МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії
програмного забезпечення

_____ Євген ДАВИДЕНКО

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ВЕБЗАСТОСУНОК ДИСТРИБУЦІЇ АУДІОКОНТЕНТУ

Спеціальність 121 Інженерія програмного забезпечення
Освітня програма «Інженерія програмного забезпечення»

Здобувач _____ **Владислав НЕДІЛЬКО**

«__» _____ 2026 р.

Керівник роботи
Ст. викладачка

_____ **Марина ФАЛЕНКОВА**

«__» _____ 2026 р.

Миколаїв – 2026

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп’ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступінь	Бакалавр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«___» _____ 2026 р.

ЗАВДАННЯ

на кваліфікаційну бакалаврську роботу здобувача

Неділько Владислав

1. Тема кваліфікаційної роботи «вебзастосунок дистрибуції аудіоконтенту» затверджена наказом ректора ЧНУ ім. Петра Могили № 349 від «26» грудня 2026 р.
2. Строк представлення кваліфікаційної роботи «___» _____ 2026 р.
3. Очікуваний результат роботи та початкові дані якщо такі потрібні.
4. Перелік питань, що підлягають розробці:

– проаналізувати сучасний стан та існуючі рішення у сфері вебдистрибуції аудіоконтенту;
– дослідити методи організації потокового передавання аудіоданих через вебсередовище;

– розробити структуру бази даних для зберігання аудіоконтенту та інформації про користувачів;
– реалізувати механізми автоматизації завантаження та відтворення аудіофайлів;
– забезпечити систему авторизації та розмежування доступу до контенту на основі ролей користувачів (гість, слухач, автор, адміністратор).

5. Перелік графічних матеріалів:

6. Консультанти:

Консультант	**Кафедра (організація)**	**Частина роботи**

Дата видачі завдання «05» __січня_____ 2026 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: Вебзастосунок дистрибуції аудіоконтенту

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Отримання завдання. Узгодження теми: «Вебдистрибуція аудіоконтенту». Формування мети та завдань роботи	26.12.2025	05.01.26	Виконано
2.	Аналіз сучасних платформ аудіодистрибуції. Дослідження архітектури SPA, REST та клієнт–серверної взаємодії	06.01.2026	20.01.2026	Виконано
3.	Проєктування архітектури вебзастосунку на основі фреймворку Next.js (React) та інтеграції BaaS	21.01.2026	05.02.2026	Виконано
4.	Налаштування хмарного середовища Firebase: конфігурація автентифікації користувачів, створення бази даних Firestore та Cloud Storage	06.02.2026	15.02.2026	Виконано
5.	Реалізація механізмів оптимізації завантаження, кешування та потокового передавання аудіофайлів з хмарного сховища	16.02.2026	05.03.2026	Виконано
6.	Створення клієнтської частини на Next.js: структура компонентів, сторінки авторизації, завантаження та відтворення аудіо	06.03.2026	20.03.2026	Виконано
7.	Стилізація UI з використанням Tailwind CSS: адаптивний дизайн, UI-компоненти, навігація	21.03.2026	05.04.2026	Виконано
8.	Інтеграція фронтенду з хмарними сервісами (Firebase SDK). Налагодження взаємодії та безпеки збереження аудіоконтенту	06.04.2026	15.04.2026	Виконано

9.	Підготовка та оформлення звіту з практики	11.04.2026	18.04.2026	Виконано
10	Відгук керівника КБР	19.04.2026	25.04.2026	Виконано
11	Оформлення КБР та презентації	26.04.2026	27.04.2026	Виконано
12	Попередній захист	25.05.2026	25.05.2026	Виконано
13	Завершення оформлення КБР та презентації	26.05.2026	10.06.2026	Виконано
14	Рецензування	12.06.2026	15.06.2026	Виконано
15	Захист кваліфікаційної роботи			

Здобувач _____ **Владислав НЕДІЛЬКО**

«__» _____ 2026 р.

Керівник роботи

Ст. викладачка

_____ **Марина ФАЛЕНКОВА**

«__» _____ 2026 р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

Вебзастосунок дистрибуції аудіоконтенту

Здобувач 408 гр.: Неділько Владислав

Керівник: ст. викладачка Фаленкова Марина

Дана робота присвячена розробці ПЗ для автоматизації процесу розповсюдження аудіоконтенту з підвищенням ефективності взаємодії користувача з медіасередовищем, ґрунтуючись на роботі з БД: аудіофайлів, метаданих виконавців та користувацьких колекцій.

Мета: автоматизація процесу доставки медіаконтенту користувачам за рахунок розробки масштабованого вебзастосунку для дистрибуції аудіоконтенту.

Об’єкт роботи: процес вебдистрибуції та потокового відтворення аудіоконтенту в сучасних інформаційних мережах.

Предмет роботи: програмні засоби створення вебзастосунку для управління, зберігання та дистрибуції аудіоматеріалів.

Кваліфікаційна робота складається із вступу, 4 розділів, висновків та переліку джерел посилання.

У вступі обґрунтовано актуальність теми, визначено об’єкт, предмет, мету та завдання дослідження.

У першому розділі проведено аналіз сучасних платформ вебдистрибуції аудіоконтенту, виявлено їхні переваги та недоліки, на основі чого сформульовано концепцію власної системи SoundFlex. Також досліджено методи потокового передавання даних у вебсередовищі та обґрунтовано вибір технологічного стеку розробки.

Другий розділ присвячено науково-технічному обґрунтуванню проєктних рішень. У ньому проведено огляд сучасних інструментаріїв і моделей розробки вебплатформ на основі актуальних наукових публікацій, проаналізовано методи оптимізації доступу до об’єктних сховищ даних, а також сформовано специфікацію функціональних і нефункціональних вимог до ПЗ.

У третьому розділі розроблено архітектуру вебзастосунку: обґрунтовано архітектурні рішення та шаблони проєктування, виконано моделювання бізнес-процесів і поведінки системи мовою UML, спроєктовано структуру бази даних та інтерфейс користувача.

У четвертому розділі детально описано програмну реалізацію клієнтської частини вебплатформи, включно з навігацією, модулями авторизації, управління медіаконтентом, а також функціоналом відтворення та завантаження аудіо. Проведено тестування працездатності системи, проаналізовано отримані результати та оцінено відповідність реалізованого функціоналу поставленим вимогам.

