидкість завантаження контенту. TypeScript використовується як основна мова програмування. Завдяки можливостям статичної типізації TypeScript допомагає виявляти помилки на етапі розробки, що підвищує надійність коду. Для стилізації додатку було обрано утилітарний CSS-фреймворк Tailwind CSS. Серверна частина додатку була написана використовуючи середовище Node.js з бібліотекою vercel/postgres для роботи з базою даних PostgreSQL, яка ще відома ще під назвою Postgres.

Клієнт-серверна архітектура є моделлю, де обчислювальні завдання та ресурси розподіляються між серверами та клієнтами. Сервери відповідають за управління даними, обробку запитів та зберігання інформації, тоді як клієнти займаються відображенням даних та взаємодією з користувачами. Такий підхід дає можливість створювати гнучкі системи, де сервери можуть бути легко оновлені або замінені без впливу на клієнтську частину [6].

Next.js є популярним фреймворком для React, що забезпечує серверний рендеринг та генерацію статичних сайтів. Серверний рендеринг забезпечує швидке завантаження сторінок та покращує SEO, що є важливим для залучення та утримання користувачів. Next.js також дозволяє створювати динамічні сторінки, що є корисним для додатку з великим обсягом контенту, такого як подкасти. Крім того, він надає розширені можливості для налаштування та оптимізації продуктивності, що робить його ідеальним вибором для створення сучасних веб-додатків [7].

TypeScript був обраний як основна мова програмування завдяки статичній типізації. Це значно підвищує надійність коду. Також, він забезпечує кращу інтеграцію з редакторами коду та інструментами розробки, що сприяє більш ефективному процесу створення додатку [8].

Tailwind CSS дозволяє швидко створювати адаптивні інтерфейси без написання великої кількості кастомного CSS. Використовуючи утиліти Tailwind, розробники можуть зосередитися на функціональності, не витрачаючи багато часу на стилізацію. Tailwind також забезпечує високу продуктивність завдяки своїй малій вазі та ефективності. Це робить його відмінним вибором для проектів, де важлива швидкість розробки та унікальність дизайну [9].

Серверна частина додатку була написана використовуючи середовище Node.js вбудоване в Next.js, яке забезпечує високу продуктивність та масштабованість завдяки своїй неблокуючій архітектурі. Node.js, також, дозволяє легко обробляти велику кількість одночасних запитів, що є важливим для веб-додатків з високим навантаженням [10].

MySQL є популярною базою даних, яка має схожі можливості з Postgres, але Postgres надає кращу підтримку складних запитів та має більш надійний механізм транзакцій. SQLite є легкою базою даних, але вона більше підходить для мобільних додатків або невеликих проектів, а не для повноцінних веб-додатків з високим навантаженням. MongoDB, будучи NoSQL базою даних, може бути хорошою альтернативою для додатків, де структура даних є гнучкою та змінною, але для нашого проекту, де структура даних є більш стабільною та реляційною, Postgres є кращим вибором.

Більш того, для роботи з базою даних Postgres в Next.js, зручно використовувати бібліотеку vercel/postgres, яка забезпечує зручний інтерфейс для взаємодії з базою даних та оптимізована для роботи з великими обсягами даних [11].

PostgreSQL відома своєю надійністю та продуктивністю, що робить її відмінним вибором для зберігання та управління даними подкастів [12].

Вибір інструментів для розробки веб-додатку для прослуховування подкастів був зроблений з урахуванням необхідності забезпечення високої продуктивності, зручності розробки, підтримки в довгостроковій перспективі та надійності. Кожна

з цих технологій має свої переваги і відповідає вимогам нашого проекту, дозволяючи створити конкурентоспроможний продукт, який задовольнить потреби користувачів.

Висновок до першого розділу

У першому розділі було проведено аналіз проблематики і рішень для створення веб-додатка для прослуховування подкастів. Мета даної роботи полягає в розробці інтуїтивно зрозумілого, продуктивного та безпечного веб-додатка, який задовольнить сучасні потреби користувачів у прослуховуванні та управлінні подкастами.

Визначення вимог та функціоналу додатка базується на глибокому аналізі існуючих аналогів, таких як Spotify, Apple Podcasts, Google Podcasts та Overcast. Цей аналіз дозволив виявити ключові функції та переваги, які мають бути реалізовані, щоб додаток став конкурентоспроможним.

З точки зору технологій, вибір Next.js – як для клієнтської, так і для серверної частини додатка, TypeScript для написання коду та Tailwind CSS для стилізації забезпечує високу продуктивність, адаптивність та зручність розробки. Використання Next.js разом з Vercel Postgres для бази даних гарантує надійність і масштабованість, необхідні для обробки великих обсягів даних та високого навантаження.

Таким чином, аналіз проблематики та методів реалізації веб-додатка для прослуховування подкастів показав, що ключем до успішної розробки є поєднання передових технологій з детальним розумінням потреб користувачів та тенденцій ринку. Це дозволить створити якісний продукт, який буде відповідати вимогам сучасного користувача та забезпечить йому максимальний комфорт при прослуховуванні подкастів.

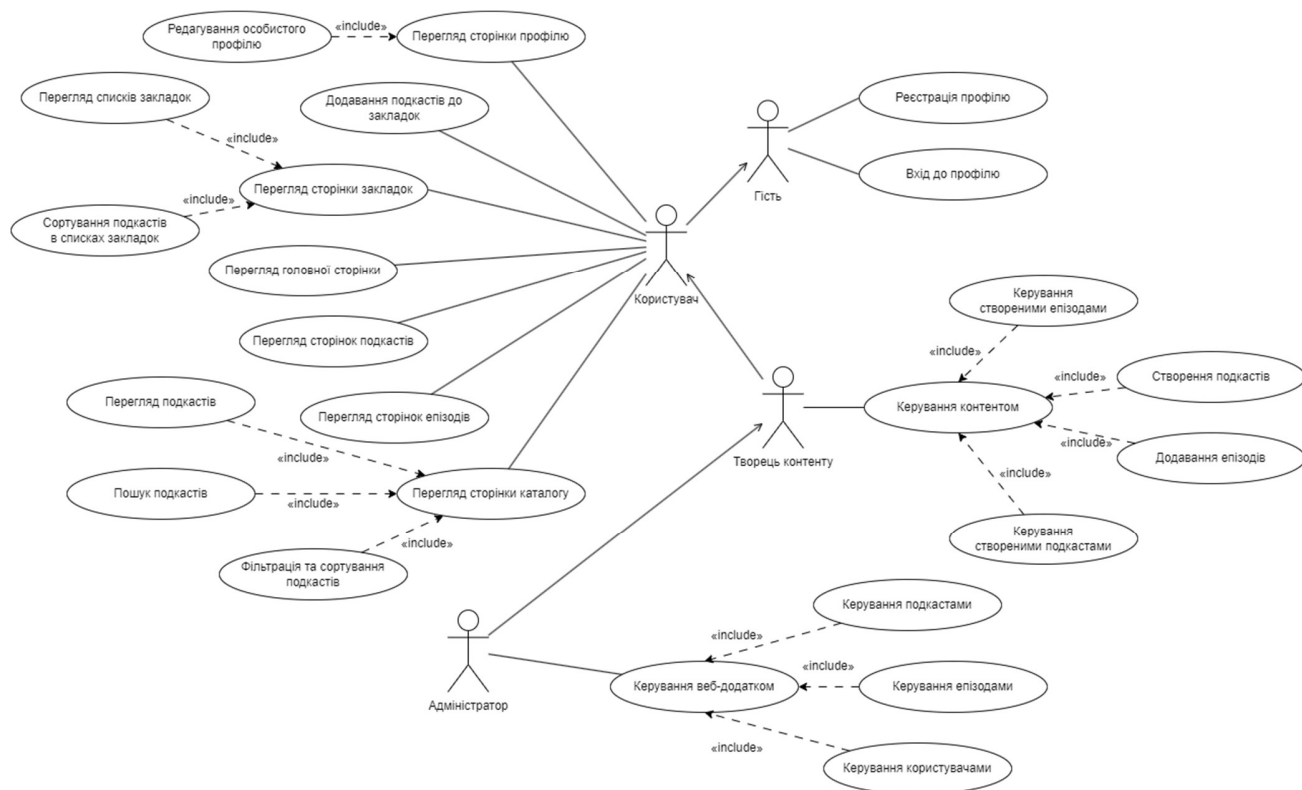
РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-ДОДАТКУ ДЛЯ ПРОСЛУХОВУВАННЯ ПОДКАСТІВ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Веб-додаток для прослуховування подкастів має інтуїтивно зрозумілий інтерфейс, який дозволяє користувачам легко переглядати різні сторінки, шукати подкасти, сортувати їх за різними параметрами, застосовувати фільтри і слухати обраний контент. Інтерфейс розроблений таким чином, щоб забезпечити максимальну зручність і простоту використання для всіх типів користувачів, незалежно від типів їх пристроїв. Система підтримує три основні типи користувачів: звичайних користувачів, творців контенту та адміністраторів. Звичайні користувачі мають доступ до базового функціоналу, що включає можливість переглядати сторінки подкастів, здійснювати пошук, сортувати та фільтрувати результати за різними критеріями та, звичайно, прослуховувати обрані подкасти. Творці контенту, окрім базових функцій, мають додаткові можливості. Вони можуть додавати нові подкасти та епізоди. Це дає можливість авторам легко поширювати свій контент серед слухачів, регулярно оновлювати свій канал новими випусками та тримати свою аудиторію в курсі останніх подій. Процес додавання нового контенту є інтуїтивно зрозумілим і не вимагає спеціальних технічних знань, що сприяє залученню великої кількості творців. Адміністратори мають найширший спектр можливостей, оскільки їм доступна панель адміністратора. Ця панель дозволяє редагувати подкасти та епізоди, а також управляти користувачами. Для того щоб краще зрозуміти, як взаємодіють елементи додатку, було розроблено діаграму варіантів використання (рис. 2.1).

Діаграма варіантів використання дозволяє узагальнити взаємодії користувачів, також відомих як актори, із системою, використовуючи спеціальні символи та з'єднувачі. Ця діаграма допомагає візуально представляти різні сценарії, в яких система або додаток взаємодіє з людьми, організаціями чи зовнішніми

Уніфікована мова моделювання (UML) – це стандартна мова візуального моделювання, що була впроваджена Групою з управління об'єктами (OMG) у 1997 році для створення та розробки комп'ютерних додатків. UML забезпечує стабільну та загальноприйнятну нотацію для моделювання, що дозволяє ІТ-фахівцям читати та поширювати плани структури і дизайну систем, подібно до того, як будівельники користуються кресленнями будівель [14].



Опис діаграми варіантів використання наведено в табл. 2.1 – 2.7.

Табл. 2.1.

Опис прецеденту «Автентифікація»

Назва варіанту використання	Автентифікація
Основні актори	Гість
Короткий опис	Щоб авторизуватися в системі, користувач повинен увійти в свій акаунт, або створити новий.

Табл. 2.2.

Опис прецеденту «Перегляд подкастів»

Назва варіанту використання	Перегляд подкастів
Основні актори	Користувач
Короткий опис	Користувач може переглядати список доступних подкастів.

Табл. 2.3.

Опис прецеденту «Пошук, фільтрування та сортування подкастів»

Назва варіанту використання	Пошук, фільтрування та сортування подкастів
Основні актори	Користувач
Короткий опис	Користувач може здійснювати пошук, фільтрування та сортування подкастів за різними критеріями.

Табл. 2.4.

Опис прецеденту «Перегляд профілю»

Назва варіанту використання	Перегляд профілю
Основні актори	Користувач

Короткий опис	Користувач може переглядати або редагувати свою особисту інформацію.
---------------	--

Табл. 2.5.

Опис прецеденту «Управління закладками»

Назва варіанту використання	Управління закладками
Основні актори	Користувач
Короткий опис	Користувач має можливість додати, переглядати та сортувати подкасти у списках закладок.

Табл. 2.6.

Опис прецеденту «Створення подкастів та епізодів»

Назва варіанту використання	Створення подкастів та епізодів
Основні актори	Творець контенту
Короткий опис	Творець контенту може створювати нові подкасти та додавати в них епізоди. Також він має можливість керувати подкастами та епізодами які створив.

Табл. 2.7.

Опис прецеденту «Управління контентом»

Назва варіанту використання	Управління контентом
Основні актори	Адміністратор
Короткий опис	Адміністратор може керувати користувачами, подкастами та епізодами.

Основна ціль розробки веб-додатку для прослуховування подкастів полягає у створенні зручної та доступної платформи, яка дозволяє користувачам легко знаходити, організовувати та слухати улюблені подкасти. Цей додаток прагне забезпечити безперервний досвід для слухачів, пропонуючи інтуїтивно зрозумілий інтерфейс, можливість керування списками закладок, а також пошук і сортування подкастів за різними критеріями. Окрім цього, платформа надає подкастерам ефективні інструменти для створення та редагування своїх подкастів і епізодів. Адміністратори, у свою чергу, мають змогу управляти контентом і користувачами, забезпечуючи належне функціонування та безпеку системи. Додаток спрямований на об'єднання слухачів і творців контенту, забезпечуючи якісний та інтерактивний досвід прослуховування.

Основна мета розробки веб-додатку для прослуховування подкастів полягає в створенні унікального користувацького досвіду, що поєднує зручність, доступність і якість контенту, сприяючи розвитку культури аудіоконтенту.

Вимоги користувачів:

Внутрішній користувач – гостей:

1. Автентифікація та авторизація для доступу до функціоналу веб-додатку.

Внутрішній користувач – користувач:

1. Перегляд подкастів;
2. Перегляд епізодів;
3. Прослуховування епізодів;
4. Пошук подкастів;
5. Сортування подкастів;
6. Фільтрація подкастів;
7. Перегляд профілю;
8. Редагування профілю;
9. Перегляд закладок;
10. Додавання подкасту до списків закладок.

Внутрішній користувач – творець контенту:

1. Створення нового подкасту;
2. Редагування створеного подкасту;
3. Додавання епізодів;
4. Керування епізодами.

Внутрішній користувач – Адміністратор:

1. Доступ до панелі адміністратора;
2. Керування подкастами;
3. Керування епізодами;
4. Керування користувачами.

Функціональні вимоги:

- Реєстрація користувачів: користувачі повинні мати можливість реєстрації акаунту.
- Авторизація користувачів: користувачі повинні мати можливість авторизації зареєстрованих користувачів за допомогою електронної пошти та пароля.
- Перегляд подкастів: Користувачі повинні мати можливість переглядати список доступних подкастів.
- Перегляд епізодів: Користувачі повинні мати можливість переглядати список епізодів вибраного подкасту.
- Прослуховування епізодів: Користувачі повинні мати можливість прослуховувати епізоди подкастів.
- Пошук подкастів: Користувачі повинні мати можливість шукати подкасти за ключовими словами або іменами авторів.
- Сортування подкастів: Користувачі повинні мати можливість сортувати списки подкастів.
- Фільтрація подкастів: Користувачі повинні мати можливість фільтрувати списки подкастів по параметрам.

- Перегляд профілю: Користувачі повинні мати можливість переглядати свій профіль та профіль інших користувачів.
- Управління профілем: Користувачі повинні мати можливість редагувати свої профілі.
- Управління закладками: Користувачі повинні мати можливість додавати подкасти до закладок для швидкого доступу та мати можливість переглядати, редагувати та видаляти свої закладки.
- Створення та управління подкастами: Творці контенту повинні мати можливість створювати та редагувати подкасти.
- Додавання та управління епізодами: Творці контенту повинні мати можливість створювати, редагувати та видаляти епізоди.
- Управління користувачами: Адміністратори повинні мати можливість переглядати та редагувати користувачів.
- Управління подкастами: Адміністратори повинні мати можливість переглядати, редагувати та видаляти подкасти.
- Управління епізодами: Адміністратори повинні мати можливість переглядати, редагувати та видаляти епізоди.

Нефункціональні вимоги:

- Продуктивність
 - Додаток повинен підтримувати одночасне обслуговування великої кількості користувачів без зниження продуктивності.
 - Додаток повинен забезпечувати час відгуку не більше 2 секунд для основних операцій.
- Сумісність
 - Додаток повинен підтримувати основні веб-браузери.
 - Додаток повинен працювати на основних операційних системах.
- Зручність використання

- Інтерфейс додатку повинен бути інтуїтивно зрозумілим, з логічною навігацією та доступом до основних функцій.
- Додаток повинен коректно відображатися та працювати на різних пристроях і розмірах екранів.
- Безпека
 - Додаток повинен використовувати безпечні методи аутентифікації та авторизації для захисту облікових записів користувачів.
 - Всі дані користувачів, включаючи особисту інформацію та паролі, повинні бути захищені за допомогою шифрування.

Системні вимоги:

Вимоги до середовища виконання:

- Процесор: 1.6 ГГц або швидше (рекомендовано 2.0 ГГц або швидше)
- Оперативна пам'ять: мінімум 2 ГБ (рекомендовано 4 ГБ або більше)
- Наявність стабільного підключення до інтернету

Вимоги до сервера інформаційної системи:

- У системі повинна бути встановлена реляційна база даних PostgreSQL, яка забезпечує надійне зберігання та ефективну організацію даних.
- Система повинна підтримувати технології node.js та next.js для забезпечення швидких та надійних оброблень запитів.

Після аналізу функціональних вимог було виділено сутності, які допоможуть забезпечити реалізацію програмної частини веб-додатку. Ці сутності відображені на діаграмі класів веб-додатку, яка представлена на рис. 2.2.

Діаграма класів – це статична структурна діаграма, що відображає структуру системи через представлення класів системи, їх атрибутів, операцій (методів) та взаємозв'язків між об'єктами [15].

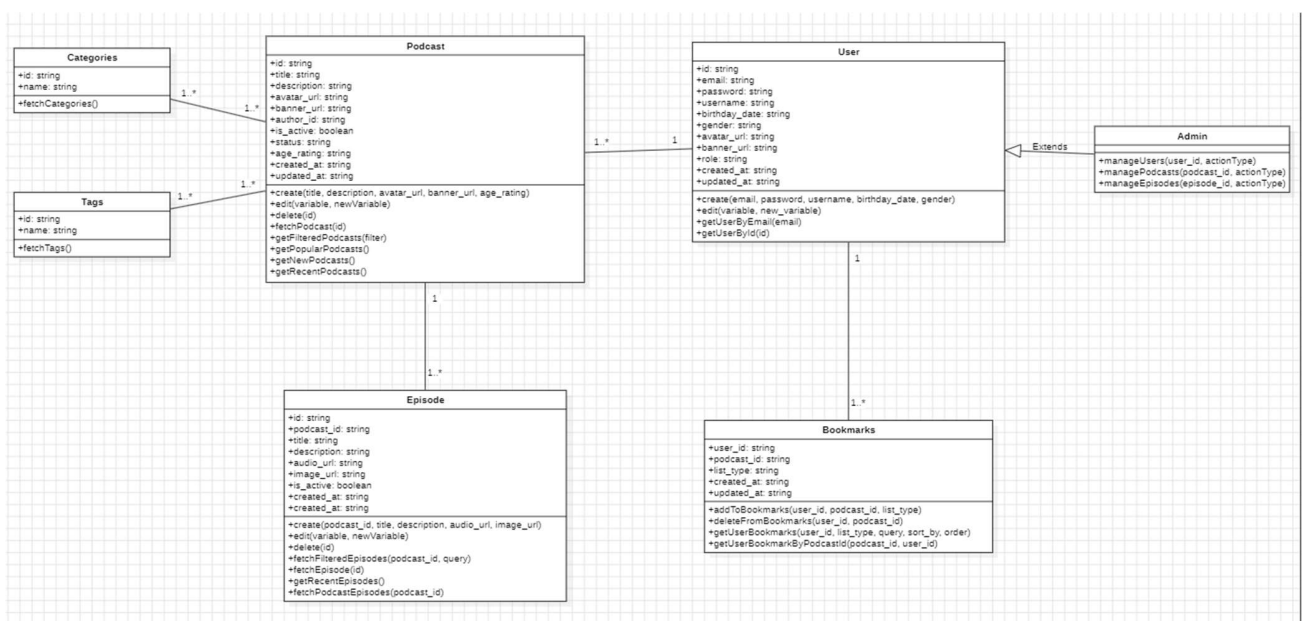


Рис. 2.2. Діаграма класів

User – клас для представлення користувача, який відповідає за зберігання та управління всією персональною інформацією.

Admin – клас, який наслідується від User, представляє користувача з правами адміністратора з розширеними можливостями управління користувачами та контентом.

Podcast – клас для представлення подкасту, який відповідає за зберігання та управління подкастами.

Episode – клас для представлення епізоду подкасту, який відповідає за зберігання та управління епізодами.

Categories – клас для представлення категорій, який відповідає за управління категоріями подкастів.

Tags – клас для представлення тегів, який відповідає за управління тегами подкастів.

Bookmarks – клас для представлення закладок користувачів, який відповідає за управління списками закладок користувачів.

2.2. Розробка бази даних системи

Процес розробки бази даних системи розпочинається з визначення основних вимог до зберігання та обробки даних, які повинна забезпечити система. На основі аналізу функціональних вимог та визначення сутностей, необхідних для реалізації додатка, формується структура бази даних. База даних повинна забезпечувати надійне зберігання даних, а також швидкий та ефективний доступ до них. Для цього використовується реляційна база даних PostgreSQL, яка відповідає усім вимогам до зберігання даних та забезпечує високу продуктивність і надійність.

База даних системи складається з восьми таблиць, кожна з яких виконує окрему функцію і має свої унікальні поля та зв'язки з іншими таблицями:

1. Таблиця Users для збереження користувачів
2. Таблиця Podcasts для збереження подкастів
3. Таблиця Episodes для збереження епізодів
4. Таблиця Categories для збереження категорій подкасту
5. Таблиця Tags для збереження тегів подкасту
6. Таблиця Bookmarks для збереження закладок користувачів
7. Таблиця PodcastCategories для реалізації зв'язку «багато-до-багатьох» між подкастами та категоріями.
8. Таблиця PodcastTags для реалізації зв'язку «багато-до-багатьох» між подкастами та тегами.

Опис структури кожної з таблиць бази даних додатку для прослуховування подкастів наведений у таблицях 2.8. – 2.15.

Табл. 2.8.

Опис структури таблиці «Users»

Назва	Тип даних	ПК	ЗК	Опис поля
id	UUID	Так	Ні	Унікальний ідентифікатор користувача
email	TEXT	Ні	Ні	Електронна адреса
password	TEXT	Ні	Ні	Пароль користувача
username	TEXT	Ні	Ні	Ім'я користувача
birthday_date	TEXT	Ні	Ні	Дата народження користувача
gender	VARCHAR(25)	Ні	Ні	Стать користувача
avatar_url	TEXT	Ні	Ні	Назва файлу аватару користувача
banner_url	TEXT	Ні	Ні	Назва файлу банеру користувача
role	VARCHAR(20)	Ні	Ні	Роль користувача
created_at	TIMESTAMPTZ	Ні	Ні	Дата створення запису
updated_at	TIMESTAMPTZ	Ні	Ні	Дата останнього оновлення запису

Табл. 2.9.