У висновках підведено підсумки виконаної роботи, підтверджено досягнення поставленої мети та окреслено перспективи подальшого розвитку розробленої вебплатформи.

Кваліфікаційна робота викладена на 79 сторінках машинописного тексту, складається із вступу, 4 розділів, загальних висновків, переліку джерел посилання з 20 найменувань та 3 додатків. Праця містить 10 таблиць та 21 рисунків.

Ключові слова: Вебдистрибуція, Аудіоконтент, Потокове відтворення, HLS, Стрімінговий сервіс, Вебплатформа, SoundFlex, Next.js, FB, Firebase, REST API, Автентифікація, Tailwind CSS, БД, Функціональні вимоги, Тестування.

ABSTRACT

to the qualifying bachelor's thesis

Audio content distribution web application

Student of 408 group: Nedilko Vladyslav

Supervisor: Senior lecturer, Falenkova Marina

This work is devoted to the development of software for automating the process of distributing audio content with increasing the efficiency of user interaction with the media environment, based on working with databases: audio files, artist metadata and user collections.

Goal: automating the process of delivering media content to users by developing a scalable web application for distributing audio content.

Object of work: the process of web distribution and streaming audio content in modern information networks.

Subject of work: software tools for creating a web application for managing, storing and distributing audio materials.

The qualification work consists of an introduction, 4 sections, conclusions and a list of reference sources.

The introduction substantiates the relevance of the topic, defines the object, subject, goal and objectives of the study.

In the first section, an analysis of modern platforms for web distribution of audio content is carried out, their advantages and disadvantages are identified, on the basis of which the concept of the SoundFlex system is formulated. Also, methods of streaming data in the web environment are investigated and the choice of the technological development stack is justified.

The second section is devoted to the scientific and technical justification of project solutions. It reviews modern tools and models for developing web platforms based on current scientific publications, analyzes methods for optimizing access to object data repositories, and also forms a specification of functional and non-functional requirements for software.

The third section develops the architecture of the web application: architectural decisions and design patterns are justified, business processes and system behaviour are modelled using UML, and the database structure and user interface are designed.

The fourth section describes in detail the software implementation of the client part of the web platform, including navigation, authorization modules, media content management, as well as audio playback and download functionality. The system's performance is tested, the results obtained are analyzed, and the compliance of the implemented functionality with the requirements is assessed.

The conclusions summarize the work done, confirm the achievement of the set goal, and outline the prospects for further development of the developed web platform.The qualification work is presented on 79 pages of typewritten text, consists of an introduction, 4 sections, general conclusions, a list of reference sources with 20 titles and 3 appendices. The work contains 10 tables and 21 figures.

Keywords: Web distribution, Audio content, Streaming, Streaming service, Web platform, *HLS,* SoundFlex, Next.js, Firebase, SPA, REST API, Authentication, Tailwind CSS, Database, Functional requirements, Testing.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – База Даних
HLS – HTTP Live Streaming
HTTP – HyperText Transfer Protocol
HTTPS – HyperText Transfer Protocol Secure
ISR – Incremental Static Regeneration
NFR – Non-Functional Requirement
SDK – Software Development Kit
SPA – Single Page Application
SQL – Structured Query Language
UI – User Interface

ВСТУП

У сучасному глобалізованому цифровому суспільстві аудіоконтент, що охоплює музичні твори, подкасти, аудіокниги та спеціалізовані освітні записи, трансформувався в один із найбільш динамічних та затребуваних сегментів медіапростору. Стрімка еволюція вебтехнологій, впровадження високоефективних хмарних обчислень та повсюдне поширення мобільних платформ створили умови для формування принципово нових підходів до генерації, довготривалого зберігання та глобальної дистрибуції мультимедійних даних через мережу Інтернет.

На сьогодні світовий ринок вебдистрибуції аудіоконтенту представлений такими технологічними гігантами, як Spotify, SoundCloud та YouTube Music. Ці платформи використовують передові технічні рішення, зокрема протоколи адаптивного потокового передавання (наприклад, HLS), складні алгоритми персоналізації на базі машинного навчання, розгалужені системи аналітики для правовласників та багаторівневі моделі монетизації (Freemium, Subscription).

Проте, попри високий рівень розвитку існуючих сервісів, перед розробниками нових систем постає низка критичних викликів:

- складність масштабування систем потокового передавання;
- забезпечення стабільності та швидкодії при великій кількості користувачів;
- ефективне зберігання великих обсягів мультимедійних даних;
- захист авторських прав;
- персоналізація рекомендацій.

Таким чином, тема вебдистрибуція аудіоконтенту є актуальною з огляду на постійне зростання обсягів цифрового медіа та потребу у створенні ефективних, масштабованих і безпечних вебсистем.

Метою роботи є удосконалення процесу вебдистрибуції аудіоконтенту шляхом розробки та впровадження вебплатформи для автоматизації завантаження, зберігання та потокового відтворення аудіофайлів.

Відповідно до мети визначено такі завдання:

- проаналізувати сучасний стан та існуючі рішення у сфері вебдистрибуція аудіоконтенту;
- дослідити методи організації потокового передавання аудіоданих через вебсередовище;
- обґрунтувати вибір архітектури та технологій реалізації системи;
- розробити структуру бази даних для зберігання аудіоконтенту та інформації про користувачів;
- реалізувати механізми автоматизації завантаження та відтворення аудіофайлів;
- забезпечити систему авторизації та управління доступом до контенту.

Об’єкт роботи: процес вебдистрибуції та потокового відтворення аудіоконтенту в сучасних інформаційних мережах.

Предмет роботи: програмні засоби створення вебзастосунку для управління, зберігання та дистрибуції аудіоматеріалів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Дослідження сфери вебдистрибуції аудіоконтенту

Аналіз наявних рішень у сфері вебдистрибуції аудіоконтенту дає змогу окреслити ключові очікування користувачів, поширені бізнес-моделі та технічні підходи, що вже довели свою ефективність. Вивчення лідерів ринку дозволяє не тільки адаптувати найкращі практики, а й виявити вузькі місця, які можна усунути у власному програмному продукті. Для дослідження обрано три потужні платформи, що охоплюють як масове HLS, так і нішу незалежних авторів: SoundCloud, Spotify та YouTube Music [1].