Опис структури таблиці «Podcasts»

Назва	Тип даних	ПК	ЗК	Опис поля
id	UUID	Так	Ні	Унікальний ідентифікатор подкасту
title	VARCHAR(100)	Ні	Ні	Назва подкасту
description	TEXT	Ні	Ні	Опис подкасту
avatar_url	TEXT	Ні	Ні	Назва файлу обкладинки подкасту
banner_url	TEXT	Ні	Ні	Назва файлу банеру подкасту
author_id	UUID	Ні	Так	Ідентифікатор автора
is_active	BOOLEAN	Ні	Ні	Статус публічного доступу подкасту
status	VARCHAR(20)	Ні	Ні	Статус подкасту
age_rating	VARCHAR(20)	Ні	Ні	Віковий рейтинг
created_at	TIMESTAMPTZ	Ні	Ні	Дата створення запису
updated_at	TIMESTAMPTZ	Ні	Ні	Дата останнього оновлення запису

Табл. 2.10.

Опис структури таблиці «Episodes»

<i>Назва</i>	<i>Тип даних</i>	<i>ПК</i>	<i>ЗК</i>	<i>Опис поля</i>
id	UUID	Так	Ні	Унікальний ідентифікатор епізоду
podcast_id	UUID	Ні	Так	Ідентифікатор подкасту
title	VARCHAR(100)	Ні	Ні	Назва епізоду
description	TEXT	Ні	Ні	Опис епізоду
image_url	TEXT	Ні	Ні	Назва файлу обкладинки епізоду
audio_url	TEXT	Ні	Ні	Назва аудіофайлу епізоду
is_active	BOOLEAN	Ні	Ні	Статус публічного доступу епізоду
created_at	TIMESTAMPTZ	Ні	Ні	Дата створення запису
updated_at	TIMESTAMPTZ	Ні	Ні	Дата останнього оновлення запису

Табл. 2.11.

Опис структури таблиці «Bookmarks»

<i>Назва</i>	<i>Тип даних</i>	<i>ПК</i>	<i>ЗК</i>	<i>Опис поля</i>
user_id	UUID	Так	Так	Унікальний ідентифікатор тегу
podcast_id	UUID	Так	Так	Назва тегу
list_type	VARCHAR(50)	Ні	Ні	Тип списку
created_at	TIMESTAMPTZ	Ні	Ні	Дата створення запису
updated_at	TIMESTAMPTZ	Ні	Ні	Дата останнього оновлення запису

Табл. 2.12.

Опис структури таблиці «Categories»

<i>Назва</i>	<i>Тип даних</i>	<i>ПК</i>	<i>ЗК</i>	<i>Опис поля</i>
id	UUID	Так	Ні	Унікальний ідентифікатор категорії
name	VARCHAR(50)	Ні	Ні	Назва категорії

Табл. 2.13.

Опис структури таблиці «Tags»

<i>Назва</i>	<i>Тип даних</i>	<i>ПК</i>	<i>ЗК</i>	<i>Опис поля</i>
id	UUID	Так	Ні	Унікальний ідентифікатор тегу
name	VARCHAR(50)	Ні	Ні	Назва тегу

Табл. 2.14.

Опис структури таблиці «PodcastCategories»

<i>Назва</i>	<i>Тип даних</i>	<i>ПК</i>	<i>ЗК</i>	<i>Опис поля</i>
podcast_id	UUID	Так	Так	Ідентифікатор подкасту
category_id	UUID	Так	Так	Ідентифікатор категорії

Табл. 2.15.

Опис структури таблиці «PodcastTags»

<i>Назва</i>	<i>Тип даних</i>	<i>ПК</i>	<i>ЗК</i>	<i>Опис поля</i>
podcast_id	UUID	Так	Так	Ідентифікатор подкасту
tag_id	UUID	Так	Так	Ідентифікатор тегу

2.3. Проектування та реалізація алгоритмів роботи системи

Ефективне функціонування системи для прослуховування подкастів залежить від ретельно спроектованих та реалізованих алгоритмів, які забезпечують її основні можливості. Ці алгоритми визначають, як система обробляє дані, взаємодіє з користувачами та виконує необхідні операції. Проектування алгоритмів є критичним етапом у розробці програмного забезпечення, оскільки саме вони забезпечують виконання логіки та забезпечують надійність і продуктивність системи.

Для подальшого уточнення і візуалізації процесів, що відбуваються в системі, було створено діаграми активностей. Вони є потужним інструментом, який допомагає моделювати бізнес-процеси і детально описувати алгоритми роботи

системи. Ці діаграми дозволяють зрозуміти, як відбувається виконання різних дій в системі, які етапи проходять користувачі, і як система обробляє різні запити та операції.

Розглянемо першу діаграму активностей, що описує процес реєстрації користувача (рис. 2.3). Цей процес є ключовим для забезпечення доступу нових користувачів до функціоналу системи. Діаграма реєстрації демонструє послідовність кроків, які необхідно пройти новому користувачу для створення облікового запису.

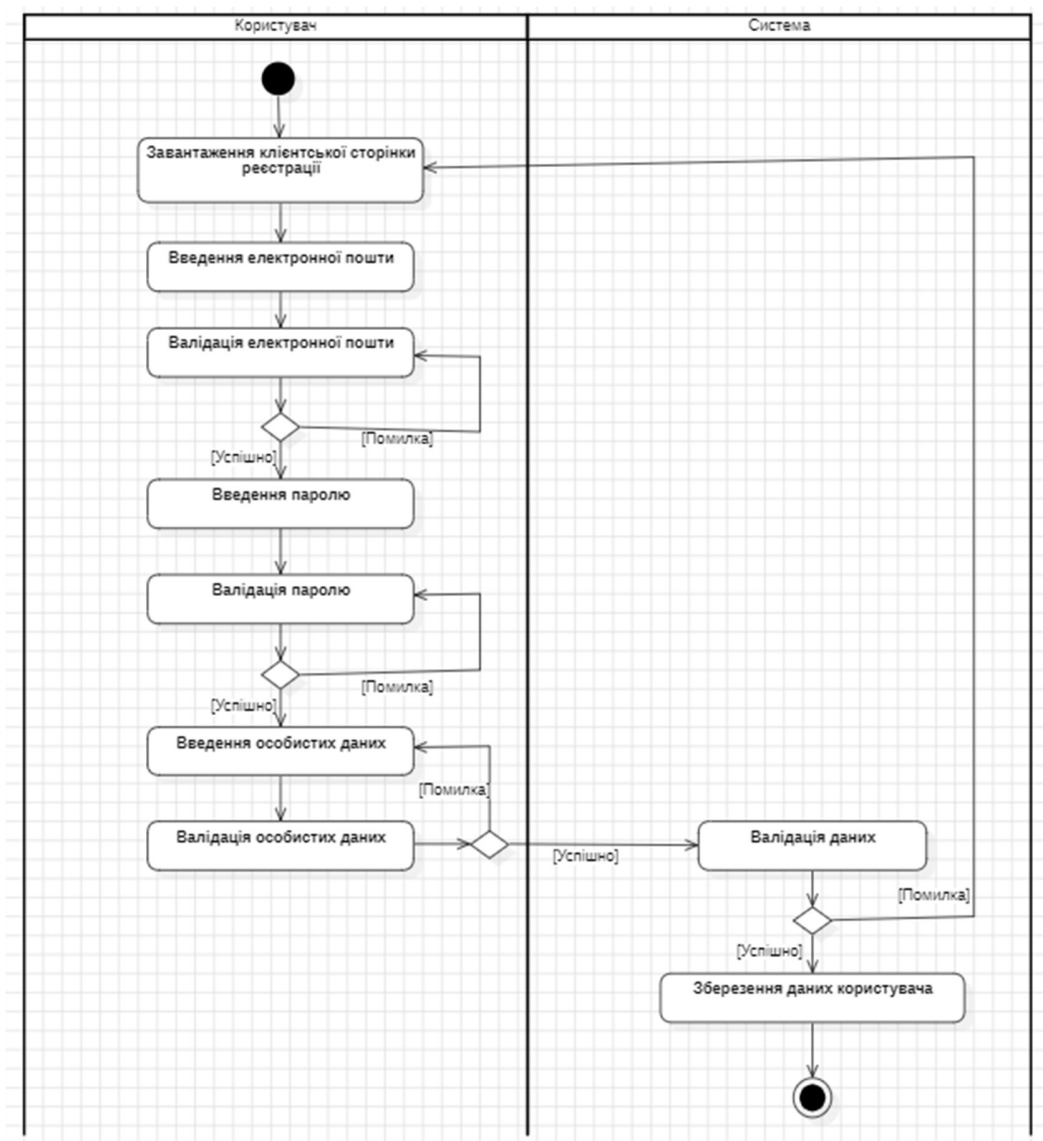


Рис. 2.3. Діаграма активності реєстрації користувача

Процес реєстрації починається з того, що користувач завантажує сторінку реєстрації, після чого він вводить свою електронну пошту, пароль та особисті дані. На кожному етапі введені дані проходять валідацію на стороні клієнта. Після того, як користувач відправив форму, поля проходять валідацію на сервері. Якщо всі дані вірні, система зберігає дані користувача в базі. У випадку помилки користувач повертається до відповідного етапу для виправлення помилок.

Для забезпечення повного розуміння процесів, які відбуваються в системі, було також створено діаграми активностей для інших важливих функцій додатка. Однією з таких функцій є процес додавання подкасту (рис. 2.4).

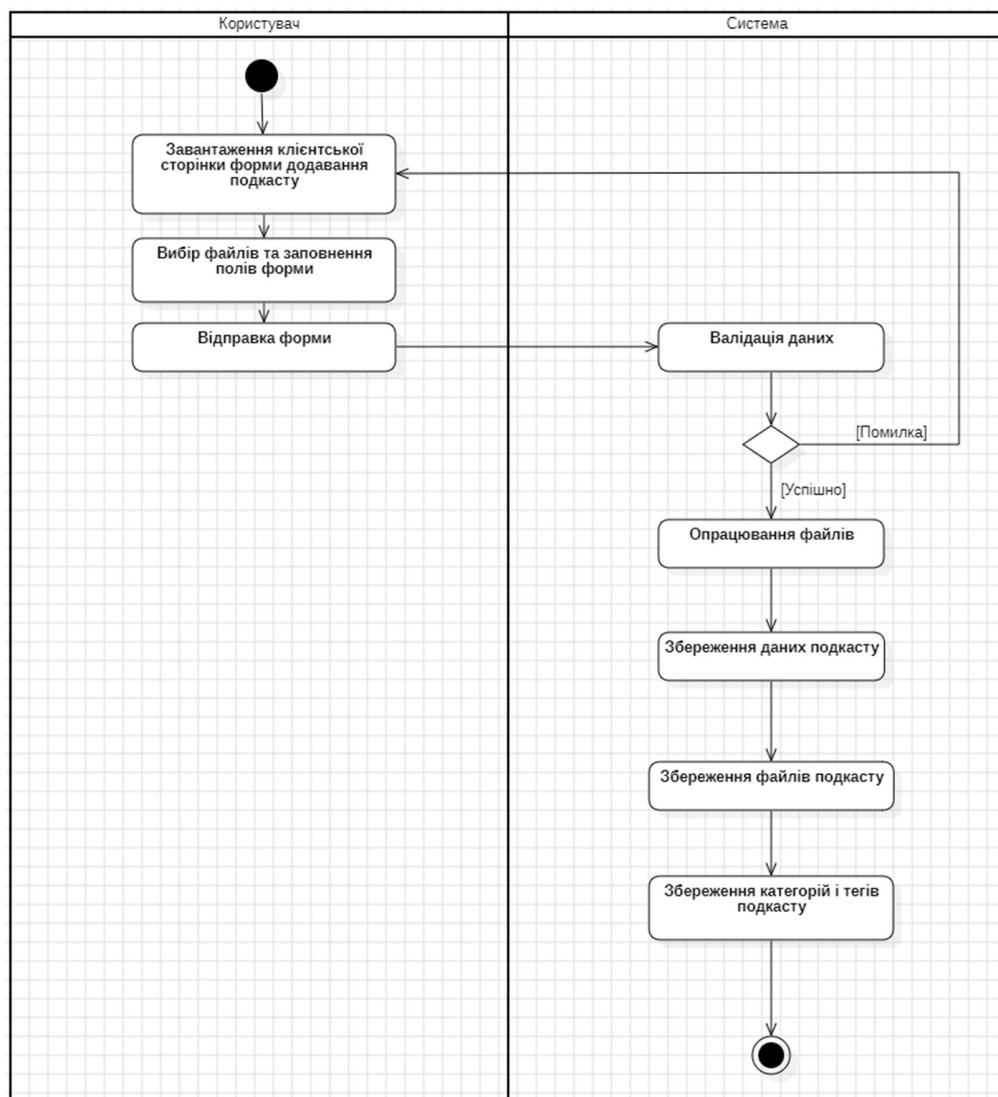


Рис. 2.4. Діаграма активності створення подкасту

Діаграма активностей для процесу додавання подкасту ілюструє кроки, які виконує користувач для завантаження нового подкасту. Створювати подкаст може лише користувач з рівнем доступу «Творець Контенту» або «Адміністратор». Процес починається з завантаження клієнтської сторінки форми додавання подкасту. Користувач вибирає файли та заповнює необхідні поля форми, після чого відправляє форму. На стороні системи дані, що надійшли, проходять валідацію. У разі успішної валідації система обробляє файли, зберігає дані подкасту, файли подкасту, а також категорії і теги, пов'язані з подкастом. Якщо під час валідації виявляються помилки, користувач повертається до відповідного етапу для їх виправлення. Ця діаграма чітко демонструє послідовність дій та взаємодію між користувачем і системою під час додавання нового подкасту.

Наступною діаграмою активностей є процес редагування епізоду подкасту (рис. 2.5). Ця діаграма відображає етапи, які проходить користувач під час внесення змін до існуючого епізоду. Слід зауважити, вносити зміни до епізоду можуть лише користувачі з правами «Адміністратор» або «Творець контенту», який створив подкаст, до якого належить епізод.

Процес починається з завантаження клієнтської сторінки налаштування епізоду, де користувач може змінити поля або файли, пов'язані з епізодом. Після внесення необхідних змін користувач відправляє форму. Система перевіряє валідність ID епізоду та виконує валідацію введених даних і файлів. Якщо перевірка пройшла успішно, система оновлює дані епізоду. У разі виникнення помилок на будь-якому з етапів валідації, користувач повертається до сторінки редагування епізоду для їх виправлення. Ця діаграма демонструє, як користувач взаємодіє з системою для оновлення інформації про епізод, забезпечуючи точність та цілісність даних.

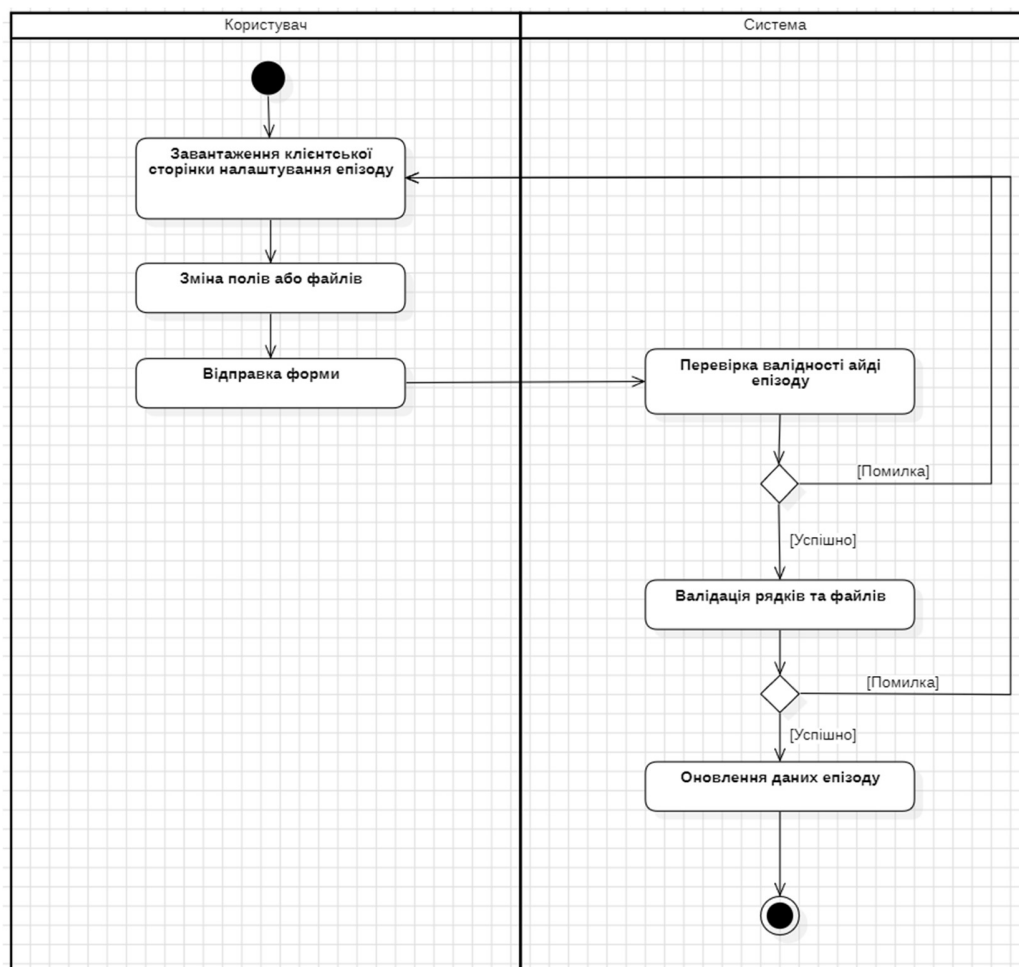


Рис. 2.5. Діаграма активності редагування епізоду

Діаграма активності для процесу переходу на сторінку панелі адміністратора детально описує кроки, які виконує користувач під час доступу до адміністративної панелі (Рис. 2.6).

Процес починається з переходу користувачем на посилання або після натискання на кнопку для переходу до панелі адміністратора. Система одразу перевіряє, чи має користувач необхідні права доступу для входу. Цей крок є критичним для забезпечення безпеки системи. Якщо доступ дозволено, користувач переходить до наступного кроку. Якщо ж доступ заборонено, система виводить повідомлення про помилку, і процес завершується для цього користувача.

Після успішної перевірки прав доступу система завантажує інтерфейс адміністративної панелі, що надає користувачу доступ до функцій управління

системою. У панелі адміністратора користувач має можливість керувати основними функціями системи: управлінням користувачами, подкастами та епізодами. Це включає управління правами користувачів, редагування та видалення користувачів, додавання нових подкастів, редагування та видалення існуючих подкастів, а також управління епізодами подкастів, включаючи редагування та видалення епізодів.

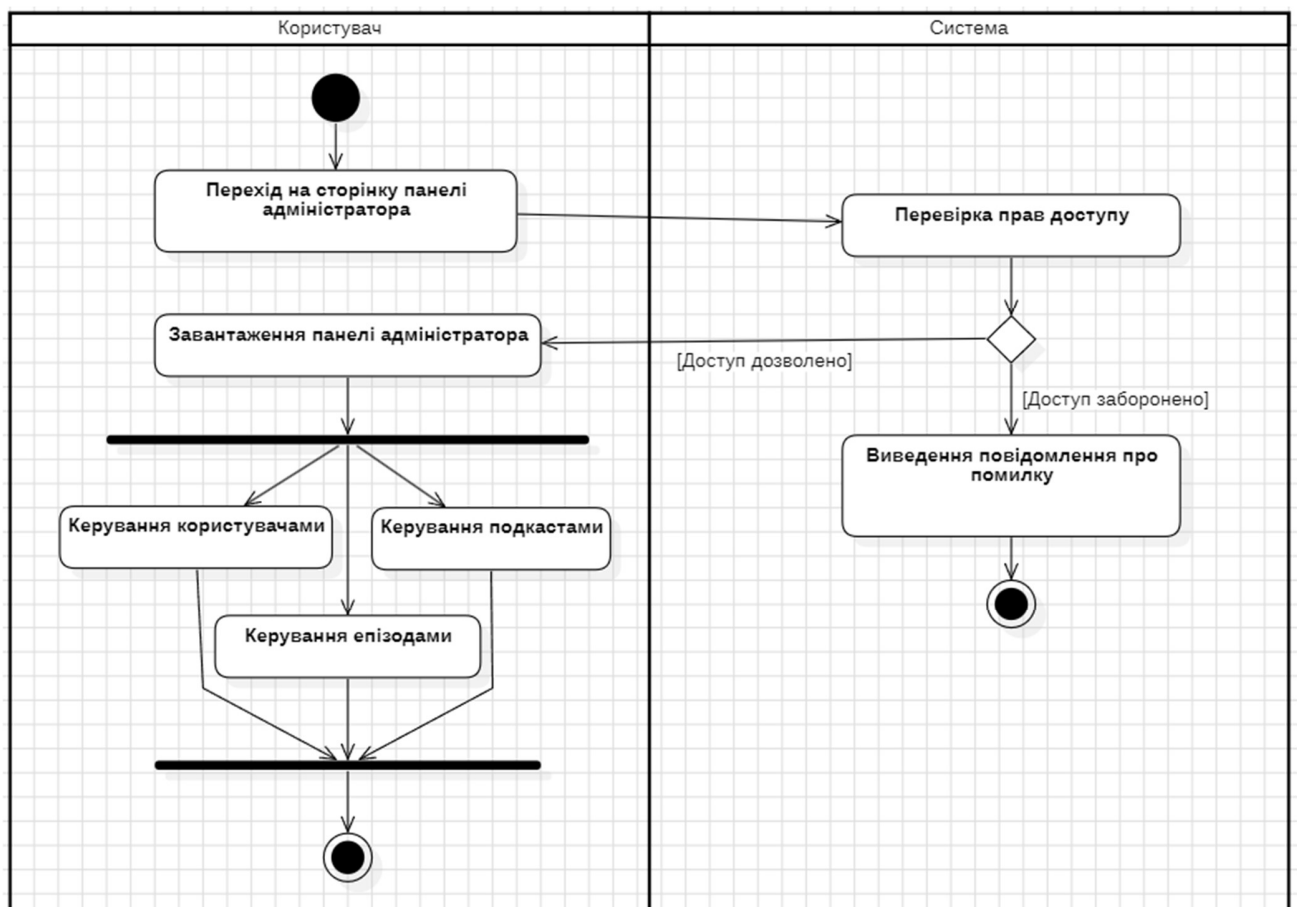


Рис. 2.6 Діаграма активності панелі адміністратора

2.4. Реалізація функціоналу веб-додатку для прослуховування подкастів

В основі додатку лежить Next.js, який використовує маршрутизацію файлової системи. Це означає, що структура папок автоматично визначає маршрути в додатку: кожна папка відповідає сегменту маршруту, який відображається у URL-

адресі. Такий підхід забезпечує логічну та зручну структуру проекту, де кожна сторінка і компонент легко знаходяться та підтримуються [16].

Архітектура додатку для прослуховування подкастів розділена на клієнтську та серверну частини. Основні серверні файли розміщені в папці /app/lib, щоб централізувати всі ключові функції.

Ключовою функцією додатку для прослуховування подкастів є збереження даних користувача у сесії після авторизації. Це необхідно для того, щоб користувач міг мати доступ до всіх можливостей додатку. Після успішної авторизації, основні дані користувача зашифровуються за допомогою токена JWT (JSON Web Token). JWT – це стандарт, який використовується для безпечної передачі інформації у вигляді зашифрованого JSON-об'єкта між сторонами [17].

Спершу, слід розглянути функцію створення сесії користувача:

```
export async function createSession({ userId, role, name, avatar_url }: SessionPayload) {
  const session = await encrypt({ userId, role, name, avatar_url });
  const expires = new Date(new Date(Date.now() + 7 * 24 * 60 * 60 * 1000));
  cookies().set("session", session, {
    httpOnly: true,
    secure: true,
    expires: expires,
    sameSite: "lax",
    path: "/",
  });
}
```

Ця функція здійснює шифрування основних даних користувача, а саме ідентифікатор користувача, роль, ім'я та назву аватару користувача, та створює сесію користувача, яка зберігається у вигляді HTTP-only куки, шифруючи її дані за допомогою функції encrypt. Вона створює новий токен JWT, який включає дані користувача та встановлює термін дії на один тиждень.