SoundCloud позиціонується насамперед як платформа для незалежних виконавців, діджеїв та подкастерів, дозволяючи їм завантажувати, поширювати й монетизувати власний контент. На відміну від багатьох конкурентів, SoundCloud зберігає соціальну складову: користувачі можуть коментувати окремі моменти треку, створювати плейлисти, підписуватися на артистів і обмінюватися повідомленнями. Технічна архітектура спирається на хмарну інфраструктуру, а клієнтська частина реалізована як прогресивний вебзастосунок, що забезпечує адаптивне відтворення в браузері без обов’язкового встановлення нативного клієнта.

Таблиця 1.1 – Опис платформи SoundCloud

Характеристика	Опис
Назва	SoundCloud
Архітектура	Багаторівнева веборієнтована система з мікросервісними компонентами
Виробник	SoundCloud Global Limited (Німеччина)
Мова реалізації	Ruby (Ruby on Rails), JavaScript, Scala, Go

Кінець таблиці 1.1

Функції	1. обмежене завантаження аудіо авторами; 2. hls у вебплеєрі з підтримкою адаптивного бітрейту; 3. коментування з прив'язкою до тайм-коду; 4. формування стрічки рекомендацій на основі вподобань; 5. монетизація для авторів через підписку soundcloud premier; 6. інтеграція з соціальними мережами.
Переваги	1. відкрита модель для незалежних авторів, без вимоги співпраці з лейблами; 2. унікальна функція коментування з часовими мітками; 3. багата бібліотека реміксів, подкастів і live-записів.
Недоліки	1. обмежений каталог мейнстрімної музики порівняно з гігантами галузі; 2. безкоштовна версія містить рекламу; 3. якість звуку залежить від оригінального завантаження, іноді нестабільна якість оригінальних файлів.

UI SoundCloud лаконічний, з акцентом на обкладинки треків та безперервне відтворення. Основна навігація охоплює домашню стрічку, розділ пошуку, бібліотеку користувача та особистий кабінет автора (рис. 1.1).

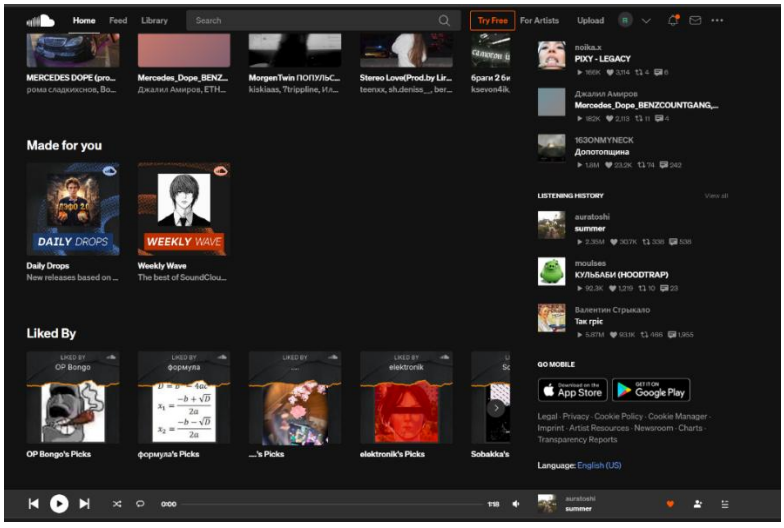

Рисунок 1.1 – Вигляд UI платформи SoundCloud

Spotify є одним із найбільших сервісів потокового аудіо у світі, пропонуючи доступ до десятків мільйонів треків, подкастів і аудіокниг. Ключова відмінність – розвинені алгоритми персоналізації, які формують

щотижневі добірки, радіо на основі настрою та автоматичні плейлисти. Бізнес-модель поєднує freemium-підхід (безкоштовний доступ з рекламою) і преміум-підписки з офлайн-режимом і вищою якістю звуку. Вебверсія Spotify реалізована з використанням сучасних фреймворків та підтримує синхронізацію з нативними застосунками на інших пристроях.

Таблиця 1.2 – Опис платформи Spotify

Характеристика	Опис
Назва	Spotify
Архітектура	Розподілена клієнт-серверна система з широким застосуванням хмарних сервісів
Виробник	Spotify AB (Швеція)
Мова реалізації	Java, Python, JavaScript (React), C++
Функції	1. необмежене потокове прослуховування музики та подкастів; 2. персоналізовані плейлисти (discover weekly, daily mix); 3. офлайн-збереження треків (преміум); 4. спільні плейлисти та групове прослуховування; 5. розумний пошук за текстом пісні, настроєм або контекстом; 6. підтримка високої якості аудіо (до 320 кбіт/с, планується lossless);
Переваги	1. величезний каталог ліцензованої музики; 2. ефективна система рекомендацій, що постійно навчається; 3. кросплатформеність і синхронізація між пристроями.

Оформлення UI Spotify тяжіє до мінімалізму та темної теми, що зменшує втому очей під час тривалого прослуховування (рис. 1.2). Головний екран одразу надає доступ до нещодавно відтвореного, рекомендованого й популярного контенту.