Функція encrypt є основною для шифрування даних користувача, щоб забезпечити їх захист під час зберігання та передачі. Спочатку функція створює новий об'єкт SignJWT, який отримує дані користувача у вигляді параметра. Далі встановлюється захищений заголовок з алгоритмом шифрування HS256, час створення токена та його термін дії на 7 днів. Нарешті, токен підписується за

допомогою закодованого ключа, який зберігається на сервері. Це гарантує, що токен не можна підробити, оскільки для його підпису використовується секретний ключ.

```
export async function encrypt(payload: SessionPayload | JWTPayload | undefined)
{
    return new SignJWT(payload)
        .setProtectedHeader({ alg: "HS256" })
        .setIssuedAt()
        .setExpirationTime("7d")
        .sign(encodedKey);
}
```

Після створення сесії важливо мати можливість її отримання. Для цього використовується функція `getSession`, яка зчитує токен сесії з куки і, якщо він існує, розшифровує його за допомогою функції `decrypt`, повертаючи дані користувача.

```
export async function getSession() {
    const session = cookies().get("session")?.value;
    if (!session) return null;
    return (await decrypt(session)) as SessionPayload;
}
```

Функція `getSession` є критичною для роботи з сесіями користувачів. Вона зчитує токен з куки, і якщо токен не знайдено, функція повертає `null`, вказуючи на відсутність активної сесії. У разі успішного зчитування токена, функція `getSession` викликає `decrypt` для його розшифровки та повертає розшифровані дані у вигляді об'єкта `SessionPayload`.

Функція `decrypt` відповідає за розшифровку токена сесії. Вона перевіряє та розшифровує токен, повертаючи збережені дані користувача. У разі невдалої перевірки, функція виводить повідомлення про помилку.

```
export async function decrypt(session: string | undefined = "") {
    try {
        const { payload } = await jwtVerify(session, encodedKey, {
            algorithms: ["HS256"],
        });
        return payload;
    } catch (error) {
        console.log("Failed to verify session");
    }
}
```

Функція `decrypt` є важливою частиною безпеки додатку, оскільки вона гарантує, що токен не був змінений або підроблений. Вона приймає токен сесії у вигляді рядка та намагається його розшифрувати за допомогою методу `jwtVerify`, використовуючи закодований ключ та алгоритм `HS256`. Якщо розшифровка проходить успішно, функція повертає `payload`, тобто дані користувача. У разі помилки, наприклад, якщо токен недійсний або закінчився термін його дії, функція виводить повідомлення про невдалу спробу розшифровки та повертає `undefined`.

Щоб забезпечити актуальність даних користувача та подовжити термін дії сесії, використовується функція `editSession`. Вона оновлює токен сесії, шифруючи нові дані користувача та зберігаючи їх у куки.

```
export async function editSession({ userId, role, name, avatar_url }: SessionPayload) {
  const payload = await encrypt({ userId, role, name, avatar_url });
  const expires = new Date(Date.now() + 7 * 24 * 60 * 60 * 1000);
  const res = NextResponse.next();
  res.cookies.set("session", payload, {
    httpOnly: true,
    secure: true,
    path: "/",
    sameSite: "lax",
    expires: expires,
  });
  return res;
}
```

Функція `editSession` призначена для оновлення сесії користувача перед кожним запитом, забезпечуючи актуальність та безпеку даних. Вона отримує нові дані користувача, шифрує їх за допомогою функції `encrypt` та створює новий токен сесії. Потім токен зберігається у куки з новим терміном дії на один тиждень.

Варто звернути увагу на деякі важливі особливості `Next.js`, наприклад, підтримка часткового рендерингу сторінок. Частковий рендеринг означає, що тільки сегменти маршруту, які змінюються під час навігації, відображаються знову на клієнті, а будь-які спільні сегменти зберігаються. Наприклад, при навігації між двома дочірніми маршрутами, `/p/catalog` та `/p/bookmarks`, сторінки `catalog` і `bookmarks` будуть оновлюватись, а спільний макет `p` залишиться без змін [18].

З виходом Next.js 14 була впроваджена нова функція під назвою частковий попередній рендеринг (partial pre-rendering). Вона дозволяє розробникам контролювати, яка частина сторінки буде попередньо відображена або відображена першою. Ця функція використовує API React Suspense, тому її легко зрозуміти і використовувати, якщо до цього були знайомі з React [19].

Частковий попередній рендеринг використовує комбінацію статичної обробки, зокрема поступового статичного відтворення і повної обробки на стороні сервера. За допомогою ISR, Next.js дозволяє попередньо оновлювати сторінки, що містять динамічні дані, під час збірки. Згодом ці сторінки поступово перерендерюються у фоновому режимі за потреби.

Також важливо зазначити, що Next.js підтримує як статичні, так і динамічні сторінки. Статичні сторінки створюються під час збірки і зберігаються у вигляді HTML-файлів, що дозволяє їх швидко завантажувати з сервера. Динамічні сторінки, навпаки, рендеряться на стороні сервера при кожному запиті, що дозволяє відображати актуальні дані у режимі реального часу [20].

Next.js надає кілька важливих файлів, які використовуються для створення загального макету додатку та обробки помилок.

- Page.tsx – є основою для кожної сторінки у Next.js. Він може містити будь-який контент, який необхідний для відображення на відповідній сторінці.
- Layout.tsx – використовується для створення загального макету додатку, який буде спільним для всіх сторінок. Зазвичай він містить навігаційне меню, футер та інші елементи, які мають бути присутніми на всіх сторінках. В додатку як основний шаблон, містить бокове навігаційне меню.
- error.tsx – відповідає за відображення сторінки помилки, коли щось пішло не так.
- not-found.tsx – використовується для відображення сторінки 404 – не знайдено.

Для реалізації основного макета веб-додатку використовується компонент Layout. Він забезпечує базову структуру додатку, включаючи навігаційну панель та аудіо плеєр, а також контекст для управління аудіо відтворенням.

```
export default function Layout({
  children,
}): Readonly<{
  children: React.ReactNode;
}> {
  const audioRef = useRef<HTMLAudioElement>(null);

  return (
    <>
      <AudioRefContext.Provider value={{ audioRef }}>
        <audio ref={audioRef} hidden preload="auto" />
        <div className="flex h-screen flex-col-reverse md:flex-row
md:overflow-hidden">
          <div className="w-full flex flex-col justify-between md:w-
64">
            <SideNav />
            <Player />
          </div>
          <div className="flex-grow md:px-6 md:py-4 overflow-y-auto
md:px-12">{children}</div>
        </div>
      </AudioRefContext.Provider>
    </>
  );
}
```

Основний компонент Layout приймає children, що представляє собою дочірні компоненти, які будуть відображатися в основному макеті. Для управління аудіо відтворенням використовується хук useRef, який створює посилання на аудіо елемент. Це посилання передається через провайдер AudioRefContext.Provider дочірнім компонентам, дозволяючи їм керувати аудіо плеєром. В середині компонента Layout знаходиться елемент audio, який зберігається в прихованому стані та попередньо завантажує аудіо файли. Це забезпечує швидке та зручне відтворення аудіо контенту без видимого відображення на сторінці. Структура компонента включає два основні блоки. Перший блок містить навігаційну панель та аудіо плеєр. Другий блок займає решту простору екрану і використовується для відображення дочірніх компонентів, переданих через children.

З основних можливостей додатку, є функціонал додавання подкасту до списку закладок. Це дозволяє користувачам швидко знаходити свої збережені подкасти та повертатися до них у будь-який час. Ось як виглядає функція додавання подкасту до закладок:

```
export async function addToBookmarks(options: BookmarkOptions){
    // Валідація вхідних даних
    const { success, data } = AddBookmarks.safeParse({ list_type:
options.list_type });
    if (!success) return { status: "error" };
    // Перевірка, чи подкаст вже в закладках
    const isBookmarked = await getUserBookmarkByPodcastId(options.podcast_id,
options.user_id);
    if (isBookmarked === data.list_type) return { status: "success", data:
data.list_type };

    const updated_at = new Date().toISOString().split("T")[0];
    try {
        // Оновлення або створення запису в базі даних
        const query = isBookmarked
        ? sql`UPDATE bookmarks SET list_type = ${data.list_type}, updated_at
= ${updated_at} WHERE user_id = ${options.user_id} AND podcast_id =
${options.podcast_id}`
        : sql`INSERT INTO bookmarks (user_id, podcast_id, list_type,
updated_at) VALUES (${options.user_id}, ${options.podcast_id}, ${data.list_type},
${updated_at}) ON CONFLICT (user_id, podcast_id) DO NOTHING`;
        await query;
        return { status: "success", data: data.list_type };
    } catch (error) {
        return { status: "error", message: "Database Error: Failed to add to
bookmarks.", error: error };
    }
}
```

Для валідації типів є кілька варіантів. Хоча можна вручну перевіряти типи, використання бібліотеки для валідації типів може зекономити час і зусилля. Для додатку ми будемо використовувати Zod – бібліотеку для TypeScript, яка спрощує цю задачу [21].

Функція `addToBookmarks` виконує додавання подкасту до відповідного списку закладок користувача. Вона приймає три параметри: ідентифікатор користувача, ідентифікатор подкасту та тип списку. Першим кроком є перевірка коректності типу списку закладок. Після цього виконується перевірка, чи вже додано подкаст до закладок користувача. Якщо подкаст вже знаходиться в

закладках користувача і його тип списку збігається з наданим, функція повертає статус «успішно». Якщо ж подкаст ще не додано або тип списку відрізняється, функція продовжує роботу. Вона перевіряє, чи подкаст вже знаходиться в закладках. У разі наявності подкасту в закладках, запис у базі даних оновлюється, а якщо подкаст новий, створюється запис у таблиці bookmarks.

Для забезпечення збереження файлів у веб-додатку використовується функція saveFile, яка дозволяє зберігати файли на сервері. Вона приймає три параметри: шлях до файлу, ідентифікатор подкасту та сам файл. Розглянемо її реалізацію більш детально.

```
export const saveFile = async (file_path: string, podcastId: string, file: File) => {
    const fileBuffer = await file.arrayBuffer();

    const folderPath = path.join(process.cwd(), "public", "assets", "podcasts",
podcastId, file_path);
    const newFolderPath = path.dirname(folderPath);

    if (!fs.existsSync(newFolderPath)) {
        fs.mkdirSync(newFolderPath, { recursive: true });
    }

    fs.writeFileSync(folderPath, Buffer.from(fileBuffer));
};
```

Функція починається з отримання буфера даних файлу. Це досягається за допомогою методу arrayBuffer(), який дозволяє працювати з файлом як з масивом байтів. Це необхідно для подальшої обробки та збереження файлу. Далі формується шлях, за яким буде збережено файл. Цей шлях включає поточну робочу директорію, піддиректорію public/assets/podcasts, ідентифікатор подкасту та відносний шлях файлу. Перед записом файлу функція перевіряє наявність директорії, куди буде збережено файл. Якщо така директорія не існує, вона створюється рекурсивно. За допомогою методу writeFileSync(), файл записується на диск за сформованим шляхом.

Для захисту маршрутів у веб-додатку використовується middleware. Це проміжне програмне забезпечення, яке дозволяє розробникам перехоплювати,

змінювати та контролювати потік запитів і відповідей у своїх програмах. У нашому випадку, middleware перевіряє, чи має користувач доступ до захищених маршрутів та перенаправляє його, якщо сесія неактивна або несанкціонована [22].

```
const protectedRoutes = ["/p"];
const publicRoutes = ["/login", "/signup", "/"];

export default async function middleware(req: NextRequest) {
  await updateSession(req);

  const path = req.nextUrl.pathname;
  const isProtectedRoute = protectedRoutes.some(route =>
    path.startsWith(route));
  const isPublicRoute = publicRoutes.includes(path);

  const cookie = req.cookies.get("session")?.value;
  if (!cookie && isProtectedRoute) return NextResponse.redirect(new
    URL("/login", req.nextUrl));
  if (!cookie) return;

  const session = await decrypt(cookie);

  if (isProtectedRoute && !session?.userId) {
    return NextResponse.redirect(new URL("/login", req.nextUrl));
  }

  if (isPublicRoute && session?.userId &&
    !req.nextUrl.pathname.startsWith("/p")) {
    return NextResponse.redirect(new URL("/p", req.nextUrl));
  }

  return NextResponse.next();
}

export const config = {
  matcher: ["/(?!api|_next/static|_next/image|.*\\.png$).*"],
};
```

Основна функція middleware отримує запит req, оновлює сесію користувача та перевіряє маршрути, до яких намагається звернутися користувач. Маршрути поділяються на захищені і загальнодоступні. Якщо сесія не дійсна, а запит здійснюється до захищеного маршруту, користувача перенаправляють на сторінку входу. Якщо користувач спробує звернутися до загальнодоступного маршруту, вже маючи активну сесію, і цей маршрут не є частиною захищених маршрутів, його перенаправляють на захищену сторінку «/p». Функція повертає NextResponse.next(), що дозволяє обробку запиту продовжити без змін, якщо всі перевірки пройдені

успішно. Конфігурація `matcher` вказує, до яких маршрутів буде застосовуватись це `middleware`.

Висновок до другого розділу

Другий розділ було присвячено проектуванню та реалізації веб-додатку для прослуховування подкастів, де детально розглянуто варіанти використання, об'єктно-орієнтовану структуру системи та реалізацію функціоналу. Було встановлено, що основними користувачами системи є звичайні користувачі, творці контенту та адміністратори, кожен з яких має специфічний набір функцій. Звичайні користувачі можуть переглядати, шукати та слухати подкасти, в той час як творці контенту отримують можливість додавати нові подкасти та епізоди. Адміністратори, маючи найширший спектр можливостей, здатні керувати контентом і користувачами через панель адміністратора.

Діаграми варіантів використання та активностей допомагають візуально представити різні сценарії взаємодії користувачів із системою, що сприяє кращому розумінню цілей, які система допомагає досягти. UML-діаграми, зокрема діаграми класів, відображають структуру системи, демонструючи взаємозв'язки між класами та їх атрибутами.

Розробка бази даних системи забезпечує надійне зберігання даних та швидкий доступ до них, використовуючи реляційну базу даних PostgreSQL. Було визначено основні таблиці бази даних, які відповідають за збереження користувачів, подкастів, епізодів, категорій, тегів та закладок, а також зв'язків між подкастами та категоріями і тегами.

Реалізація функціоналу веб-додатку для прослуховування подкастів ґрунтується на використанні Next.js, який забезпечує маршрутизацію на основі файлової системи. Архітектура додатку розділена на клієнтську та серверну частини.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ДОДАТКОМ ДЛЯ ПРОСЛУХОВУВАННЯ ПОДКАСТІВ

3.1. Порядок встановлення та налаштування параметрів додатку

Веб-додаток для прослуховування подкастів складається з кількох компонентів, які потребують окремого розгортання та налаштування. Для демонстрації компонентів системи та візуалізації процесу їх розгортання була створена діаграма розгортання, яка зображена на рис. 3.1:

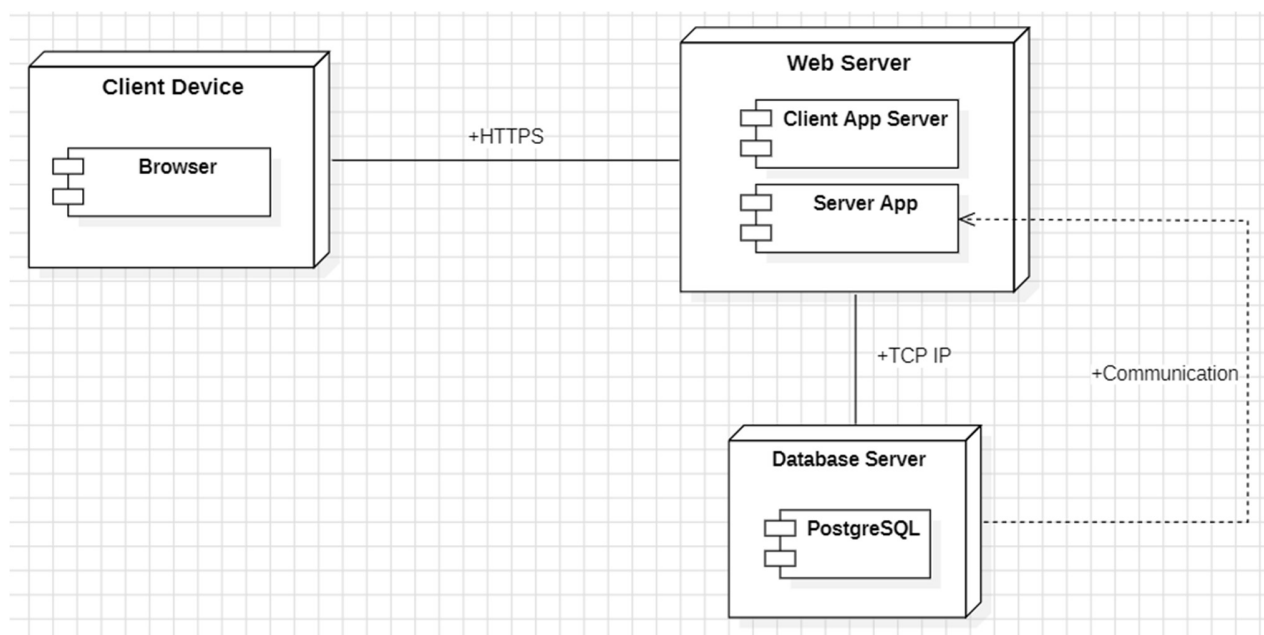


Рис. 3.1. Діаграма розгортання

Діаграма розгортання – це графічне зображення, яке показує, як програмні компоненти фізично розташовані на апаратних пристроях. Вона демонструє спосіб взаємодії програмних частин з фізичними ресурсами системи, такими як сервери, процесори, пристрої зберігання та мережеві з'єднання [23].

Основними елементами системи є клієнтські пристрої, веб-сервер і сервер бази даних. Клієнтські пристрої, такі як настільні комп'ютери, ноутбуки, планшети та смартфони, взаємодіють з додатком через браузер, використовуючи протоколи HTTP/HTTPS. Це дозволяє користувачам отримувати доступ до додатку, переглядати та слухати подкасти.

HTTP є протоколом або набором правил для комунікації клієнт-сервер. Коли користувач відвідує веб-сайт, браузер відправляє HTTP-запит на веб-сервер, який відповідає HTTP-відповіддю. В свою чергу, HTTPS є більш безпечною версією HTTP. У HTTPS браузер і сервер встановлюють безпечне, зашифроване з'єднання перед передачею даних [24].

Веб-сервер є центральним елементом системи, на якому розміщено два основних компоненти: Client App Server та Server App. Client App Server відповідає за обробку клієнтських запитів та забезпечення інтерфейсу користувача, тоді як Server App виконує основну логіку і взаємодіє з базою даних. Взаємодія між цими двома компонентами відбувається через внутрішні комунікаційні канали, що забезпечує ефективну роботу додатку. Сервер бази даних зберігає всю необхідну інформацію для роботи додатку, включаючи дані про користувачів, подкасти, епізоди та закладки. Для взаємодії з веб-сервером використовуються протоколи TCP/IP, що забезпечує надійну передачу даних між Server App та базою даних PostgreSQL.

3.2. Структура інтерфейсу веб-додатку для прослуховування подкастів

Користувачі мають можливість управляти додатком з будь-якого пристрою, що забезпечує їм зручний доступ. Це можуть бути як комп'ютери, так і планшети та смартфони.

Коли користувач перейде за посиланням на додаток, перше, що він побачить, буде стартова сторінка (рис. 3.2). На цій сторінці користувач побачить дві основні опції: увійти в існуючий акаунт або створити новий. Щоб отримати повний доступ до функціоналу додатку, користувач повинен авторизуватися. Якщо він вже має акаунт, він може ввести свої облікові дані та увійти. Якщо ж користувач ще не зареєстрований, він може швидко створити новий акаунт, пройшовши процес реєстрації.

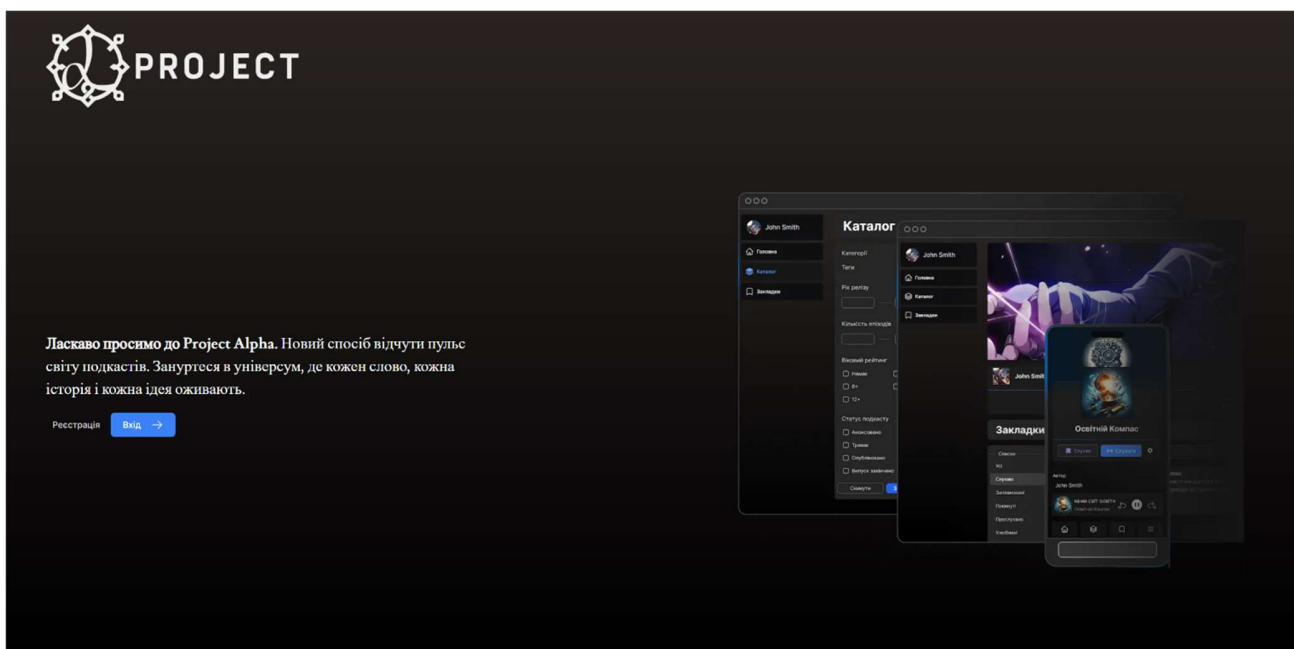


Рис. 3.2. Стартова сторінка

Почнемо з реєстрації нового користувача (рис. 3.3). Процес реєстрації включає в себе чотири кроки. Перший крок полягає у введенні електронної адреси. Користувач повинен вказати свою електронну пошту в спеціальному полі.