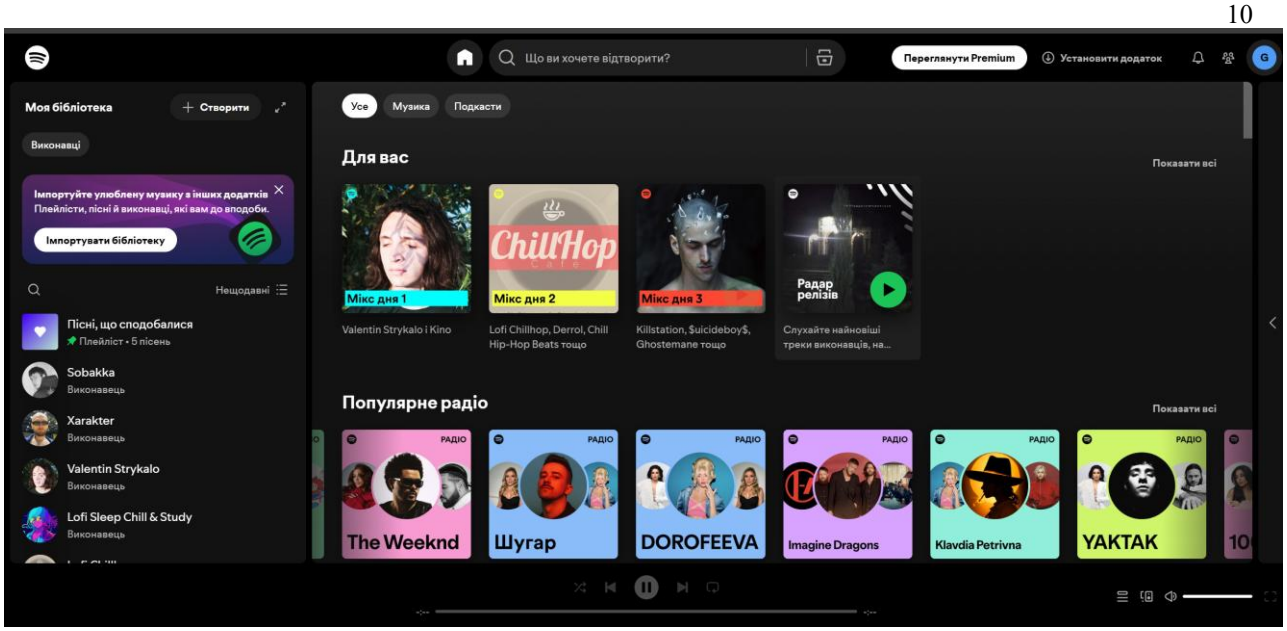

Рисунок 1.2 – Вигляд UI платформи Spotify

YouTube Music вирізняється інтеграцією з відеоплатформою YouTube, завдяки чому каталог містить не лише офіційні треки, а й безліч каверів, концертних записів і реміксів, відсутніх у традиційних музичних сервісах. Алгоритм рекомендацій використовує історію переглядів та прослуховувань обох платформ, що дає змогу пропонувати надзвичайно релевантний контент. Також доступна функція перемикання між відео та аудіо режимами.

Таблиця 1.3 – Опис платформи YouTube Music

Характеристика	Опис
Назва	YouTube Music
Архітектура	Багаторівнева система, тісно інтегрована з відеоплатформою YouTube
Виробник	Google LLC (США)
Мова реалізації	Java, Python, C++, JavaScript
Функції	1. hls музичних треків і відеокліпів; 2. перемикання між режимами «тільки аудіо» та відео; 3. розумні рекомендації на основі історії youtube; 4. фонове відтворення та завантаження (преміум); 5. інтеграція з google акаунтом і єдиною підпискою youtube premium; 6. текстовий та голосовий пошук із розпізнаванням контексту.

Кінець таблиці 1.3

Переваги	1. найширший каталог завдяки контенту, створеному користувачами; 2. висока точність рекомендацій через використання даних youtube; 3. можливість дивитися відео, що розширює XP споживання контенту.
Недоліки	1. безкоштовна версія відображає відеорекламу та не підтримує фонове відтворення; 2. ui місцями перевантажений візуальними елементами; 3. якість звуку у безкоштовній версії залежить від оригінального завантаження відео.

UI YouTube Music схожий на традиційний YouTube, але з акцентом на обкладинки альбомів та спрощену навігацію в стилі музичних сервісів (рис. 1.3). Основні розділи: Головна, Пошук та Бібліотека.

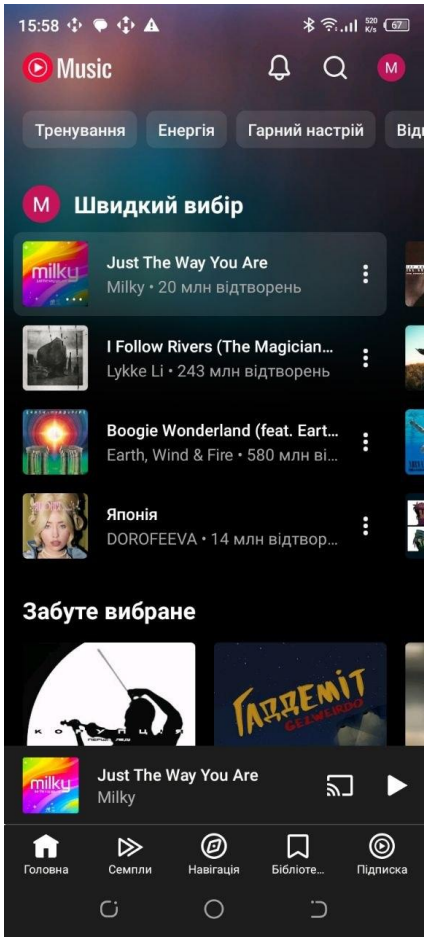

Рисунок 1.3 – Вигляд UI платформи YouTube Music

Проведений огляд демонструє, що сучасні платформи вебдистрибуції аудіоконтенту об’єднують HLS високої якості, персоналізовані рекомендації, соціальні інструменти та різноманітні моделі монетизації. Водночас жодна з них не є універсальним рішенням: незалежні автори часто стикаються з низькими роялті на Spotify чи обмеженою видимістю на YouTube Music, а слухачі – із суворими лімітами безкоштовних версій. Отримані результати будуть використані під час формування функціональних вимог та архітектурних рішень власної системи.

1.2 Аналіз системи, що розробляється

Беручи до уваги результати дослідження предметної області та обмеження наявних стрімінгових платформ, сформульовано концепцію власної вебплатформи під умовною назвою SoundFlex. Система створюється як високопродуктивний сервіс для незалежної дистрибуції аудіоконтенту, покликаний усунути зайвих посередників між автором та слухачем. Основний акцент зроблено на швидкодії, мінімалістичному UI та зручності управління персональною медіабібліотекою.

Технічну основу платформи становить фреймворк Next.js із використанням App Router, що забезпечує гібридний рендеринг, оптимізовану навігацію та високу продуктивність клієнтської частини. Для стилізації застосовано утилітарний CSS-фреймворк Tailwind CSS у поєднанні з бібліотекою готових компонентів (конфігурація бібліотеки shadcn/ui), що дозволяє витримувати єдиний дизайн-код за мінімальних витрат часу на верстку.

Серверна частина реалізована на базі хмарної платформи Firebase. Автентифікація користувачів покладається на Firebase Authentication, зберігання метаданих та потокових аудіофайлів – на Firebase Storage, а структуровані дані (профілі, плейлисти, інформація про треки) розміщуються в базі даних Firestore. Правила доступу до бази даних визначені у файлі

firestore.rules, що гарантує захист інформації на рівні серверної інфраструктури без необхідності розгортання окремого серверного API.

Внутрішня організація кодової бази передбачає чітке розділення відповідальностей. У директорії src/app зосереджено маршрути додатку, у src/components – повторно використовувані UI елементи, у src/hooks – логіка стану компонентів, а в src/lib – допоміжні функції для роботи з даними. Така модульна структура спрощує супроводження та масштабування системи.

Узагальнені характеристики платформи SoundFlex наведено в таблиці 1.4

Таблиця 1.4– Характеристика системи SoundFlex, що розробляється

Характеристика	Опис
Назва	SoundFlex
Архітектура	Односторінковий вебзастосунок (SPA) з клієнтським рендерингом та хмарним серверним бекендом (Firebase).
Виробник	Владислав Віталійович Неділько
Мова реалізації	TypeScript, JavaScript
Основні технології	Next.js, React, Tailwind CSS, shadcn/ui, Firebase (Auth, Firestore, Storage, Hosting)
Головна мета	Створення персоналізованого середовища для публікації та споживання цифрового аудіоконтенту «SoundFlex» з акцентом на продуктивність, мінімалізм і зручне управління персональною бібліотекою
Цільова аудиторія	1. автори: інді-виконавці, подкастери, які потребують простого інструменту для зберігання та публікації робіт; 2. слухачі: користувачі, які цінують чистий UI, високу швидкість роботи та прямий доступ до контенту без посередників. 3. адміністратори/модератори: користувачі, які здійснюють контроль якості контенту, модерацію коментарів та управління обліковими записами.

Кінець таблиці 1.4

Ключові переваги	1. мінімалістичний ui/ux: дизайн орієнтований на контент, без перевантаження рекламою та зайвими алгоритмічними підказками; 2. висока продуктивність: використання next.js із гібридним рендерингом та cdn-кешуванням забезпечує миттєвий відгук плеєра та швидкі переходи між сторінками; 3. спрощене керування бібліотекою: автори отримують інтуїтивний механізм завантаження та організації треків.
Бізнес-модель	Freemium. Безкоштовний доступ містить ліміт на завантаження контенту та періодичну рекламу після кожних 10 треків. Преміум-підписка надає розширене хмарне сховище та можливість прослуховування у високій якості без втрат (lossless).

Отже, SoundFlex є легковаговим, але функціональним рішенням, яке усуває ключові недоліки розглянутих аналогів: UI перевантаженість, недостатню швидкодію та складність управління контентом. Завдяки опорі на сучасний стек технологій (Next.js + Firebase) система здатна забезпечити стабільне HLS навіть за умов нестабільного мережевого з’єднання, що робить її привабливою як для авторів-початківців, так і для кінцевих слухачів.

1.3 Методи та технології передачі потокових мультимедійних даних

Сучасна вебдистрибуція аудіоконтенту базується на здатності доставляти звуковий потік користувачеві без необхідності повного завантаження файлу перед початком відтворення. Це досягається за допомогою технологій потокової передачі, які забезпечують плавне, безперервне прослуховування навіть за умов нестабільної пропускної здатності мережі. У цьому підрозділі розглянуто основні методи, протоколи та формати, що застосовуються для передачі аудіопотоків у вебсередовищі, а також обґрунтовано вибір підходів для платформи SoundFlex.

Прогресивне завантаження та справжнє потокове відтворення

Історично склалися два принципові підходи до відтворення аудіо в браузері. Перший – прогресивне завантаження (progressive download) – передбачає отримання файлу за звичайним протоколом HTTP, при цьому відтворення може розпочатися, щойно завантажено достатній початковий сегмент. Такий метод простий у реалізації, але має обмеження: користувач не може вільно переміщуватись по не завантаженій частині треку, а зміна якості під час відтворення неможлива. Другий – справжнє HLS (true streaming) – використовує спеціалізовані протоколи та сервери, що передають дані невеликими пакетами в реальному часі, дозволяючи динамічно адаптувати бітрейт і виконувати довільну навігацію по аудіоматеріалу.

Адаптивні потокові протоколи (HLS та MPEG-DASH)

Найбільш поширеними сьогодні протоколами справжнього потокового відтворення є HLS (HTTP Live Streaming) від Apple [2] та MPEG-DASH (Dynamic Adaptive Streaming over HTTP) [3]. Обидва рішення працюють поверх HTTP, що дозволяє використовувати звичайні вебсервери та мережі доставлення контенту (CDN). Аудіофайл розбивається на короткі сегменти тривалістю кілька секунд, кожен з яких кодується в кількох варіантах якості. Спеціальний файл-маніфест (.m3u8 для HLS, .mpd для DASH) описує доступні бітрейтні профілі та посилання на сегменти. Клієнтський плеєр вимірює поточну швидкість з’єднання та автоматично перемикається між якісними рівнями, забезпечуючи безперервне відтворення без буферизації. Для аудіоконтенту така адаптивність особливо цінна у мобільних мережах із нестабільною пропускною здатністю.

Інші протоколи реального часу

Окремо варто згадати протоколи RTMP (Real-Time Messaging Protocol) та WebRTC. RTMP довгий час використовувався для передачі аудіо й відео між сервером і Flash-плеєром, проте на сьогодні втратив актуальність через відмову від Flash-технологій. WebRTC, навпаки, набуває популярності

завдяки наднизькій затримці, що робить його ідеальним для відеоконференцій та прямих етерів, однак для асинхронного музичного стрімінгу, де затримка не є критичною, його застосування невиправдано ускладнює інфраструктуру.

Аудіокодеки та формати контейнерів

Ефективність потокової передачі суттєво залежить від вибору кодека. Традиційний MP3 досі підтримується всіма платформами, проте сучасніші AAC (Advanced Audio Coding) та Opus забезпечують кращу якість за нижчих бітрейтів. Opus, зокрема, є відкритим, безліцензійним кодеком, оптимізованим як для мови, так і для музики, і рекомендований IETF як основний кодек для WebRTC [4]. Для lossless-аудіо застосовують FLAC, що дозволяє зберегти студійну якість, проте вимагає значно більшої пропускної здатності. Контейнери (MP4, WebM, Ogg) поєднують аудіопотік із метаданими, вибір контейнера часто диктується сумісністю з обраним протоколом: HLS традиційно працює з MP4-сегментами (fMP4), тоді як DASH є контейнернезалежним.