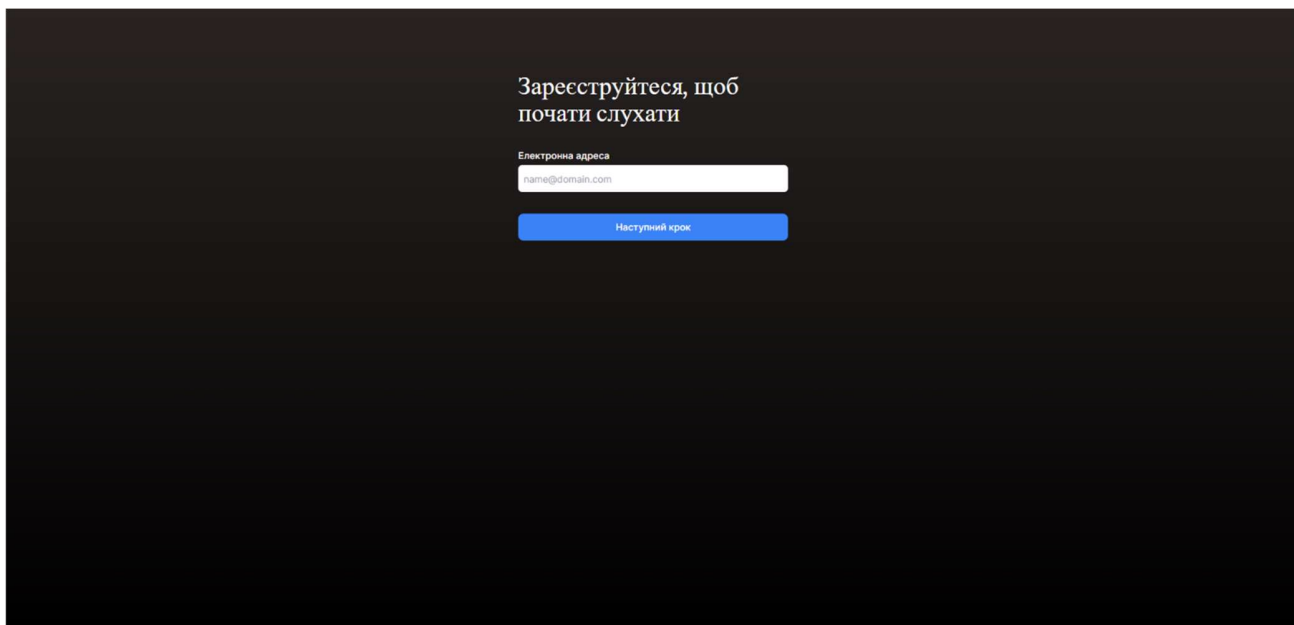


Рис. 3.3. Реєстрація користувача – введення електронної пошти

Після натискання кнопки «Наступний крок» система на клієнті перевіряє, чи правильно введена адреса. Якщо введена адреса є правильною, користувач

автоматично переходить до наступного кроку. Якщо ж введена адреса некоректна, система виведе повідомлення про помилку, і користувачеві потрібно буде ввести адресу повторно.

Наступний крок – це створення паролю (рис. 3.4).

Рис. 3.4. Реєстрація користувача – створення паролю

Для того, щоб пройти валідацію, пароль повинен містити щонайменше вісім символів, одну літеру, а також одне число або спеціальний символ. Користувачеві також надається можливість показати або приховати введений пароль. Крім того, користувач може повернутися назад до попереднього кроку, якщо йому потрібно змінити або перевірити введені раніше дані. Після того, як користувач ввів пароль, і всі пункти умов валідації стали зеленого кольору, з'являється можливість перейти до наступного пункту – введення персональної інформації (рис. 3.5).

Крок 2 з 3
Розкажіть про себе

Ім'я
James Bond

Дата народження
11 Листопад 1977

Гендер/стать
Чоловіча

Наступний крок

Рис. 3.5. Реєстрація користувача – персональна інформація

На цьому етапі користувач має заповнити поля з особистими даними, такими як ім'я, дата народження та гендер/стать. У разі помилки валідації, користувач побачить повідомлення про помилку під проблемним полем. Це допоможе користувачеві зрозуміти, що саме потрібно виправити, щоб продовжити процес реєстрації (рис. 3.6).

Крок 3 з 3
Угоди та підтвердження

☒ Я згоден із Умовами використання та Політикою конфіденційності.

☐ Я б хотів уникати отримання маркетингових повідомлень від Project Alpha.

Зареєструватися

Рис. 3.6. Реєстрація користувача – угоди та підтвердження

Останній етап реєстрації включає згоду користувача з умовами використання та політикою конфіденційності, а також вибір щодо отримання повідомлень розсилки.

При натисканні кнопки "Зареєструватися" форма відправляється на сервер. На сервері дані проходять додаткову валідацію для перевірки коректності. Якщо валідація успішна, дані користувача зберігаються у відповідній таблиці бази даних. У разі успішного збереження, основна його інформація зберігається в сесії, а користувача перенаправляє на головну сторінку веб-додатку. Якщо ж виникають помилки, сервер повертає відповідне повідомлення, яке відображається користувачеві для виправлення.

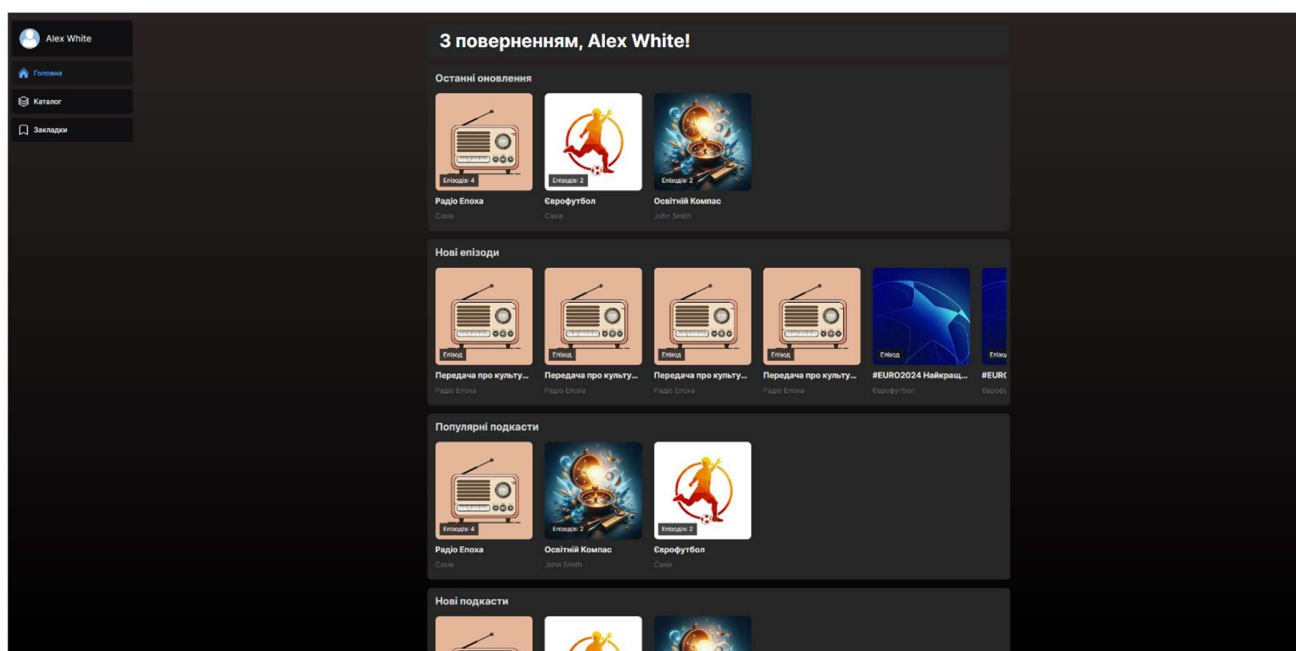


Рис. 3.7. Головна сторінка додатку

Користувача зустрічає інтерфейс головної сторінки, на якій представлені кілька різних секцій (рис. 3.7). Наприклад, секція "Останні оновлення" показує подкасти, що були нещодавно оновлені. На ній відображаються максимум десять подкастів, які були останніми оновлені, мають принаймні один епізод і є публічними. Це дозволить користувачам швидко ознайомитися з новим контентом і знайти подкасти, що зможуть їх зацікавити.

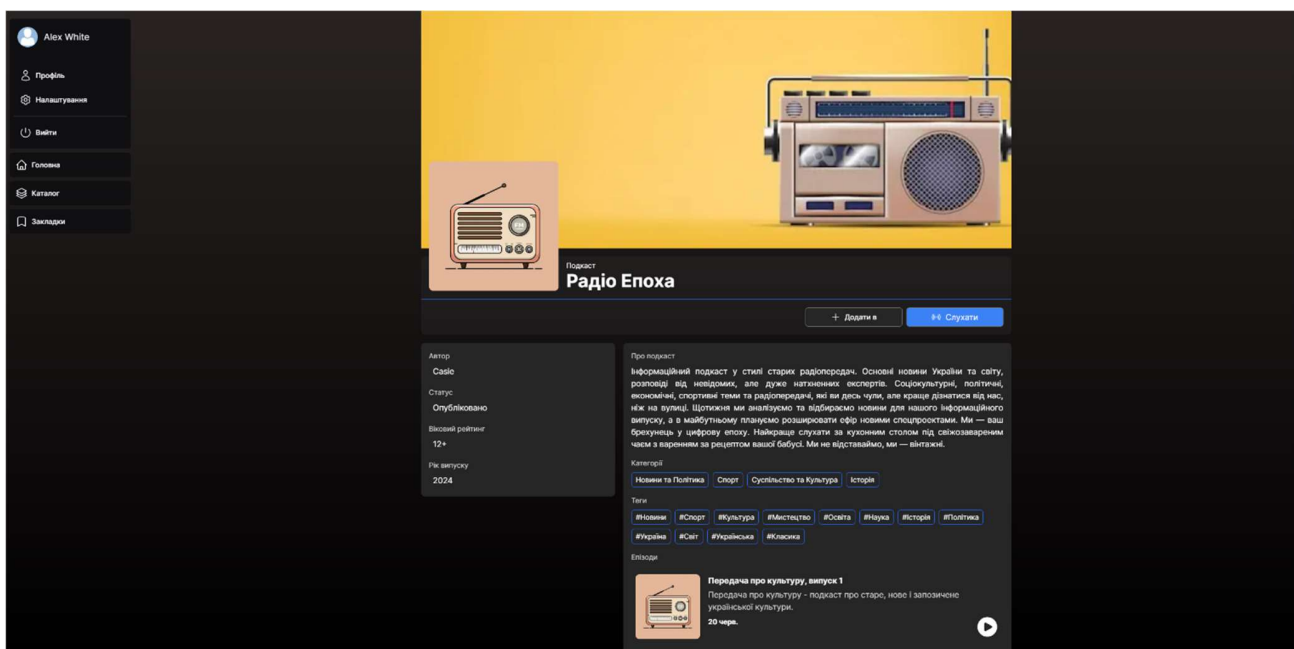


Рис. 3.8. Сторінка подкасту

Сторінка подкасту поділяється на три основних блоки (рис. 3.8). Перший блок – це хедер, де відображаються банер, обкладинка, назва подкасту, а також кнопки для взаємодії. Користувач може зберегти подкаст в один зі списків закладок, натиснувши кнопку «Додати в», або почати прослуховування подкасту, натиснувши кнопку «Слухати». Під хедером розташовані два блоки з загальною інформацією. У лівому блоці користувач може переглянути інформацію про автора, статус подкасту, віковий рейтинг та рік випуску. Правий блок містить основну інформацію: опис подкасту, категорії, до яких він належить, та теги. Також у цьому блоці користувач може переглянути список епізодів подкасту.

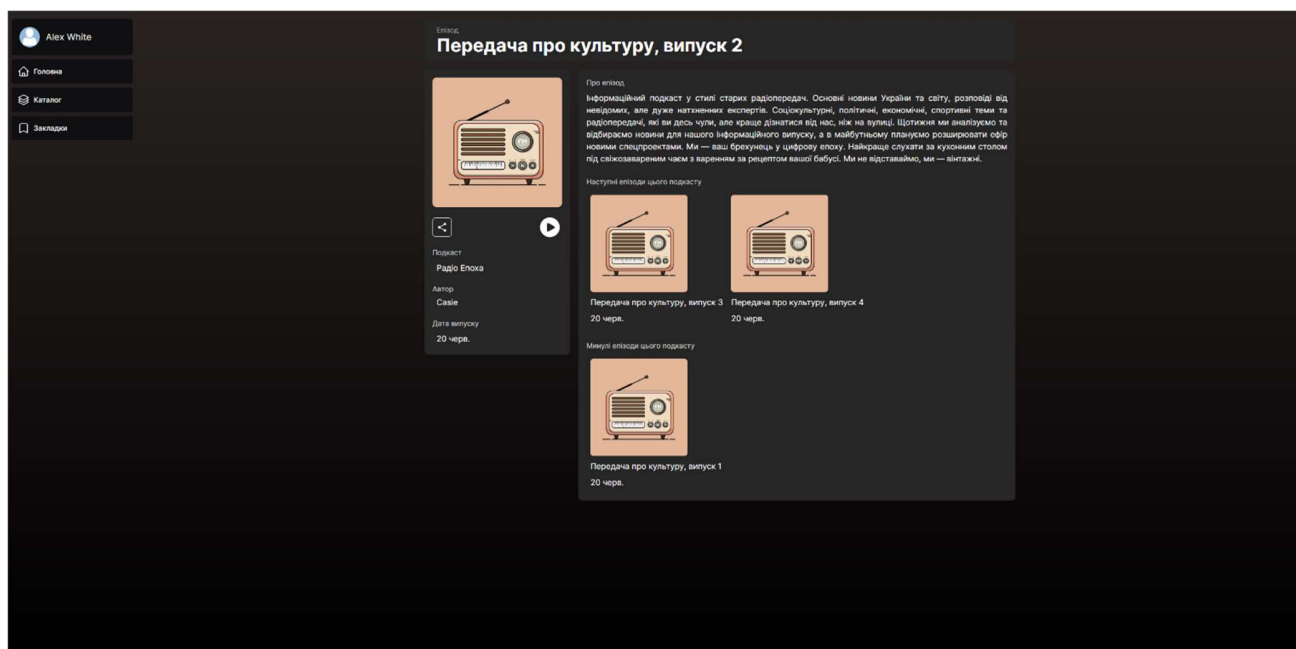


Рис. 3.9. Сторінка епізоду

Сторінка епізоду поділена на два блоки (рис. 3.9). З лівого боку відображається обкладинка епізоду, кнопки для копіювання посилання на епізод та його прослуховування, назва подкасту, до якого належить епізод, ім'я автора та дата релізу. Праворуч знаходиться інформація про подкаст та перелік наступних і попередніх епізодів, якщо вони є.

Важливою складовою додатка для прослуховування подкастів є відтворення епізодів. Для керування цим відтворенням був розроблений плеєр (рис. 3.10).

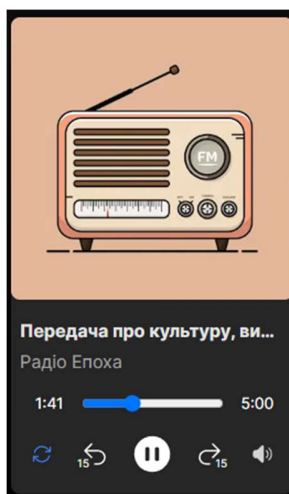


Рис. 3.10. Плеєр

Веб-плеєр надає користувачам зручні інструменти для керування прослуховуванням. У плеєрі є можливість паузи та відтворення, перемотування назад і вперед на 15 секунд, зациклення аудіо, регулювання гучності та перемотування за допомогою доріжки прогресу.

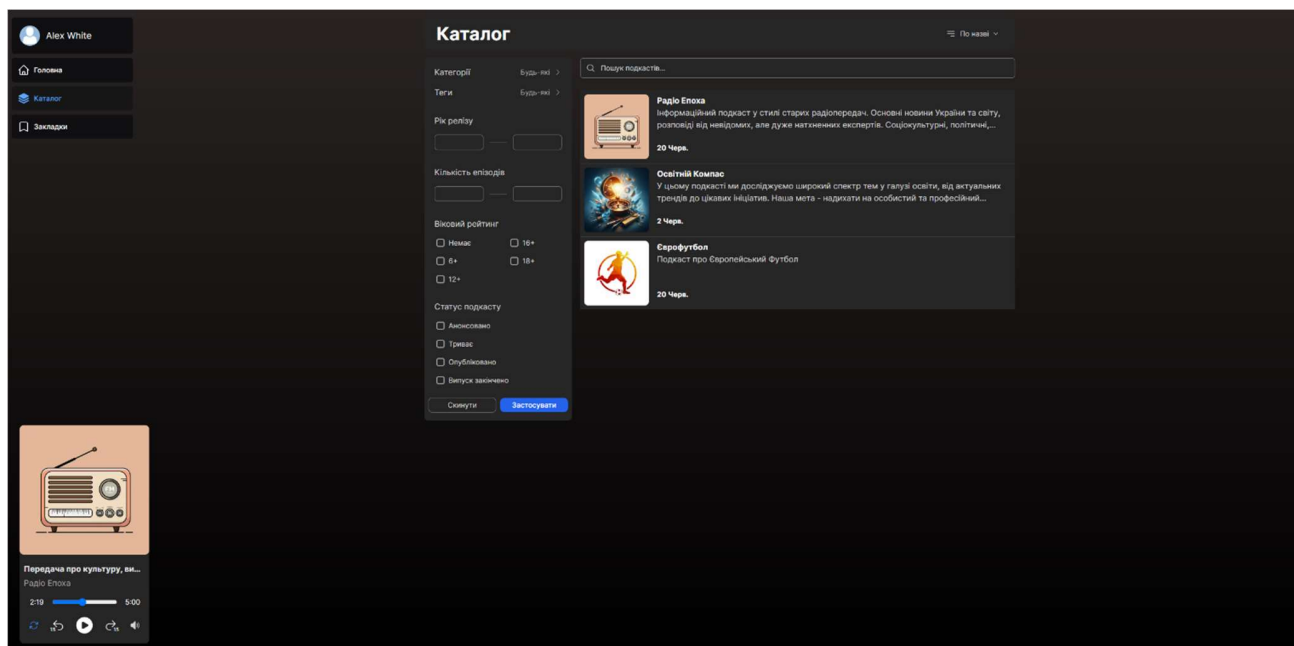


Рис. 3.11. Сторінка каталогу

На сторінці каталогу користувач може знайти усі доступні подкасти (рис. 3.11). Також є можливість пошуку, сортування та фільтрації.

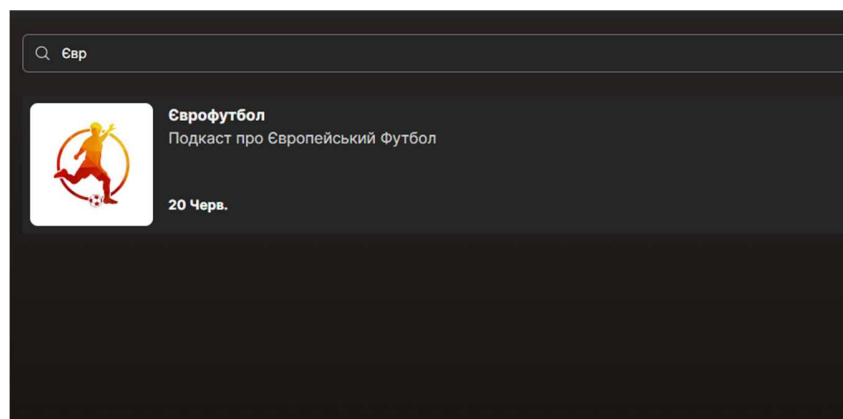


Рис. 3.12. Пошук подкастів

Під час вводу в поле пошуку URL оновлюється при кожному натисканні клавіші, і, таким чином, робиться запит до бази даних при кожному натисканні

клавiші (рис. 3.12). Це не є проблемою, оскільки поки додаток невеликий, але якщо припустити, що додаток матиме тисячі користувачів, кожен з яких відправлятиме новий запит до бази даних при кожному натисканні клавiші. Debouncing - це програмна практика для обмеження частоти виконання функції. У нашому випадку, потрібно робити запит до бази даних тільки тоді, коли користувач припинив вводити текст. Для виконання цієї задачі був використаний хук use-debounce [25].

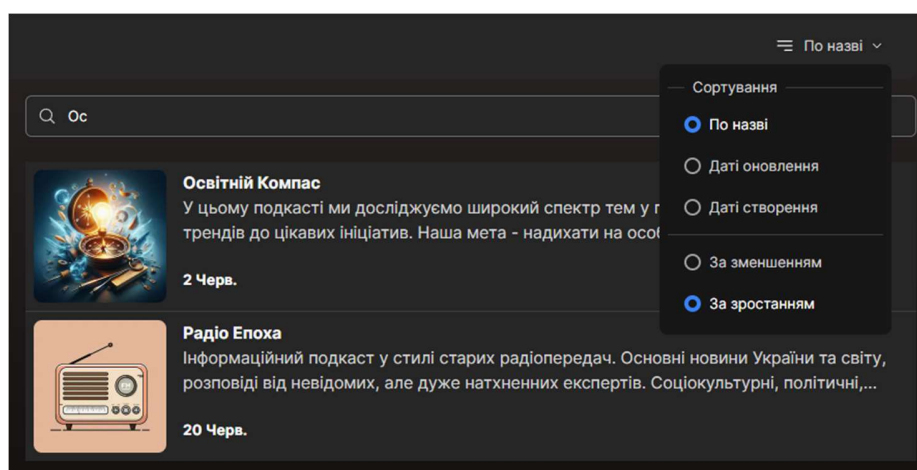


Рис. 3.13. Сортування списку подкастів.

Сортування списків може відбуватися по трьом категоріям: по назві, по даті оновлення та по даті створення. Також можна виставити порядок сортування за зростанням або за зменшенням (рис. 3.13).

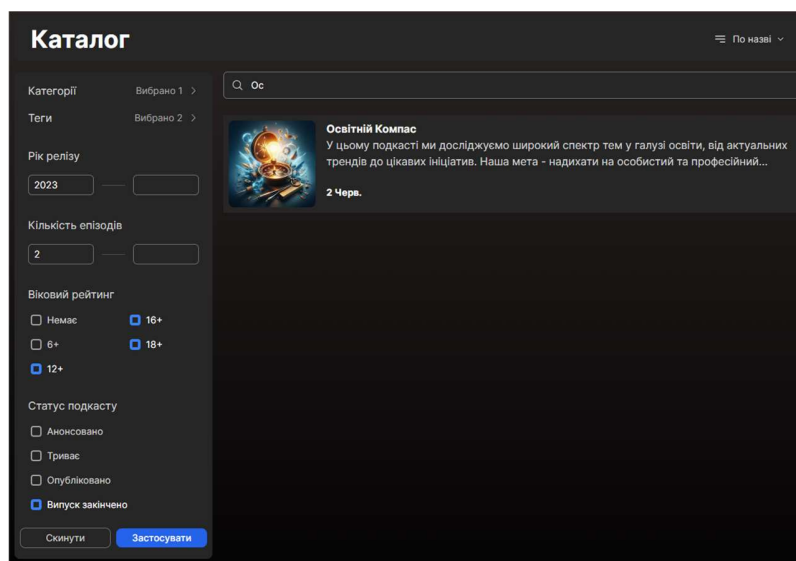


Рис. 3.14. Фільтрація списку подкастів

Фільтрація в додатку може відбуватися за кількома параметрами (рис. 3.14). Користувач може обрати цікаві для нього категорії та теги, ввести рік випуску від і до, а також вказати бажану кількість епізодів. Крім того, можна вибрати віковий рейтинг та статус подкасту. Після натискання кнопки «Застосувати» відбувається запит до бази даних, і каталог подкастів оновлюється відповідно до обраних фільтрів.

Давайте тепер додамо подкаст до списку закладок (рис. 3.15).

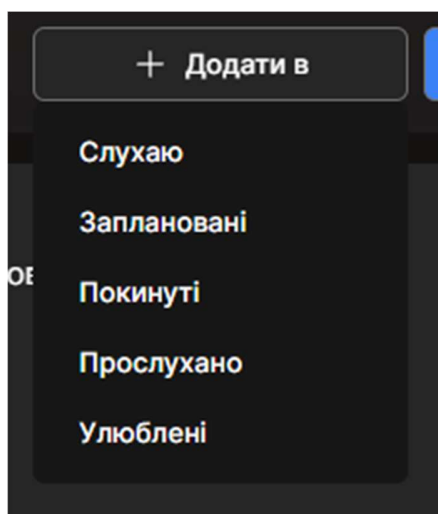


Рис. 3.15. Додавання подкасту до закладок

Існує п'ять різних списків закладок:

- Слухаю: сюди користувач додає подкасти, які він слухає в даний момент;
- Заплановані: сюди можна додати подкасти, які користувач планує прослухати;
- Покинуті: тут зберігаються подкасти, які користувач почав слухати, але не закінчив;
- Прослухано: сюди потрапляють подкасти, які користувач уже прослухав до кінця;
- Улюблені: в цьому списку зберігаються улюблені подкасти, які користувач хоче прослухати знову.

Після додавання подкасту до закладок, кнопка змінить свій вигляд (рис. 3.16)

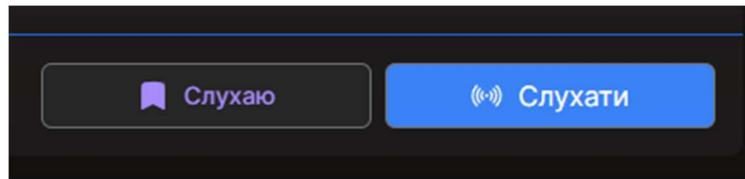


Рис. 3.16. Зміна вигляду кнопки додавання до закладок

Тепер перейдемо до сторінки закладок і подивимось, чи додався подкаст до списку (рис. 3.17).

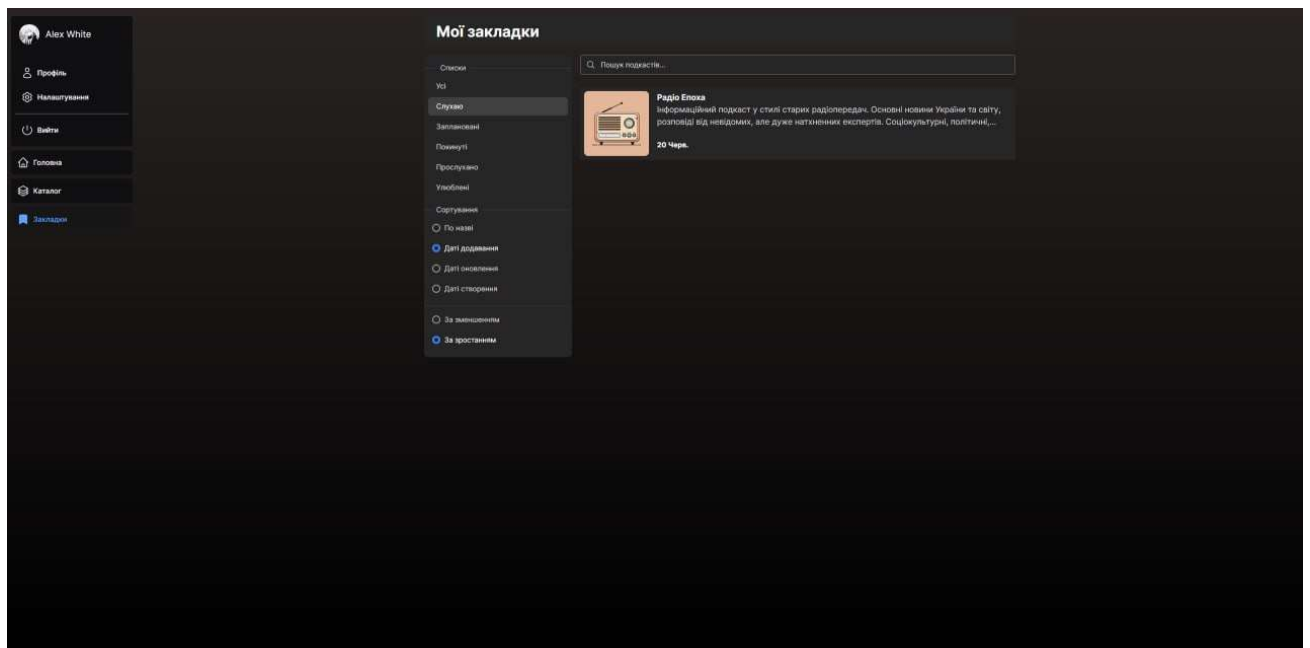


Рис. 3.17. Сторінка закладок

При переході на сторінку закладок, користувач одразу бачить список подкастів, доданих до списку «Слухаю». Тут також реалізовані пошук та можливість сортування. Ліворуч від списку подкастів розташована панель, що дозволяє перемикатися між різними списками закладок та вибирати спосіб сортування. На панелі є, також, вкладка зі всіма збереженими в закладки подкастами.

Перейдемо до профілю користувача (рис. 3.18).

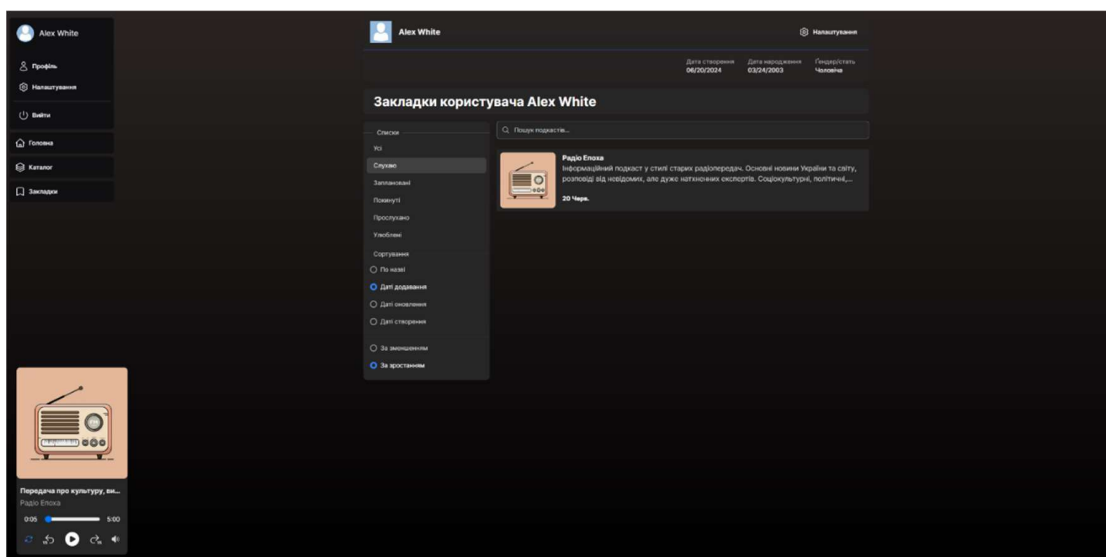


Рис. 3.18. Сторінка профілю

Оскільки наш користувач щойно зареєструвався, його профіль виглядає досить порожнім. Замість аватара відображається плейсхолдер. Поруч з аватаром знаходиться ім'я користувача, а з іншого боку блоку розташована кнопка для налаштувань профілю. Трохи нижче представлена інформація про користувача, включаючи дату народження, дату створення облікового запису та стать. Під цим блоком знаходяться списки його закладок, які можуть переглядати всі користувачі додатку.

Давайте перейдемо до налаштування профілю користувача (рис. 3.19).

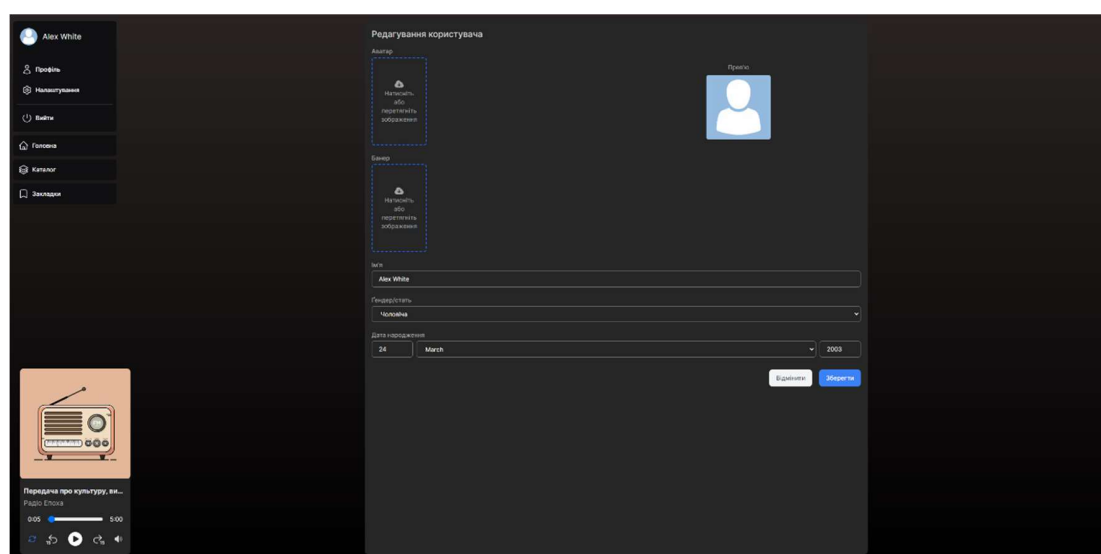


Рис. 3.19. Сторінка редагування користувача

На цій сторінці користувач має можливість персоналізувати свій профіль, зробивши його більш привабливим та інформативним. Він може змінити свій аватар, обравши нове зображення для представлення себе в додатку. Крім того, користувач може налаштувати банер свого профілю, додавши індивідуальний фон, який підкреслює його особистість або інтереси. Також доступна опція редагування особистих даних, таких як ім'я, стать та дата народження. Після відправки форми, користувача перенаправляє назад на його вже змінений профіль.

Тож давайте додамо свій аватар та банер до цього профілю (рис. 3.20).

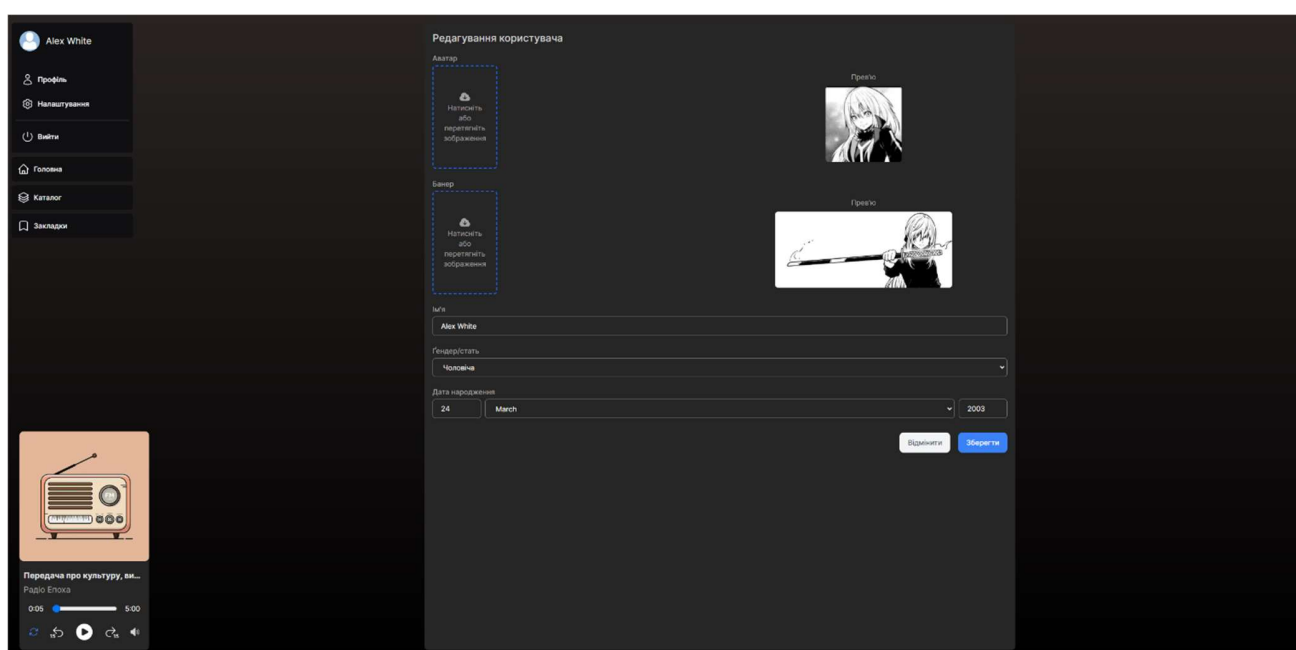


Рис. 3.20. Редагування профілю

При завантаженні своїх зображень аватару та банеру, користувач може побачити їх прямо у додатку на прев'ю.

Давайте збережемо зміни і подивимось як зміниться профіль.

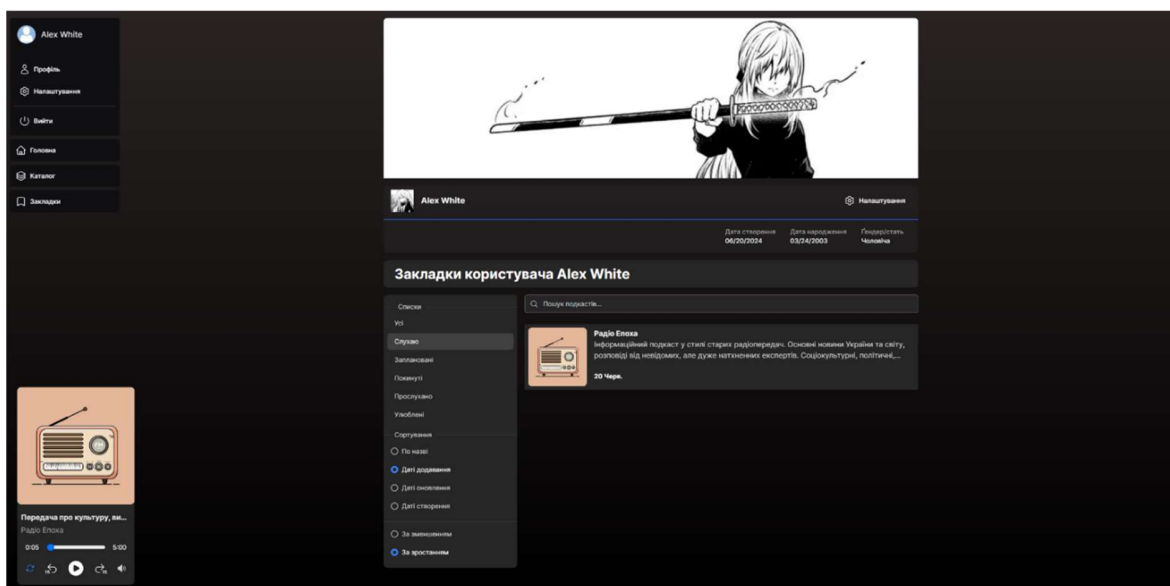


Рис. 3.21. Сторінка оновленого профілю

Після внесення змін ми можемо побачити, що зображення аватара було оновлено, і нове зображення тепер відображається на профілі користувача (рис. 3.21). Крім того, зверху сторінки з'явився банер, який додає індивідуальний стиль та підкреслює унікальність профілю. Тепер профіль виглядає більш привабливим.

Тепер давайте змінимо користувача (рис. 3.22). У нашій системі є три типи користувачів: звичайний зареєстрований користувач, творець контенту та адміністратор.

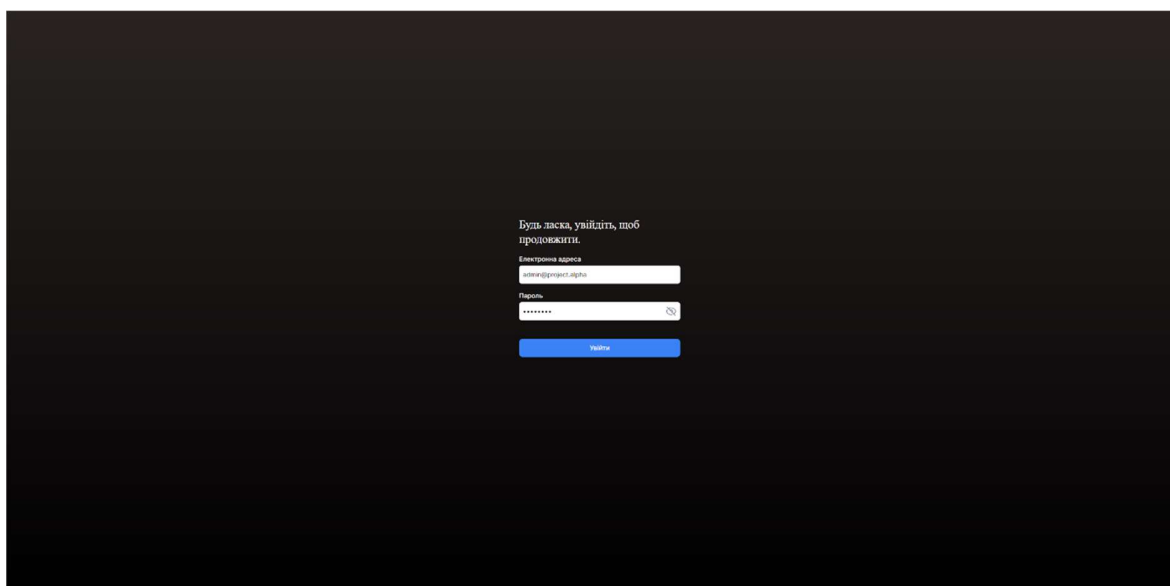


Рис. 3.22. Сторінка входу

Роль творця контенту дає можливість користувачу створювати свої подкасти так керувати ними. В навігаційному меню з'являється кнопка створення нового подкасту (рис. 3.23).

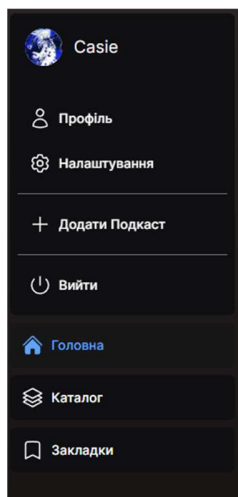


Рис. 3.23. Навігаційне меню творця контенту

Оскільки адміністратор має ті ж можливості, що і творець контенту, ми одразу перейдемо до розгляду його облікового запису. Звісно, адміністратор може не тільки створювати нові подкасти, а й управляти контентом додатку. Для цього, у нього в навігаційному меню є можливість перейти до панелі управління (рис. 3.24).

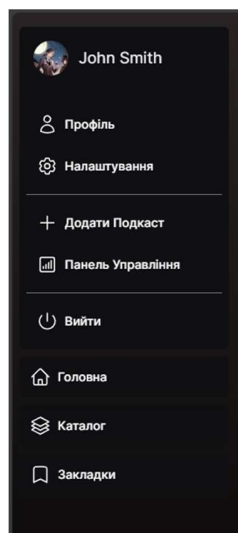


Рис. 3.24. Навігаційне меню адміністратора

Але про це ми поговоримо пізніше, а зараз розглянемо можливості творця контенту, зокрема процес створення нового подкасту.

Після натискання кнопки «Додати подкаст» користувача перенаправляє на форму створення подкасту (рис. 3.25).

Рис. 3.25. Сторінка створення нового подкасту

Для того щоб створити новий подкаст, користувач повинен завантажити обкладинку та банер для подкасту. Треба звернути увагу, що обкладинка буде представляти подкаст у списках та пошуку, а банер стане частиною візуального оформлення сторінки самого подкасту. Проте, якщо того не хоче користувач, він може не завантажувати свій банер. Тоді його сторінка подкасту буде відображатися без банеру.

Далі потрібно ввести назву та опис подкасту. Назва повинна бути короткою та лаконічною, щоб привертати увагу слухачів. Тоді як опис має надавати більше інформації про зміст подкасту. Після цього користувач обирає віковий рейтинг подкасту, категорії та теги для подкасту. Категорії допомагають класифікувати подкаст за темою, а теги дозволяють покращити пошук і навігацію.

Давайте заповнимо новий подкаст (рис. 3.26).

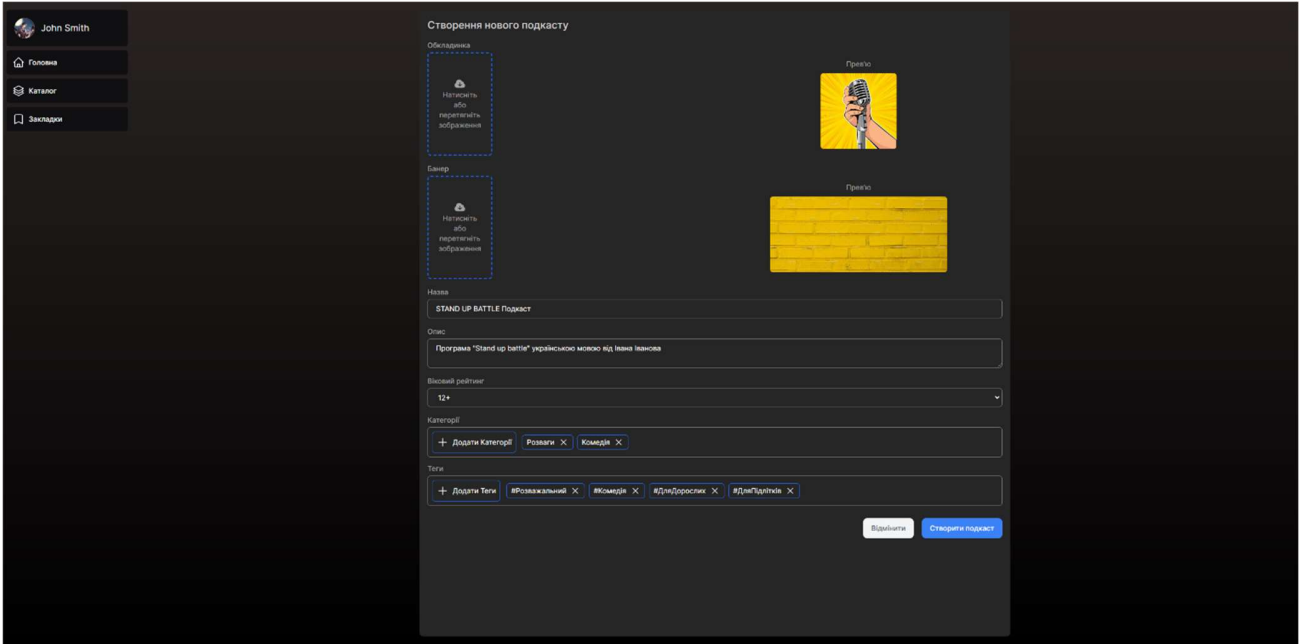


Рис. 3.26. Створення подкасту

Так само, як це було з налаштуванням профілю, користувач може переглянути завантажені обкладинку та банер перед відправкою форми.