Використання CDN та роль хмарного бекенду

Для глобальної доставки аудіоконтенту мінімальною затримкою застосовують мережі доставлення контенту (CDN). Платформа Firebase, обрана для проєкту SoundFlex, зберігає файли у Google Cloud Storage, яке автоматично інтегрується з глобальним CDN Google, забезпечуючи кешування на крайових вузлах, близьких до кінцевих користувачів. Завдяки цьому навіть прогресивне завантаження або сегменти HLS доставляються з мінімальною затримкою без розгортання додаткової інфраструктури.

Вибір підходу для платформи SoundFlex

З огляду на цільову аудиторію (незалежні автори та слухачі), а також технічний стек (Next.js + Firebase), для реалізації потокового відтворення в проєкті обрано комбінований підхід. Базовим механізмом виступатиме прогресивне завантаження аудіофайлів через пряме посилання на Firebase Storage з підтримкою HTTP-запитів із частковим вмістом (byte-range requests).

Це дає змогу розпочинати відтворення до завершення завантаження всього файлу та виконувати перемотування, не вимагаючи при цьому складної серверної обробки. Для майбутнього масштабування архітектура не виключає впровадження протоколу HLS, який дозволить динамічно змінювати якість залежно від швидкості з'єднання слухача. Кодування завантажених авторами файлів у формати AAC або Opus (з пріоритетом останнього) забезпечить баланс між якістю звуку та обсягом спожитого трафіку.

1.4 Обґрунтування вибору технологічного стеку розробки.

Вибір технологічного стеку є одним із ключових рішень під час проєктування ПЗ, оскільки він безпосередньо впливає на продуктивність, масштабованість, терміни розробки та зручність подальшої підтримки системи. Для платформи SoundFlex, яка створюється як легковаговий стрімінговий сервіс для незалежних авторів і слухачів, було поставлено такі вимоги до технологій:

– висока швидкість завантаження сторінок та відтворення аудіо;
– мінімалістичний, адаптивний UI без зайвої складності;
– відсутність потреби в адмініструванні серверної інфраструктури;
– можливість швидкого ітеративного прототипування та внесення змін.

Виходячи з цих вимог, було обрано зв'язку Next.js, Firebase, Tailwind CSS, shadcn/ui та TypeScript. Нижче наведено детальне обґрунтування вибору кожного компонента стеку.

Клієнтська частина: фреймворк Next.js та TypeScript

Next.js є одним із найпопулярніших фреймворків для побудови вебзастосунків на основі React. Його ключовою перевагою є підтримка гібридного рендерингу, яка дозволяє поєднувати серверний рендеринг (SSR), статичну генерацію (SSG) та клієнтський рендеринг у межах одного проєкту [5]. Архітектура App Router, запроваджена в актуальних версіях Next.js, надає

зручну файлову маршрутизацію [6] , підтримку серверних компонентів React, а також оптимізацію зображень, шрифтів і скриптів «з коробки». Для стрімінгової платформи це означає, що головна сторінка або сторінка виконавця може бути згенерована статично з подальшою інкрементальною ревалідацією (ISR), що забезпечує миттєве завантаження та позитивно впливає на SEO.

На відміну від звичайного React-додатку, скомпільованого виключно під клієнт, Next.js дозволяє виконувати частину логіки на сервері, зменшуючи обсяг JavaScript, що передається у браузер [7]. Це критично важливо для слухачів із мобільними пристроями та повільним інтернет-з'єднанням. Використання TypeScript як основної мови програмування додає жорстку типізацію, що знижує ймовірність помилок під час взаємодії з даними Firestore та під час передачі пропсів між компонентами, полегшуючи підтримку проєкту в довгостроковій перспективі [8].

Стилізація

Для реалізації мінімалістичного, візуально чистого UI було обрано утилітарний CSS-фреймворк Tailwind CSS [9]. Його ідеологія «utility-first» дозволяє будувати складні макети безпосередньо в класах HTML, що пришвидшує верстку та не вимагає написання громіздких каскадних стилів. Tailwind CSS також добре інтегрується з механізмом «treeshaking», автоматично видаляючи невикористані стилі під час збірки, що зменшує фінальний розмір CSS-файлів.

Бібліотека shadcn/ui надає готові компоненти – кнопки, діалогові вікна, картки, форми, побудовані на основі Radix UI, з високим рівнем доступності (accessibility). На відміну від традиційних npm-пакетів, shadcn/ui інтегрується шляхом копіювання сирцевого коду компонентів у проєкт (конфігурація components.json), що дає повний контроль над їхньою реалізацією та стилізацією. Це ідеально узгоджується з вимогою мінімалізму: розробник

може використовувати лише необхідні компоненти, не перевантажуючи застосунок зайвими залежностями.

Серверна частина та хмарна інфраструктура: Firebase

Для реалізації серверної логіки та управління інфраструктурою було обрано екосистему Firebase від компанії Google. Firebase позиціонується як комплексне Backend-as-a-Service (BaaS) рішення, інтеграція якого дозволяє абстрагуватися від низькорівневих задач адміністрування серверів, налаштування середовищ виконання та підтримки складного серверного коду. Вибір даної платформи ґрунтується на необхідності створення високонавантаженого прототипу в обмежені терміни та базується на наступних архітектурних перевагах:

1. Firebase authentication надає готові механізми автентифікації через електронну пошту, google-акаунти та інші провайдери. інтеграція з next.js відбувається через зручні sdk, що суттєво скорочує час розробки модуля авторизації;

2. cloud firestore – це гнучка nosql БД реального часу, яка дозволяє зберігати метадані треків, профілі користувачів і плейлисти. її клієнтські бібліотеки забезпечують автоматичну синхронізацію даних між сервером і клієнтом, що спрощує реалізацію реактивних оновлень UI (наприклад, миттєве відображення щойно завантаженого треку без перезавантаження сторінки). правила безпеки (firestore.rules) дають змогу гнучко контролювати доступ до даних без додаткового проміжного шару api;

3. Firebase storage використовується для зберігання аудіофайлів. він базується на google cloud storage [10] і автоматично інтегрується з глобальною мережею доставлення контенту google cdn. це означає, що аудіотреки будуть доступні з високою швидкістю незалежно від географічного положення слухача, що особливо важливо для потокового відтворення, описаного в підрозділі 1.3;

4. Firebase hosting забезпечує швидке та безпечне розгортання вебзастосунку з автоматичним ssl-сертифікатом та інтеграцією з глобальним cdn. це усуває потребу в налаштуванні nginx, docker чи kubernetes і дозволяє зосередитися безпосередньо на функціональності.