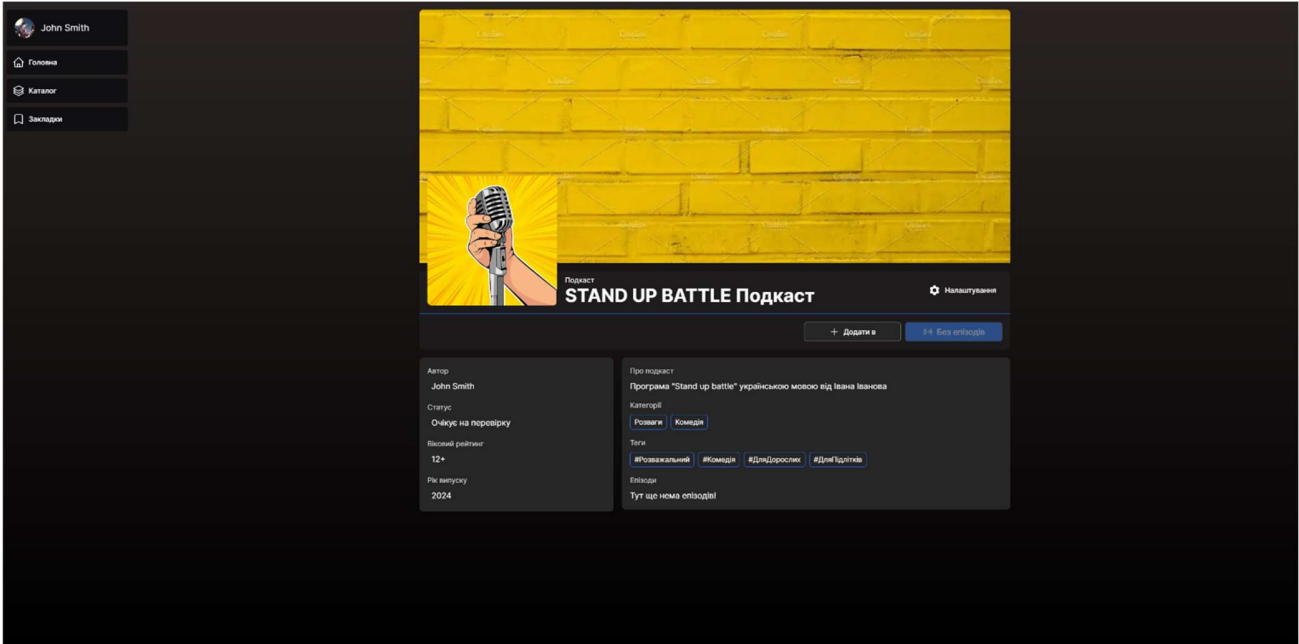


Рис. 3.27. Сторінка створеного подкасту

Після натискання кнопки «Створити подкаст», користувача перенаправляє на сторінку створеного ним подкасту (рис. 3.27). Тут ми можемо побачити, що статус подкасту – «Очікує на перевірку». Це означає, що подкаст поки не має публічного

доступу і знаходиться на розгляді у адміністрації. Також, так як подкаст тільки-но створений, епізодів він не має.

Перейдемо до налаштувань подкасту. В налаштуваннях є п'ять вкладок. Перемикатися по вкладкам можна за допомогою меню зліва.

Перша вкладка – це Основна інформація (рис. 3.28).

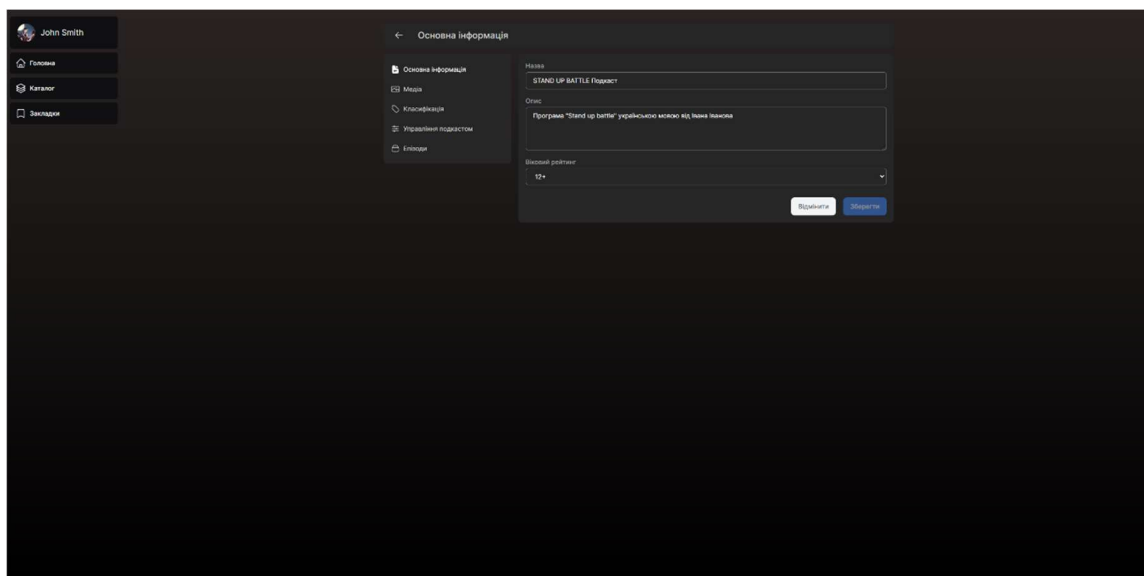


Рис. 3.28. Сторінка налаштувань подкасту – Основна інформація
На цій вкладці можна змінити назву, опис та віковий рейтинг подкасту.
Наступна – це вкладка Медіа (рис. 3.29).

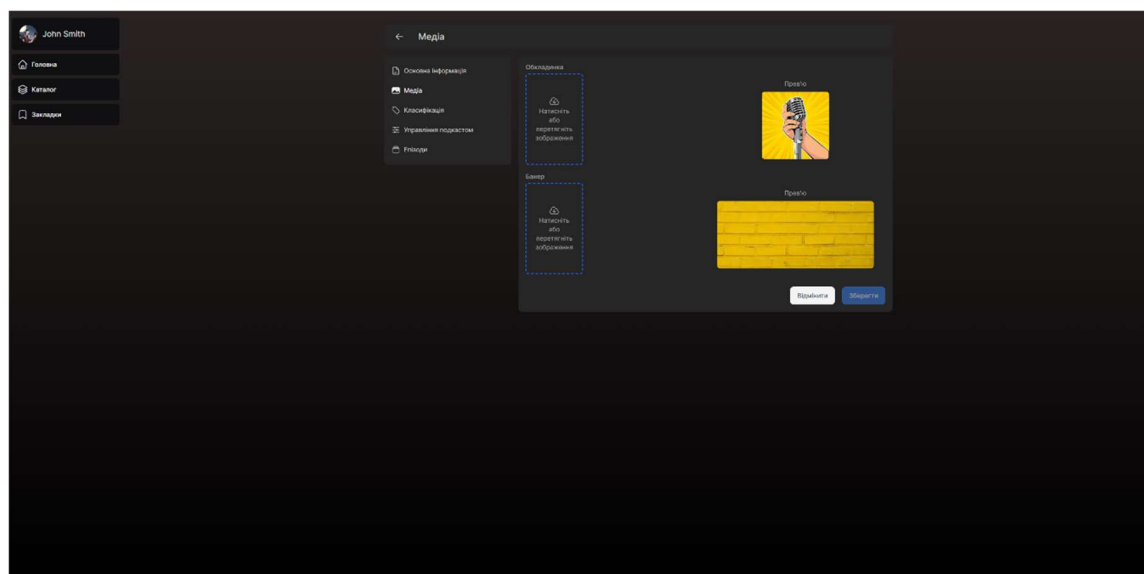


Рис. 3.29. Сторінка налаштувань подкасту – Медіа

Тут можна змінити медіа файли подкасту, а саме зображення обкладинки та банеру.

Далі йде вкладка Класифікації. На ній відбувається керування категоріями та тегами (рис. 3.30).

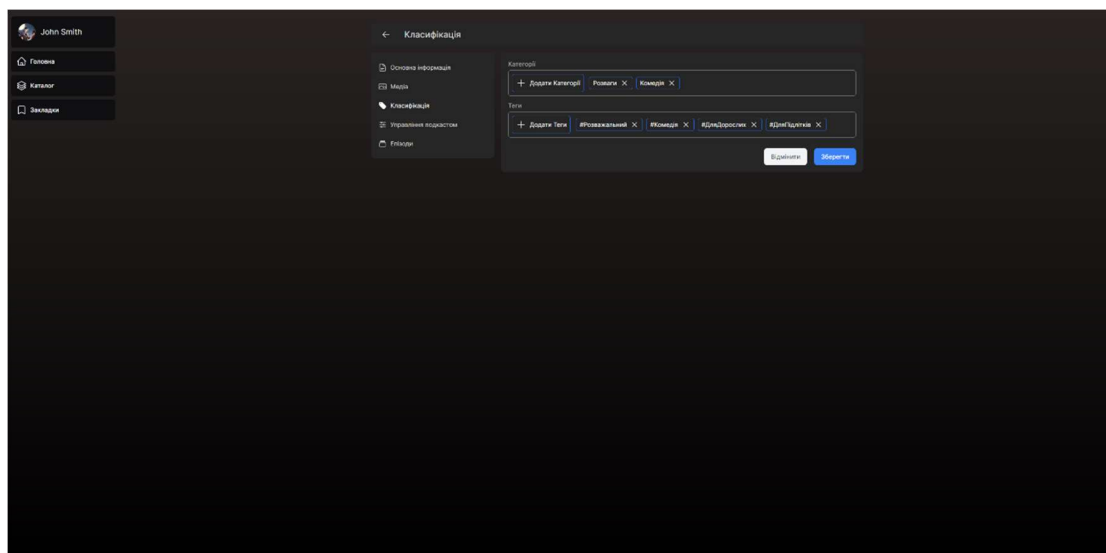


Рис. 3.30. Сторінка налаштувань подкасту – Класифікація
Наступна вкладка – це Управління подкастом (рис. 3.31).

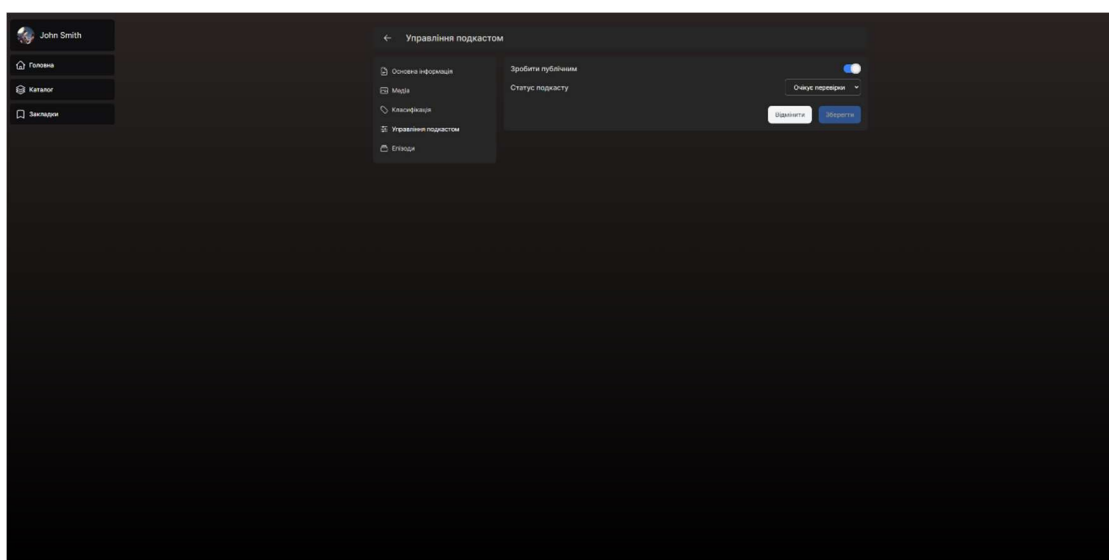


Рис. 3.31. Сторінка налаштувань подкасту – Управління подкастом
Це меню для взаємодії з подкастом. Тут можна регулювати публічність подкасту та змінювати його статус. Якщо автором подкасту є творець і статус його

подкасту – «Очікує перевірки», він не може редагувати цей статус, доки його не перевірять.

Остання вкладка – це вкладка з епізодами подкасту (рис. 3.32).

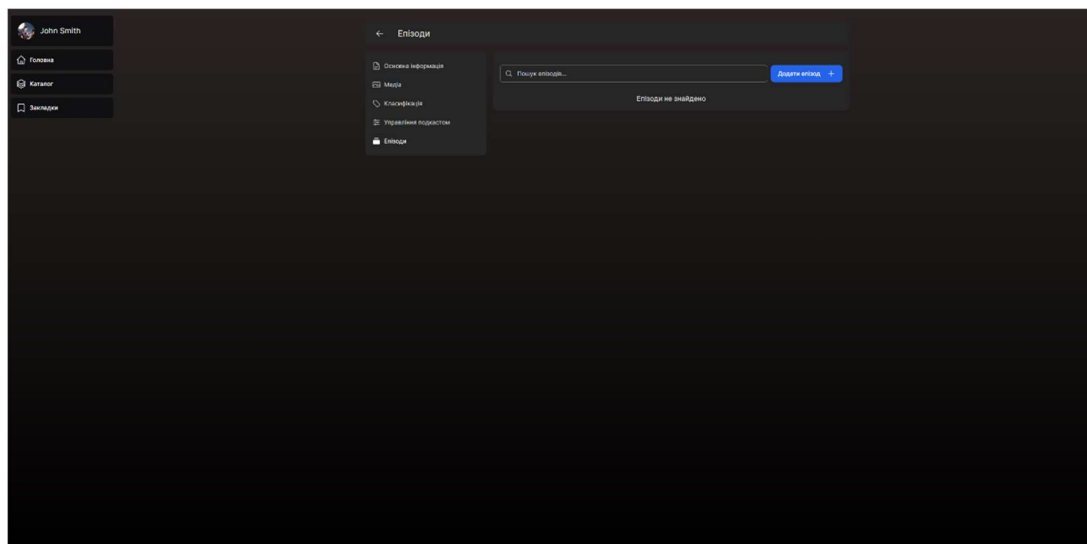


Рис. 3.32. Сторінка налаштувань подкасту – Епізоди

Це ключова вкладка для керування епізодами. Тут можна переглядати список усіх епізодів подкасту, здійснювати пошук потрібних епізодів, а також створювати нові епізоди. До речі про нові епізоди, давайте подивимось, як їх створювати.

Для створення нового епізоду користувач повинен натиснути кнопку «Створити епізод», що відкриє форму для введення деталей (рис. 3.33).

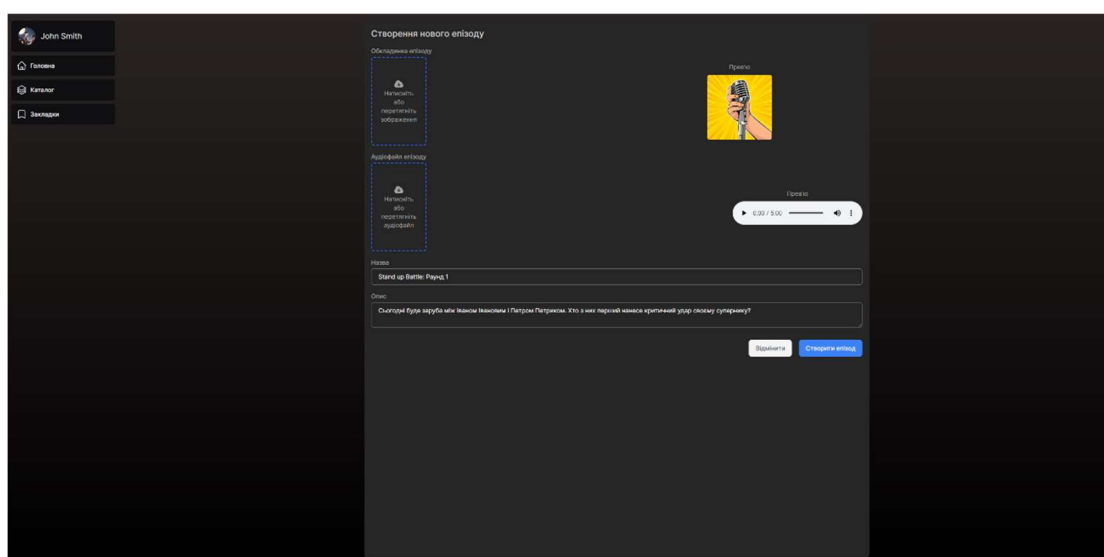


Рис. 3.33. Створення нового епізоду

Для створення нового епізоду потрібно обрати обкладинку епізоду, завантажити аудіофайл, ввести назву та опис епізоду, а потім натиснути кнопку «Створити епізод». Користувача перенаправляє на сторінку подкасту, де він одразу може побачити новостворений епізод (рис. 3.34).

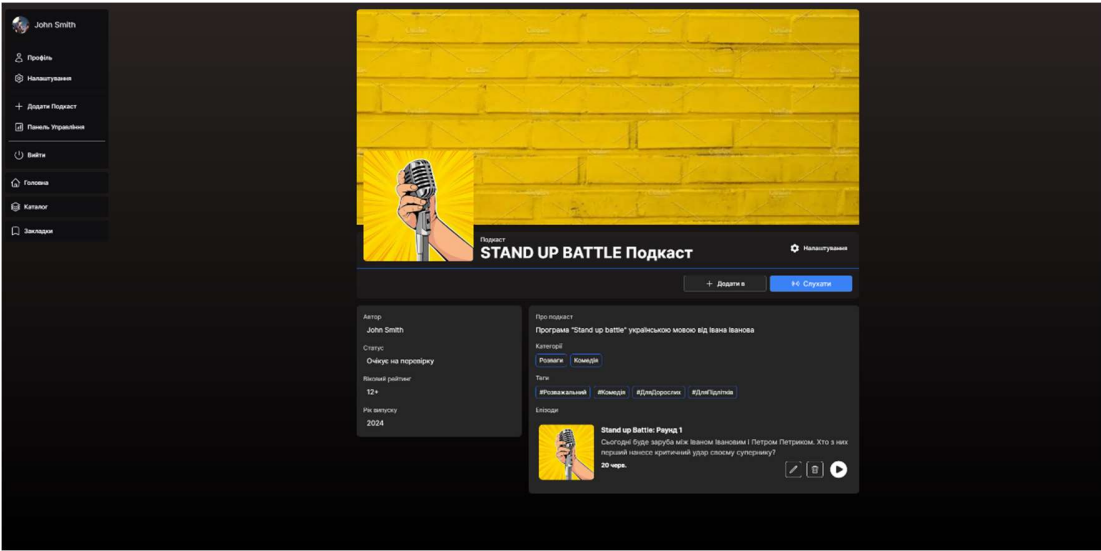


Рис. 3.34. Сторінка подкасту зі створеним епізодом

Автор подкасту та адміністратори мають право редагувати епізод. Це можна зробити натиснувши на олівець у списку епізодів. Для видалення епізоду, треба натиснути на кнопку з корзиною, що знаходиться поруч. Повернемося на вкладку епізодів в налаштуваннях подкасту (рис. 3.35).

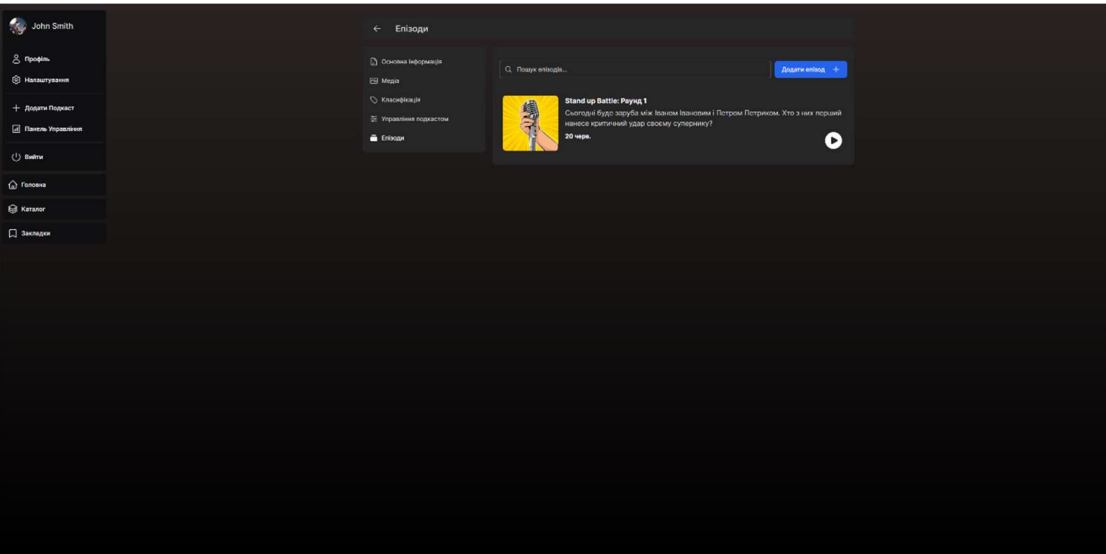


Рис. 3.35. Перегляд списку епізодів в налаштуваннях подкасту

Як ми бачимо, новий епізод успішно додано до списку епізодів подкасту. Перейдемо до перегляду сторінки цього епізоду (рис. 3.36)

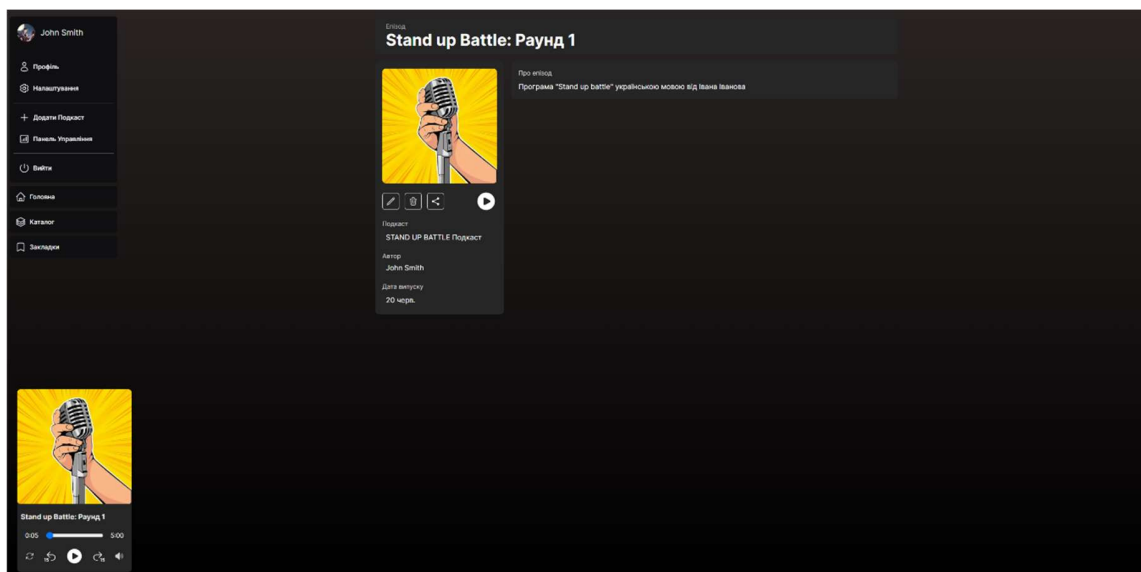


Рис. 3.36. Сторінка створеного епізоду

На цій сторінці ми теж можемо редагувати та видаляти подкаст. Давайте переглянемо сторінку редагування епізоду (рис. 3.37).

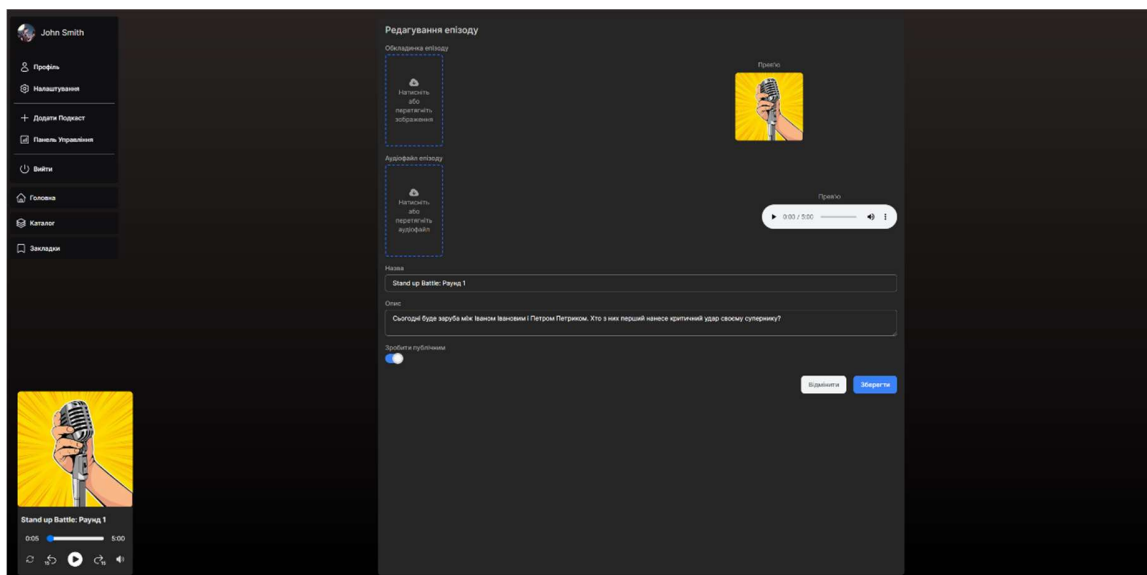


Рис. 3.37. Сторінка редагування епізоду

Як і на попередніх сторінках редагування, тут також є можливість переглядати прев'ю медіафайлів. Крім того, користувач має змогу налаштовувати публічність епізоду, визначаючи, чи можуть користувачі його переглядати.

Перейдемо до функціоналу адміністратора. Можливості адміністраторі представлені на сторінці панелі управління. Панель управління поділяється на три вкладки: Подкасти, Епізоди та Користувачі. Почнемо з вкладки подкастів (рис. 3.38).

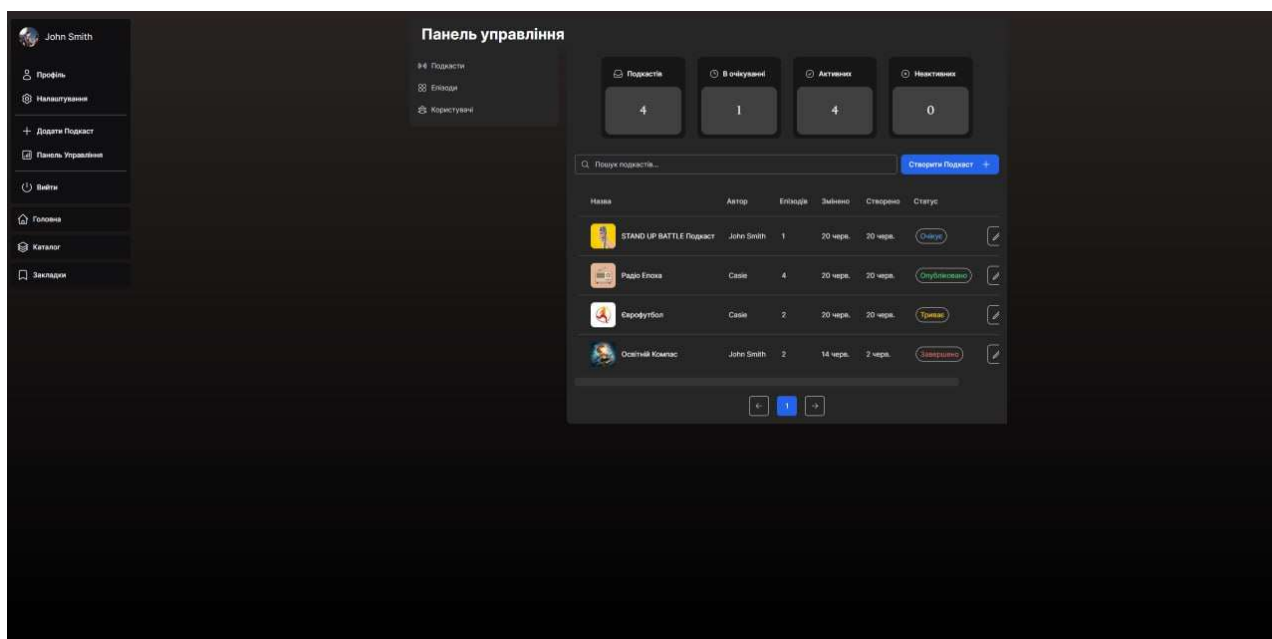


Рис. 3.38. Сторінка панелі управління – Подкасти

На вкладці подкастів адміністратор може переглянути короткий огляд подкастів у вигляді карток, де вказані: загальна кількість подкастів, кількість подкастів в очікуванні, кількість активних і неактивних подкастів. Нижче розташована таблиця всіх подкастів з функцією пошуку та кнопкою для додавання нового подкасту. Таблиця містить списки подкастів із такими даними: обкладинка, назва, автор, кількість епізодів, дата створення, дата останньої зміни, статус та кнопки керування. Для оптимізації запитів, таблиця може містити максимум 6 записів на сторінку.

Наступна вкладка Епізоди містить, як зрозуміло з назви, інформацію про епізоди (рис. 3.39).

Вкладка користувачів, подібно до попередніх, містить огляд у вигляді карток, де вказані загальна кількість користувачів, кількість нових користувачів за останній місяць, а також кількість адміністраторів і творців контенту (рис. 3.40). Нижче розташована таблиця з усіма користувачами, яка має функцію пошуку. У таблиці наведені такі дані: аватар користувача, ім'я, електронна пошта, дата народження, роль, гендер/стать, дата створення, дата останньої зміни та кнопки для керування.

Ми обговорили всі ключові дії та основний функціонал додатку, але є один важливий момент, який ми пропустили – адаптивність інтерфейсу. Вийдемо з облікового запису та подивімося, як виглядає стартова сторінка на смартфоні (рис. 3.41).

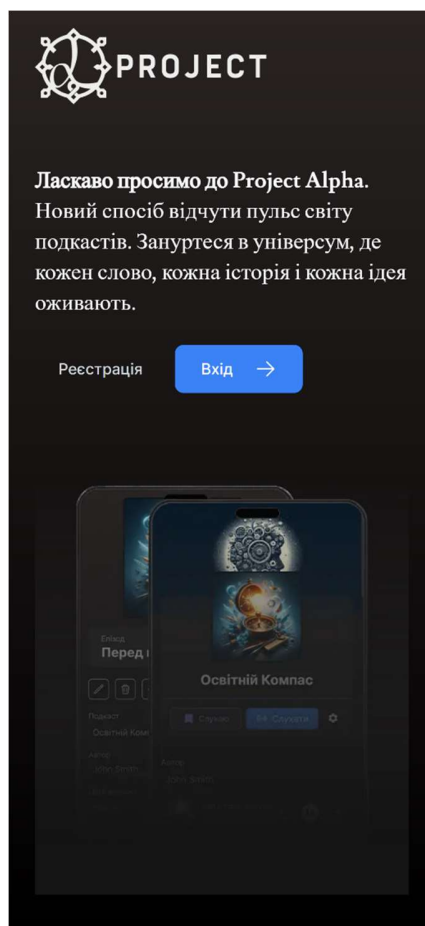


Рис. 3.41. Стартова сторінка на мобільних девайсах

Для мобільних девайсів відображається своя картинка з інтерфейсом додатку. Перейдемо до перегляду інтерфейсу у самому додатку (рис. 3.42).

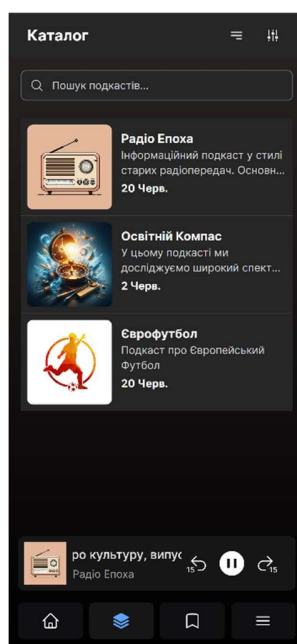


Рис. 3.42. Інтерфейс для мобільного пристрою

Як можна побачити, навігаційна панель перемістилася на низ для комфортного керування додатком. При відтворенні аудіо з'являється зручний плеєр за допомогою якого можна керувати потоком аудіо. Цей плеєр відображається у згорнутому вигляді. У ньому є можливість перемотування вперед та назад на 15 секунд та кнопка відтворення/паузи. Для розгортання плеєра треба натиснути на зону обкладинки, назви епізоду та назви подкасту (рис. 3.43).

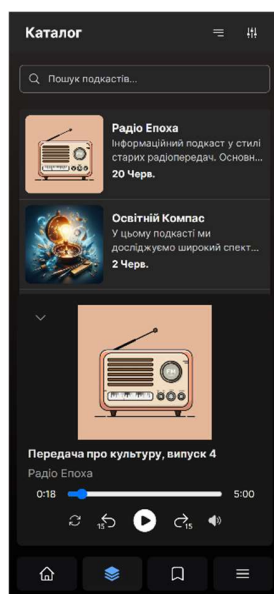


Рис. 3.43. Вигляд розгорнутого плеєра

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						73

Після розгортання з'являється більше можливостей для керування аудіо. Користувач тепер може увімкнути зациклення аудіо, регулювати гучність та перемотувати за допомогою доріжки прогресу.

При натисканні на значок праворуч на навігаційній панелі відкривається меню, де користувач може перейти до свого профілю, налаштувань та виходу з облікового запису (рис. 3.44). Також є кнопки для переходу на створення подкасту та панелі управління, в залежності від ролі користувача.

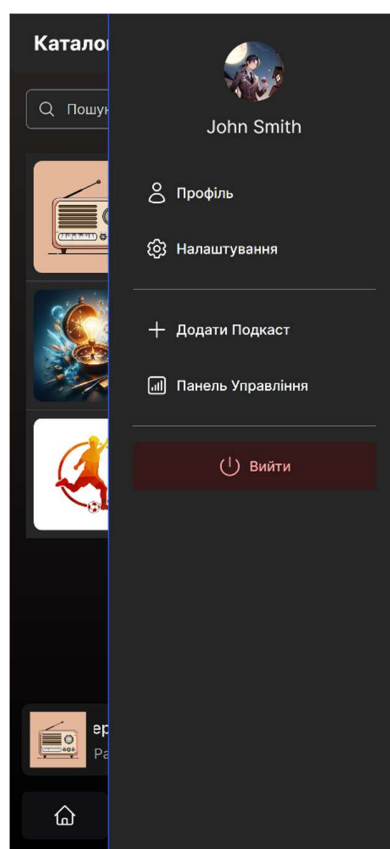


Рис. 3.44. Вигляд меню на мобільних пристроях

Щоб закрити меню треба натиснути на місце поза блоком меню.

Також було змінено вигляд деяких сторінок для зручного та інтуїтивного використання на мобільних пристроях.

Для прикладу візьмемо сторінку профілю користувача (рис. 3.45).

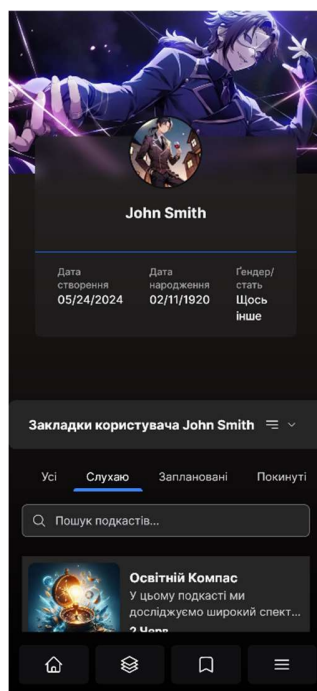


Рис. 3.45. Сторінка профілю користувача на мобільних пристроях

Окрім візуальних змін, були перенесені списки закладок у горизонтальне положення над списком для більш зручного та інтуїтивного користувацького досвіду.

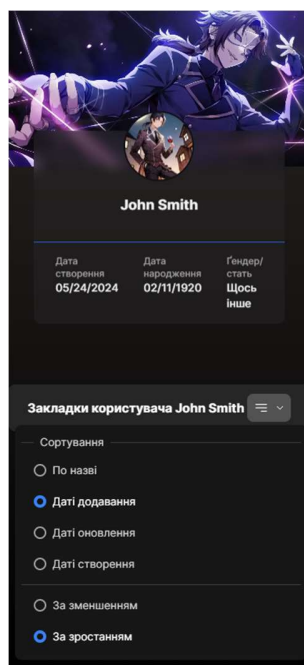


Рис. 3.46. Вигляд меню сортування

При натисканні на кнопку сортування, відкриється меню, де користувач може обрати спосіб сортування (рис. 3.46).

Вигляд таблиць в панелі адміністратора теж був змінений під інтерфейс телефонів (рис. 3.47).

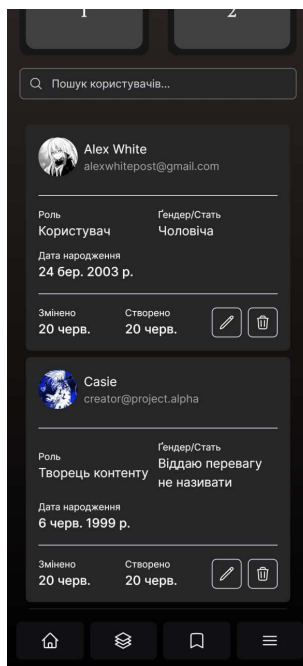


Рис. 3.47. Вигляд таблиці користувачів на мобільних девайсах

Тепер записи користувачів виглядають більш комфортно та зрозуміло для користувача.

3.3. Тестування роботи веб-додатку

Після завершення основних етапів розробки веб-додатку для прослуховування подкастів, наступним важливим кроком є проведення ретельного тестування. Тестування дозволяє виявити можливі помилки, оцінити продуктивність та забезпечити стабільну роботу додатку під час його використання кінцевими користувачами.

При розробці додатку були використані юніт-тести для знаходження помилок. Юніт-тестування — це перший рівень тестування в життєвому циклі розробки програмного забезпечення. Це автоматизовані тести на рівні коду, які пишуть і виконують розробники програмного забезпечення для перевірки окремих

частин програми, що називаються юнітами. Ці тестування дозволили нам перевірити правильність роботи окремих модулів і компонентів додатку та допомогли виявити помилки на ранніх етапах розробки, забезпечивши впевненість у тому, що кожна частина системи працює коректно [26].

Тест-кейс є документом, який складається з послідовності кроків для перевірки конкретної функціональності програмного продукту. Його ціль полягає у тому, щоб забезпечити систематичну перевірку програмного забезпечення та забезпечити досягнення очікуваного результату. [27].

Під час створення функціоналу додатка було підготовлено набір тест-кейсів, які використовуються для перевірки правильності роботи основних функцій програми.

1. Авторизація користувача

Передумови: Користувач вже має зареєстрований обліковий запис та дійсні облікові дані (логін і пароль).

Кроки виконання:

1. Відкрити сторінку авторизації;
2. Ввести електронну пошту і пароль;
3. Натиснути кнопку «Увійти».

Очікуваний результат: Користувач успішно авторизується і потрапляє на головну сторінку додатку.

Фактичний результат: Користувач успішно авторизується і потрапляє на головну сторінку додатку.

2. Створення нового подкасту

Передумови: Користувач авторизований в системі під обліковим записом творця контенту або адміністратора.

Кроки виконання:

1. Відкрити сторінку створення подкасту;

2. Заповнити необхідні дані (назву, опис, обкладинку, банер, категорії, теги);

3. Натиснути кнопку "Створити подкаст".

Очікуваний результат: Подкаст успішно створений і має статус «Очікує на перевірку».

Фактичний результат: Подкаст успішно створений і має статус «Очікує на перевірку».

3. Створення нового епізоду

Передумови: Користувач авторизований в системі під обліковим записом творця контенту або адміністратора.

Кроки виконання:

1. Відкрити сторінку налаштувань подкасту;

2. Перейти на вкладку «Епізоди»;

3. Натиснути на кнопку «Додати епізод»;

4. Відкрити сторінку створення епізоду;

5. Заповнити необхідні дані (назву, опис, обкладинку, аудіофайл);

6. Натиснути кнопку "Створити епізод".

Очікуваний результат: Епізод успішно створений і відображається у списку епізодів на сторінці подкасту.

Фактичний результат: Епізод успішно створений і відображається у списку епізодів на сторінці подкасту.

4. Зміна аватару та банера користувача

Передумови: Користувач авторизований в системі.

Кроки виконання:

1. Відкрити сторінку налаштувань користувача;

2. Обрати нове зображення аватару та банера;

3. Натиснути на кнопку «Зберегти»;

Очікуваний результат: Аватар або банер успішно змінені і відображаються на сторінці профілю.

Фактичний результат: Аватар або банер успішно змінені і відображаються на сторінці профілю.

5. Відтворення подкасту

Передумови: Користувач авторизований в системі.

Кроки виконання:

1. Відкрити сторінку будь-якого подкасту;
2. Натиснути кнопку «Слухати»;

Очікуваний результат: Подкаст успішно відтворюється у вбудованому плеєрі.

Фактичний результат: Подкаст успішно відтворюється у вбудованому плеєрі.

6. Додавання подкасту до списку закладок «Слухаю»

Передумови: Користувач авторизований в системі.

Кроки виконання:

1. Відкрити сторінку будь-якого подкасту;
2. Натиснути кнопку «Додати в»;
3. Вибрати «Слухаю» у списку, що з'явився.

Очікуваний результат: Подкаст успішно додається до списку закладок користувача, кнопка змінюється на «Слухаю», а на сторінці закладок у списку «Слухаю» відображається доданий подкаст.

Фактичний результат: Подкаст успішно додається до списку закладок користувача, кнопка змінюється на «Слухаю», а на сторінці закладок у списку «Слухаю» відображається доданий подкаст.

7. Видалення подкасту із закладок

Передумови: Користувач авторизований в системі і має подкаст збережений в закладках.

Кроки виконання:

1. Відкрити сторінку збереженого в закладках подкасту;
2. Натиснути кнопку з назвою списку в якому збережений подкаст;
3. Вибрати «Видалити зі списку» у списку, що з'явився.

Очікуваний результат: Подкаст успішно видаляється зі списку закладок користувача і кнопка змінюється на «Додати в».

Фактичний результат: Подкаст успішно видаляється зі списку закладок користувача і кнопка змінюється на «Додати в».

Тестування веб-додатку для прослуховування подкастів показало, що основні функціональні можливості додатку працюють коректно. Під час тестування не було виявлено жодних недоліків чи помилок у роботі.

Висновок до третього розділу

У третьому розділі було охоплено ключові аспекти, пов'язані з розробкою, налаштуванням, структурою інтерфейсу та тестуванням веб-додатку для прослуховування подкастів. Було детально розглянуто порядок встановлення та налаштування параметрів додатку. Також було побудовано діаграму розгортання, яка ілюструє взаємодію між клієнтськими пристроями, веб-сервером та сервером бази даних.

Було описано основні компоненти інтерфейсу. Особливу увагу приділено зручності використання та інтуїтивності навігації для забезпечення найкращого досвіду користувачів.

Додаток був протестований для виявлення потенційних помилок. Описано різні тест-кейси для перевірки ключових функціональних можливостей додатку, таких як авторизація користувачів, створення подкастів та епізодів, зміна аватару та банеру користувача, відтворення подкастів та управління закладками. Також було зазначено, що при розробці були використані юніт-тести, які допомогли виявити помилки на ранніх етапах та забезпечити високу якість коду.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було розроблено додаток для прослуховування подкастів. Мета роботи полягала у розробці інтуїтивно зрозумілого, продуктивного та безпечного веб-додатка, який задовольнить сучасні потреби користувачів у прослуховуванні та управлінні подкастами.

Аналіз існуючих аналогів допоміг визначити ключові функції та переваги, які мають бути реалізовані. Це дозволило розробити додаток, що може бути конкурентоспроможним на ринку. Вибір технологій був зроблений з урахуванням отриманих вимог та критеріїв. Next.js використовується як для клієнтської, так і для серверної частин додатку. TypeScript забезпечує високу продуктивність і надійність коду, а Tailwind CSS — адаптивність і зручність стилізації. Використання Next.js разом з Vercel Postgres для бази даних гарантує надійність і масштабованість системи, необхідні для обробки великих обсягів даних та високого навантаження.

Оскільки ефективне функціонування системи для прослуховування подкастів залежить від ретельно спроектованих та реалізованих алгоритмів, які забезпечують її основні можливості, було розроблено основний алгоритм роботи системи, а також створено діаграми активності для користувачів та адміністраторів. Веб-додаток був повністю спроектований та реалізований з урахуванням усіх ключових функцій, вимог до продуктивності, безпеки і зручності використання. Також було приділено особливу увагу оптимізації роботи з базою даних, швидкості обробки запитів та надійності механізмів кешування. Було визначено основні категорії користувачів додатка: звичайні користувачі, творці контенту та адміністратори. Кожна з цих категорій має специфічний набір функцій. Звичайні користувачі можуть переглядати, шукати та слухати подкасти, творці контенту мають можливість додавати нові подкасти та епізоди, а адміністратори керують контентом і користувачами через панель адміністратора. Завдяки цьому, було забезпечено різноманітні можливості взаємодії з системою для різних типів користувачів.

Розробка бази даних системи забезпечила надійне зберігання даних та швидкий доступ до них. Було створено таблиці для збереження користувачів, подкастів, епізодів, категорій, тегів та закладок, а також зв'язків між подкастами та категоріями і тегами.

Для забезпечення найкращого досвіду користувачів була створена зручна та інтуїтивно зрозуміла структура інтерфейсу, яка адаптується до різних типів пристроїв та розмірів екранів. Дизайн інтерфейсу додатку був ретельно продуманий з метою забезпечення зручності взаємодії користувачів з основними функціями, зокрема навігацією, пошуком контенту та керуванням особистим профілем. Додаток був протестований для виявлення потенційних помилок. Використання юніт-тестів допомогло виявити помилки на ранніх етапах розробки та забезпечити високу якість коду.

Додаток для прослуховування подкастів дозволяє звичайним користувачам зручно слухати та організовувати свої улюблені подкасти. Творцям контенту він надає інструменти для публікації своїх програм і взаємодії з аудиторією через відкриті функції платформи. Адміністратори отримують повний контроль над змістом та управлінням користувачами, що дозволяє ефективно відповідати на поточні потреби спільноти. Також додаток має міцну основу для майбутніх оновлень та розвитку на ринку медіа-платформ.