Альтернативним варіантом могло б стати розгортання самостійного серверу на Node.js з фреймворком Express та базою даних MongoDB Atlas (або PostgreSQL). Однак такий підхід вимагав би написання та документування REST API, налаштування середовища виконання, моніторингу, резервного копіювання та масштабування. Для дипломного проєкту, орієнтованого на швидку розробку повнофункціонального прототипу, витрати на подібну інфраструктуру були б нераціональними. Firebase надає ті самі можливості, але з мінімальними початковими зусиллями та передбачуваною ціновою моделлю (freemium із щедрим безкоштовним рівнем), що повністю відповідає бізнес-моделі SoundFlex.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи проведено всебічний аналіз предметної області вебдистрибуції аудіоконтенту. Здійснено огляд провідних стрімінгових платформ – SoundCloud, Spotify та YouTube Music, у межах якого розглянуто їхній функціонал, особливості UI, переваги, недоліки та бізнес-моделі. Це дало змогу виявити ключові очікування користувачів, а також окреслити нішу для власної розробки – легковагової платформи, орієнтованої на незалежних авторів і слухачів без зайвих посередників.

На підставі отриманих результатів сформульовано концепцію системи SoundFlex, визначено її цільову аудиторію, основні функціональні характеристики та бізнес-модель freemium. Проведено детальний аналіз методів і технологій передачі потокових мультимедійних даних, зокрема протоколів HLS, MPEG-DASH, прогресивного завантаження, а також сучасних аудіокодеків. Це дозволило обґрунтувати вибір комбінованого

підходу до відтворення контенту з опорою на byte-range запити та перспективою впровадження адаптивного стрімінгу.

У завершальній частині розділу аргументовано обрано технологічний стек проєкту: Next.js із TypeScript для клієнтської частини, Tailwind CSS і shadcn/ui для стилізації, а також Firebase як хмарну серверну інфраструктуру. Така комбінація забезпечує високу продуктивність, швидкість розробки та мінімальне адміністрування, що повністю відповідає вимогам до системи.

Виконаний аналіз дав змогу розвинути навички пошуку та усунення недоліків наявних рішень. Результати, отримані в цьому розділі, виступають фундаментом для подальшого науково-технічного обґрунтування, конкретизації функціональних і нефункціональних вимог, а також для проєктування архітектури вебзастосунку, що буде розглянуто в наступних розділах.

Узагальнюючи, можна стверджувати, що проведений у розділі аналіз має не лише оглядовий, а й прикладний характер: на його основі визначено конкретні критерії відбору технологій та архітектурних рішень, які безпосередньо враховують виявлені слабкі місця наявних сервісів - недостатню видимість контенту незалежних авторів у пошукових системах, затримки під час старту відтворення та ускладнене адміністрування. Чітке розмежування відомих рішень і власних пропозицій дає змогу зосередити подальшу роботу на створенні затребуваного програмного продукту та задає технічні орієнтири, дотримання яких перевірятиметься під час проєктування, реалізації й тестування системи.

2 НАУКОВО-ТЕХНІЧНЕ ОБҐРУНТУВАННЯ ТА ВИМОГИ ДО СИСТЕМИ

2.1 Огляд сучасних інструментаріїв та моделей розробки вебплатформ.

Побудова веборієнтованої стрімінгової платформи для дистрибуції аудіоконтенту вимагає вибору архітектурної моделі, здатної забезпечити високу продуктивність, зручність індексації пошуковими системами та швидке завантаження на різноманітних пристроях. Сучасна практика пропонує дві домінуючі парадигми: Single Page Application (SPA) та серверний рендеринг (Server Side Rendering, SSR) разом із похідними (Static Site Generation, SSG; ISR, що реалізовані в межах фреймворків типу Next.js. Вибір між ними безпосередньо впливає на поведінку користувачів, охоплення аудиторії через пошукові системи та загальну сприйману швидкодію.

SPA-архітектура передбачає, що вебзастосунок завантажується одноразово у вигляді JavaScript-пакету, після чого всі наступні переходи та оновлення даних відбуваються без повного перезавантаження сторінки. Для платформ із насиченою інтерактивністю, таких як аудіоплеєри з миттєвою реакцією на керування відтворенням, SPA надає суттєві переваги у вигляді плавної навігації та зменшення трафіку. Водночас дослідження показують, що класичні SPA мають низку обмежень: перший рендер часто затримується через великий обсяг початкового бандла, а пошукові роботи не завжди здатні коректно виконати JavaScript для індексації динамічного контенту. Для музичних сервісів, де кожна сторінка треку або виконавця повинна бути видимою в результатах пошуку, це створює серйозну проблему.

Серверний рендеринг, реалізований у фреймворку Next.js, пропонує компроміс: під час першого запиту користувачеві повертається повністю сформований HTML з усіма метаданими, після чого на клієнті відбувається гідратація React-компонентів, і додаток починає поводитися як SPA. Такий

підхід особливо цінний саме для каталогів аудіоконтенту, оскільки назви треків, імена авторів, обкладинки альбомів та текстовий опис стають доступними для індексації без будь-яких додаткових зусиль . Крім того, Next.js підтримує статичну генерацію (SSG) та інкрементальну ревалідацію (ISR): сторінки з динамічним вмістом, наприклад список нових релізів, можуть оновлюватися з визначеною періодичністю без повного перескладання. Це зменшує навантаження на сервер і водночас гарантує актуальність даних.

Варто підкреслити, що SEO-аспект є критичним фактором для SoundFlex як платформи, орієнтованої на незалежних авторів. Контент, що генерується авторами, має бути максимально видимим для пошукових систем, аби слухачі могли знаходити треки за ключовими словами навіть без прямої реклами. Згідно з даними [3] , вебзастосунки із серверним рендерингом отримують суттєво кращі позиції в органічному пошуку порівняно з SPA без додаткових налаштувань. У випадку SoundFlex, де кожна сторінка аудіозапису має унікальний URL, серверний рендеринг через Next.js гарантує, що відповідні meta-теги, заголовки та мікродані будуть присутні у вихідному HTML і правильно опрацьовані пошуковими роботами.