Джерела інформації

1. Подкаст. Що це і для чого [Електронний ресурс] – Режим доступу до ресурсу: <https://urbanspaceradio.com/archives/events/podkast-shho-ce-i-dlja-chogo>
2. Spotify [Електронний ресурс] – Режим доступу до ресурсу: <https://open.spotify.com/>.
3. Apple Podcasts [Електронний ресурс] – Режим доступу до ресурсу: <https://podcasts.apple.com/us/>.
4. Google Podcasts [Електронний ресурс] – Режим доступу до ресурсу: <https://podcasts.google.com/>.
5. Overcast [Електронний ресурс] – Режим доступу до ресурсу: <https://overcast.fm/>.
6. What is Client-Server Architecture? Everything You Should Know [Електронний ресурс] – Режим доступу до ресурсу: <https://www.simplilearn.com/what-is-client-server-architecture-article>.
7. Next.js [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org/>.
8. TypeScript [Електронний ресурс] – Режим доступу до ресурсу: <https://www.typescriptlang.org/>.
9. Tailwind CSS [Електронний ресурс] – Режим доступу до ресурсу: <https://tailwindcss.com/>.
10. Node.js® [Електронний ресурс] – Режим доступу до ресурсу: <https://nodejs.org/uk/>.
11. Vercel Postgres [Електронний ресурс] – Режим доступу до ресурсу: <https://vercel.com/docs/storage/vercel-postgres/>.
12. PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: <https://www.postgresql.org/>.
13. UML Use Case Diagram Tutorial [Електронний ресурс] – Режим доступу до ресурсу: <https://www.lucidchart.com/pages/uml-use-case-diagram>.

14. An introduction to the Unified Modeling Language [Электронный ресурс] – Режим доступа до ресурсу:
<https://developer.ibm.com/articles/an-introduction-to-uml/>.
15. What is Class Diagram [Электронный ресурс] – Режим доступа до ресурсу:
<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/>.
16. Creating Layouts and Pages [Электронный ресурс] – Режим доступа до ресурсу: <https://nextjs.org/learn/dashboard-app/creating-layouts-and-pages>.
17. JSON Web Tokens [Электронный ресурс] – Режим доступа до ресурсу:
<https://auth0.com/docs/secure/tokens/json-web-tokens>.
18. Partial Rendering in NextJs [Электронный ресурс] – Режим доступа до ресурсу: <https://webkul.com/blog/partial-rendering-in-nextjs/>.
19. A guide to enabling partial pre-rendering in Next.js[Электронный ресурс] – Режим доступа до ресурсу:
<https://blog.logrocket.com/guide-enabling-partial-pre-rendering-next-js>
20. A Deep Dive into Static vs Dynamic Rendering with NextJS [Электронный ресурс] – Режим доступа до ресурсу:
<https://themobilereality.com/blog/static-vs-dynamic-rendering>.
21. Zod [Электронный ресурс] – Режим доступа до ресурсу:
<https://zod.dev/>
22. Middleware in Next.js [Электронный ресурс] – Режим доступа до ресурсу:
<https://medium.com/@zachshallbetter/middleware-in-next-js-a-comprehensive-guide-7dd0a928541a>.
23. Deployment Diagram in UML [Электронный ресурс] – Режим доступа до ресурсу:

<https://www.geeksforgeeks.org/deployment-diagram-unified-modeling-languageuml/>.

24. What's the Difference Between HTTP and HTTPS [Электронный ресурс]
– Режим доступа до ресурсу:

<https://aws.amazon.com/compare/the-difference-between-https-and-http/>.

25. useDebounce [Электронный ресурс] – Режим доступа до ресурсу:

<https://usehooks.com/usedebounce>.

26. Правила оформления теми тест-кейса [Электронный ресурс] – Режим
доступу до ресурсу:

<https://training.qatestlab.com/blog/course-materials/test-case-topic/>.

27. What is Unit Testing? – A useful introduction [Электронный ресурс] –
Режим доступа до ресурсу: <https://qa.world/what-is-unit-testing/>.

ДОДАТКИ

					ПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						86

Лістинг @/app/lib/actions/auth.ts:

```

export async function signUp(state: UserFormState, formData: FormData) {
  const validatedFields = SignupFormSchema.safeParse({
    email: formData.get("email"),
    password: formData.get("password"),
    username: formData.get("username"),
    birthday_day: Number(formData.get("birthday_day")),
    birthday_month: Number(formData.get("birthday_month")),
    birthday_year: Number(formData.get("birthday_year")),
    gender: formData.get("gender"),
  });

  if (!validatedFields.success) {
    return {
      errors: validatedFields.error.flatten().fieldErrors,
      message: "Missing Fields. Failed to Create User.",
    };
  }

  const { email, password, username, birthday_day, birthday_month,
    birthday_year, gender } = validatedFields.data;

  const user = await getUserByEmail(email);
  if (user) {
    return {
      message: "This email address already in use.",
    };
  }

  const birthday_date = `${birthday_year}-${birthday_month}-${birthday_day}`;
  const hashedPassword = await bcrypt.hash(password, 10);
  try {
    const result = await sql`
      INSERT INTO users (email, password, username, birthday_date, gender)
      VALUES (${email.toLowerCase()}, ${hashedPassword}, ${username},
        ${birthday_date}, ${gender})
      RETURNING id, role;
    `;

    const user = result.rows[0];

    if (!user) {
      return {
        message: "Під час створення вашого облікового запису виникла помилка.",
      };
    }

    await createSession({ userId: user.id, role: user.role, name: username,
      avatar_url: null });
  } catch (error) {

```

```

        return {
            message: "Database Error: Failed to create user.",
            error: error,
        };
    }

    revalidatePath("/");
    redirect("/p/");
}

export async function signIn(state: { message: string } | undefined, formData:
FormData) {
    const validatedFields = SigninFormSchema.safeParse({
        email: formData.get("email"),
        password: formData.get("password"),
    });

    if (!validatedFields.success) {
        return {
            message: "Неправильний логін або пароль.",
        };
    }

    const { email, password } = validatedFields.data;

    const user = await getUserByEmail(email.toLowerCase());

    if (!user) {
        return {
            message: "Неправильний логін або пароль.",
        };
    }

    const passwordsMatch = await bcrypt.compare(password, user.password);

    if (passwordsMatch)
        await createSession({ userId: user.id, role: user.role, name:
user.username, avatar_url: user.avatar_url });
    else
        return {
            message: "Неправильний логін або пароль.",
        };

    revalidatePath("/");
    redirect("/p/");
}

export async function signOut() {
    await deleteSession();
    revalidatePath("/");
    redirect("/");
}

```

Лістинг @/app/lib/data.ts:

```
export async function getFilteredPodcasts(
  query: string,
  sort_by: string,
  order: string,
  categories: string[],
  tags: string[],
  age: string[],
  status: string[],
  year_max: string,
  year_min: string,
  episode_max: string,
  episode_min: string
) {
  noStore();

  const whereClauses: string[] = ["Podcasts.is_active = true",
  "Podcasts.status != 'pending'"];
  const values: any[] = [];

  if (query) {
    whereClauses.push(
      `(Podcasts.title ILIKE '%' || ${values.length + 1} || '%' OR
Podcasts.description ILIKE '%' || ${values.length + 1}
      ) || '%'`
    );
    values.push(`%${query}%`);
  }

  if (categories.length > 0) {
    whereClauses.push(`Podcasts.id IN (
      SELECT podcast_id
      FROM podcastcategories
      JOIN Categories
      ON podcastcategories.category_id = Categories.id
      WHERE Categories.name IN (${categories
        .map((_, i) => `${i + values.length + 1}`)
        .join(", ")})
      )`);
    values.push(...categories);
  }

  if (tags.length > 0) {
    whereClauses.push(`Podcasts.id IN (
      SELECT podcast_id
      FROM podcastTags
      JOIN Tags
      ON podcasttags.tag_id = Tags.id
      WHERE Tags.name IN (${tags.map((_, i) => `${i +
values.length + 1}`)
        .join(", ")})
      )`);
    values.push(...tags);
  }
}
```

```

        if (age.length > 0) {
            whereClauses.push(`Podcasts.age_rating IN (${age.map((_, i) => `$$${i +
values.length + 1}`.join(", ")}))`);
            values.push(...age);
        }

        if (status.length > 0) {
            whereClauses.push(`Podcasts.status IN (${status.map((_, i) => `$$${i +
values.length + 1}`.join(", ")}))`);
            values.push(...status);
        }

        if (year_min) {
            whereClauses.push(`Podcasts.created_at >= $$${values.length + 1}`);
            values.push(new Date(year_min));
        }

        if (year_max) {
            whereClauses.push(`Podcasts.created_at <= $$${values.length + 1}`);
            values.push(new Date(year_max));
        }

        if (episode_min || episode_max) {
            whereClauses.push(`Podcasts.id IN (
                SELECT podcast_id
                FROM (
                    SELECT podcast_id, COUNT(*) as episode_count
                    FROM Episodes
                    GROUP BY podcast_id
                ) as episode_counts
                WHERE
                    episode_counts.episode_count >= $$${values.length
+ 1} AND
                    episode_counts.episode_count <= $$${values.length
+ 2}
            )`);
            values.push(episode_min || 0, episode_max || Number.MAX_SAFE_INTEGER);
        }

        const whereClause = whereClauses.length > 0 ? whereClauses.join(" AND ") :
        "";

        let sortByClause;
        switch (sort_by) {
            case "title":
                sortByClause = "Podcasts.title";
                break;
            case "updated_at":
                sortByClause = "Podcasts.updated_at";
                break;
            case "created_at":
                sortByClause = "Podcasts.created_at";
                break;
            default:

```

```

        sortByClause = "Podcasts.title";
        break;
    }
    sortByClause += ` ${order}`;

    const queryString = `
        SELECT
            Podcasts.id,
            Podcasts.title,
            Podcasts.description,
            Podcasts.avatar_url,
            Podcasts.banner_url,
            Podcasts.author_id,
            Podcasts.is_active,
            Podcasts.status,
            Podcasts.age_rating,
            Podcasts.created_at,
            Podcasts.updated_at
        FROM
            Podcasts
        WHERE
            ${whereClause}
        ORDER BY
            ${sortByClause}
    `;

    try {
        const data = await sql.query(queryString, values);
        return data.rows;
    } catch (error) {
        console.error("Database Error:", error);
        throw new Error("Failed to fetch podcasts.");
    }
}

```

Лістинг @/app/utils/hooks/usePlayer.ts:

```

interface UsePlayer {
    isPlaying: boolean;
    isLooping: boolean;
    playAudio: () => void;
    pauseAudio: () => void;
    loopAudio: () => void;
    changeVolume: (volume: number) => void;
    skipForward: () => void;
    skipBackward: () => void;
    setEpisode: (episode: ExtendedEpisode) => void;
}

const usePlayer = (): UsePlayer => {
    const { audioRef } = useAudioRef();
    const [isPlaying, setIsPlaying] = useState(false);
    const [isLooping, setIsLooping] = useState(false);
    const { state, dispatch } = usePodcastContext();

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						91

```

const episode = state.episode;

const playAudio = useCallback(() => {
  audioRef.current?.play();
  setIsPlaying(true);
}, [audioRef]);

const pauseAudio = useCallback(() => {
  audioRef.current?.pause();
  setIsPlaying(false);
}, [audioRef]);

const loopAudio = useCallback(() => {
  if (audioRef.current) {
    audioRef.current.loop = !audioRef.current.loop;
    setIsLooping(audioRef.current.loop);
  }
}, [audioRef]);

const changeVolume = useCallback(
  (volume: number) => {
    if (audioRef.current) {
      audioRef.current.volume = volume;
    }
  },
  [audioRef]
);

const skipForward = useCallback(() => {
  if (audioRef.current) {
    audioRef.current.currentTime += 15;
  }
}, [audioRef]);

const skipBackward = useCallback(() => {
  if (audioRef.current) {
    audioRef.current.currentTime -= 15;
  }
}, [audioRef]);

useEffect(() => {
  const handleEnded = () => {
    setIsPlaying(false);
  };

  const currAudioRef = audioRef.current;
  currAudioRef?.addEventListener("ended", handleEnded);

  return () => {
    currAudioRef?.removeEventListener("ended", handleEnded);
  };
}, [audioRef]);

useEffect(() => {
  if (episode && audioRef.current) {

```

```

const src =
`/assets/podcasts/${episode.podcast_id}/${episode.id}/${episode.audio_url}`;
audioRef.current.src = src;
playAudio();
}
// eslint-disable-next-line react-hooks/exhaustive-deps
}, [episode]);

const setEpisode = useCallback(
  (episode: ExtendedEpisode) => {
    dispatch({ type: "SET_EPISODE", payload: episode });
  },
  [dispatch]
);

return {
  isPlaying,
  isLooping,
  playAudio,
  pauseAudio,
  loopAudio,
  changeVolume,
  skipForward,
  skipBackward,
  setEpisode,
};
};

export default usePlayer;

```

Лістинг @/app/lib/actions/mainActions.ts:

```

export async function createPodcast(state: PodcastFormState, formData:
FormData) {
  // validation of the form data
  const validatedFields = CreatePodcast.safeParse({
    title: formData.get("title"),
    description: formData.get("description"),
    avatar: formData.get("avatar"),
    banner: formData.get("banner") === "" ? null : formData.get("banner"),
    age_rating: formData.get("age_rating"),
  });

  if (!validatedFields.success) {
    return {
      errors: validatedFields.error.flatten().fieldErrors,
      message: "Відсутні обов'язкові поля. Створення подкасту не
вдалося.",
    };
  }

  const selectedCategories = JSON.parse(formData.get("categories") as
string) || [];

```

```

const selectedTags = JSON.parse(formData.get("tags") as string) || [];

const [categories, tags] = await Promise.all([fetchCategories(),
fetchTags()]);

const selectedCategoriesId = categories
    .filter((category) => selectedCategories.includes(category.name))
    .map((category) => ({ category_id: category.id }));

const selectedTagsId = tags.filter((tag) =>
selectedTags.includes(tag.name)).map((tag) => ({ tag_id: tag.id }));

// Getting data from validated fields
const { title, description, avatar, banner, age_rating } =
validatedFields.data;

// Processing the image data
const avatar_url = `${v4()}.${avatar.name.split(".").pop()}`;
const banner_url = banner ? `${v4()}.${banner.name.split(".").pop()}` :
null;

const session = await getSession();
const userId = session?.userId;

const result = await sql`
    INSERT INTO podcasts (title, description, avatar_url,
banner_url, age_rating, author_id)
    VALUES (${title}, ${description}, ${avatar_url}, ${banner_url},
${age_rating}, ${userId})
    RETURNING id
`;
const podcast = result.rows[0];
if (!podcast) {
    return {
        message: "Під час створення нового подкасту виникла помилка.",
    };
}

await saveFile(avatar_url, podcast.id, avatar);
banner_url && (await saveFile(banner_url, podcast.id, banner));

const categoriesJson = JSON.stringify(selectedCategoriesId);
const tagsJson = JSON.stringify(selectedTagsId);

await sql
    .query(
        `INSERT INTO podcastCategories (podcast_id, category_id)
        SELECT $1, category_id
        FROM json_populate_recordset(NULL::podcastCategories,
$2::json)`,
        [podcast.id, categoriesJson]
    )
    .catch((err) => {
        console.log(err);
    });

```

```

        await sql
            .query(
                `INSERT INTO podcastTags (podcast_id, tag_id)
                 SELECT $1, tag_id
                 FROM json_populate_recordset(NULL::podcastTags, $2::json)`,
                [podcast.id, tagsJson]
            )
            .catch((err) => {
                console.log(err);
            });

        revalidatePath("/p/podcast/");

        redirect(`/p/podcast/${podcast.id}`);
    }

export async function editPodcast(state: PodcastFormState, formData: FormData) {
    const action = formData.get("action") as string;
    const id = formData.get("id") as string;

    if (!id || !validate(id)) {
        return { message: "Неправильний ідентифікатор подкасту." };
    }

    try {
        const podcast = await fetchPodcast(id);

        if (!podcast) {
            return { message: "Подкаст не знайдено." };
        }

        const updated_at = new Date().toISOString().split("T")[0];

        if (action === "update-main-data") {
            const dataToValidate: { [key: string]: any } = {};
            formData.forEach((value, key) => {
                dataToValidate[key] = value;
            });

            const UpdatePodcast = PodcastFormSchema.pick({ title: true,
description: true, age_rating: true });

```

```

const validatedFields = UpdatePodcast.safeParse(dataToValidate);

if (!validatedFields.success) {
  return {
    errors: validatedFields.error.flatten().fieldErrors,
    message: "Validation error. Please check your input.",
  };
}

const { title, description, age_rating } = validatedFields.data;

await sql`
  UPDATE podcasts
  SET title = ${title}, description = ${description}, age_rating
= ${age_rating}, updated_at = ${updated_at}
  WHERE id = ${id}
`;
} else if (action === "update-media") {
  const dataToValidate: { [key: string]: any } = {};
  formData.forEach((value, key) => {
    dataToValidate[key] = value;
  });
  const UpdatePodcast = PodcastFormSchema.pick({ avatar: true,
banner: true }).partial();

  const validatedFields = UpdatePodcast.safeParse(dataToValidate);

  if (!validatedFields.success) {
    return {
      errors: validatedFields.error.flatten().fieldErrors,
      message: "Validation error. Please check your input.",
    };
  }

  for (const [key, value] of Object.entries(validatedFields.data))
{

```

```

        console.log(key, value);
        if (key === "avatar") {
            const filePath =
`/assets/podcasts/${id}/${podcast.avatar_url}`;
            Promise.all([
                overwriteFile(filePath, value),
                sql`
                    UPDATE podcasts
                    SET updated_at = ${updated_at}
                    WHERE id = ${id}
                `,
            ]);
        } else if (key === "banner") {
            const banner_url = podcast.banner_url ||
`${v4()}.${value.name.split(".").pop()}`;
            const filePath = `/assets/podcasts/${id}/${banner_url}`;
            Promise.all([
                overwriteFile(filePath, value),
                sql`
                    UPDATE podcasts
                    SET updated_at = ${updated_at}, banner_url =
${banner_url}
                    WHERE id = ${id}
                `,
            ]);
        }
    }
} else if (action === "update-classification") {
    const [podcastCategories, podcastTags, categories, tags] = await
Promise.all([
        getPodcastCategories(id),
        getPodcastTags(id),
        fetchCategories(),
        fetchTags(),
    ]);

    const selectedCategories = categories

```

```

        .filter((category) => JSON.parse(formData.get("categories")
as string).includes(category.name))
        .map((category) => category.id);
        const currCategories = podcastCategories.map((category) =>
category.id);

        const categoriesToAdd = selectedCategories
        .filter((id) => !currCategories.includes(id))
        .map((category) => ({ category_id: category }));
        const categoriesToRemove = currCategories.filter((id) =>
!selectedCategories.includes(id));

        const selectedTags = tags
        .filter((tag) => JSON.parse(formData.get("tags") as
string).includes(tag.name))
        .map((tag) => tag.id);
        const currTags = podcastTags.map((tag) => tag.id);
        const tagsToAdd = selectedTags.filter((id) =>
!currTags.includes(id)).map((tag) => ({ tag_id: tag }));
        const tagsToRemove = currTags.filter((id) =>
!selectedTags.includes(id));

        if (categoriesToAdd.length > 0) {
            await sql.query(
                `INSERT INTO podcastCategories (podcast_id, category_id)
                SELECT $1, category_id
                FROM json_populate_recordset(NULL::podcastCategories,
$2::json)`,
                [id, JSON.stringify(categoriesToAdd)]
            );
        }
        if (categoriesToRemove.length > 0) {
            const placeholders = categoriesToRemove.map((_, index) =>
`$$${index + 2}`).join(", ");
            await sql.query(
                `DELETE FROM podcastCategories
                WHERE podcast id = $1

```

```

        AND category_id IN (${placeholders})`,
        [id, ...categoriesToRemove]
    );
}
if (tagsToAdd.length > 0) {
    await sql.query(
        `INSERT INTO podcastTags (podcast_id, tag_id)
        SELECT $1, tag_id
        FROM json_populate_recordset(NULL::podcastTags, $2::json)`,
        [id, JSON.stringify(tagsToAdd)]
    );
}
if (tagsToRemove.length > 0) {
    const placeholders = tagsToRemove.map((_, index) => `$$${index
+ 2}`).join(", ");

    await sql.query(
        `DELETE FROM podcastTags
        WHERE podcast_id = $1
        AND tag_id IN (${placeholders})`,
        [id, ...tagsToRemove]
    );
}
} else if (action === "update-management") {
    const dataToValidate: { [key: string]: any } = {
        is_active: (formData.get("is_active") as string) === "true",
        status: formData.get("status"),
    };

    const UpdatePodcast = PodcastFormSchema.pick({
        is_active: true,
        status: true,
    }).partial();

    const validatedFields = UpdatePodcast.safeParse(dataToValidate);

    if (!validatedFields.success) {

```

```

        return {
            errors: validatedFields.error.flatten().fieldErrors,
            message: "Validation error. Please check your input.",
        };
    }

    const { is_active, status } = validatedFields.data;

    await sql`
        UPDATE podcasts
        SET is_active = ${is_active}, status = ${status}, updated_at
= ${updated_at}
        WHERE id = ${id}
    `;
    } else {
        return { message: "Invalid action." };
    }
} catch (error) {
    console.error("Error updating podcast:", error);
    return {
        message: "An error occurred while updating the podcast.",
        error: error,
    };
}

revalidatePath(`/p/podcast/${id}`);
redirect(`/p/podcast/${id}`);
}

export async function deletePodcast(id: string) {
    try {
        await Promise.all([sql`DELETE FROM podcasts WHERE id = ${id}`,
removePodcastFiles(id)]);

        revalidatePath(`/p/podcast/`);
        return { message: "Подкаст видалено." };
    } catch (err) {
        return {
            message: "Database Error: Failed to delete podcast.",
            error: err,
        };
    }
}

```

```

    };
  }
}

export async function addToBookmarks(options: {
  user_id: string;
  podcast_id: string;
  list_type: Bookmarks["list_type"];
}) {
  const validatedFields = AddBookmarks.safeParse({
    list_type: options.list_type,
  });

  if (!validatedFields.success) {
    return {
      status: "error",
    };
  }

  const isBookmarked = await getUserBookmarkByPodcastId(options.podcast_id,
options.user_id);

  const { list_type } = validatedFields.data;

  if (isBookmarked === list_type) {
    return {
      status: "success",
      data: list_type,
    };
  }

  const updated_at = new Date().toISOString().split("T")[0];
  try {
    if (isBookmarked) {
      await sql`
        UPDATE bookmarks
        SET list_type = ${list_type}, updated_at = ${updated_at}
        WHERE user_id = ${options.user_id} AND podcast_id =
${options.podcast_id}
      `;
    } else {
      await sql`
        INSERT INTO bookmarks (user_id, podcast_id, list_type,
updated_at)
        VALUES (${options.user_id}, ${options.podcast_id},
${list_type}, ${updated_at})
        ON CONFLICT (user_id, podcast_id) DO NOTHING;
      `;
    }

    return {
      status: "success",
      data: list_type,
    };
  }
}

```

```

        } catch (error) {
            return {
                status: "error",
                message: "Database Error: Failed to add to bookmarks.",
                error: error,
            };
        }
    }

export async function deleteFromBookmarks(user_id: string, podcast_id: string)
{
    try {
        await sql`
            DELETE FROM bookmarks
            WHERE user_id = ${user_id} AND podcast_id = ${podcast_id}
        `;
    } catch (error) {
        return {
            status: "error",
            message: "Database Error: Failed to delete from bookmarks.",
            error: error,
        };
    }
}

```