Для наочного зіставлення двох архітектурних підходів та їхнього впливу на індексацію аудіоконтенту наведено рисунок 2.1.

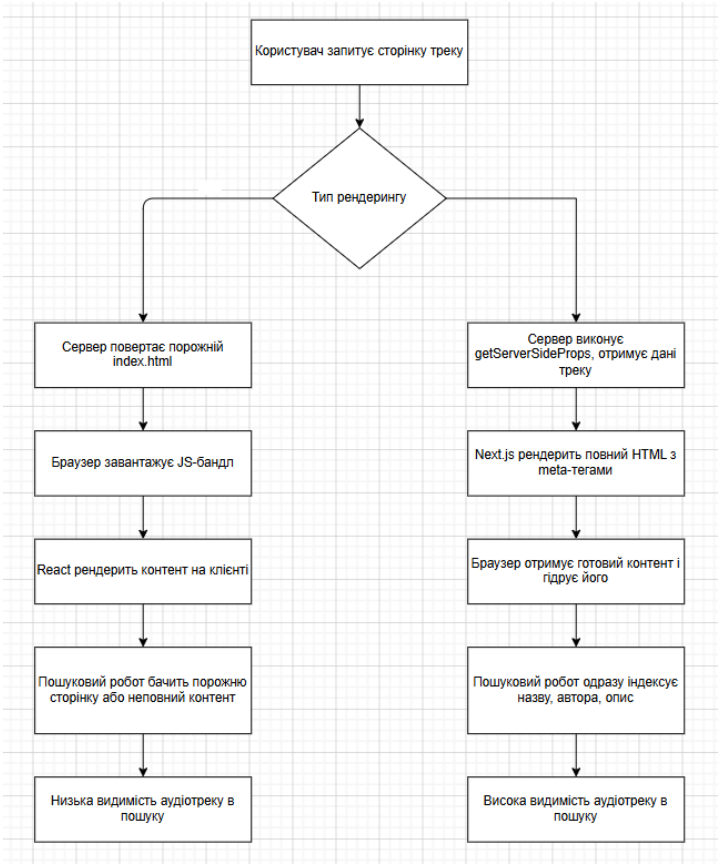

Рисунок 2.1 – Порівняння моделей SPA та SSR у контексті SEO для сторінок аудіотреків

Поряд із SSR/SSG, для стрімінгових сервісів часто застосовують бекенд-архітектуру на основі мікросервісів або Serverless-функцій. Проте, відповідно до концепції SoundFlex, де важливо мінімізувати адміністрування та знизити поріг входження для розробки, використання хмарних платформ (Firebase, Supabase тощо) у поєднанні з API-маршрутами Next.js видається найбільш раціональним. Це дозволяє уникнути розгортання окремих сервісів і зосередитися на головному – швидкій доставці аудіо.

2.2 Аналіз методів оптимізації доступу до об’єктних сховищ даних.

Ефективна дистрибуція аудіоконтенту неможлива без надійного та швидкого механізму зберігання й доставки великих бінарних файлів. Традиційний підхід, за якого файли розміщуються безпосередньо на віртуальному або фізичному сервері поряд із програмним кодом, стикається з

низкою обмежень, особливо у стрімінгових сценаріях. Зростання обсягу даних, необхідність глобальної доступності та високі вимоги до швидкості змушують шукати альтернативи у вигляді об’єктних сховищ (object storage), інтегрованих із мережами доставлення контенту (CDN) [11].

Об’єктні сховища проти традиційних файлових серверів

Традиційні файлові сервери передбачають зберігання файлів у звичайній ієрархічній файловій системі, доступ до якої здійснюється через протоколи FTP або HTTP. Для невеликих проєктів такий підхід є простим, проте зі зростанням кількості користувачів виникають проблеми масштабування: пропускна здатність сервера обмежена, резервування потребує складних конфігурацій, а географічне розташування сервера безпосередньо впливає на затримку для віддалених слухачів.

Об’єктні сховища, такі як Google Cloud Storage (що лежить в основі Firebase Storage), Amazon S3 або Azure Blob Storage, усувають ці недоліки. Вони оперують не файлами в класичному розумінні, а об’єктами, кожен з яких має унікальний ідентифікатор і зберігається в плоскому просторі імен. Це дає змогу горизонтально масштабувати сховище без видимого для клієнта переривання сервісу. Крім того, об’єктні сховища за замовчуванням реплікують дані в кількох дата-центрах, забезпечуючи високу доступність і відмовостійкість [13].

Роль CDN та кешування

Навіть найшвидше хмарне сховище не здатне повністю компенсувати фізичну відстань між сервером і клієнтом. Тому критично важливим компонентом стає мережа доставлення контенту (CDN). Firebase Storage автоматично інтегрується з Google Cloud CDN, що забезпечує кешування аудіофайлів на крайових вузлах, максимально наближених до кінцевого користувача. Коли слухач запитує трек, перший запит проходить до найближчого вузла CDN; якщо об’єкт уже знаходиться в кеші, він доставляється з мінімальною латентністю, оминаючи центральне сховище.

Якщо ж кеш порожній, вузол завантажує файл із регіонального бакета Google Cloud Storage, одночасно зберігаючи його для наступних запитів.

Окрім зменшення затримки, CDN знижує навантаження на магістральний канал сховища, що дозволяє обслуговувати значно більшу кількість одночасних слухачів без деградації продуктивності. Для незалежних авторів, чиї треки можуть стати вірусними несподівано, така еластичність є беззаперечною перевагою.

Використання часткових HTTP-запитів (byte-range requests)

Стрімінгове відтворення вимагає не лише швидкої доставки, а й довільного доступу до окремих частин файлу. Firebase Storage підтримує HTTP-запити з частковим вмістом (Range-запити), що дозволяє клієнтському плеєру отримувати необхідні сегменти аудіо без завантаження всього файлу. Це особливо важливо при перемотуванні або старті відтворення з середини треку. Поєднання CDN-кешування та byte-range запитів створює ефективний механізм доставки, близький за характеристиками до спеціалізованих потокових протоколів, але значно простіший в реалізації [12].

Переваги Firebase Storage для платформи SoundFlex

З точки зору незалежного автора або невеликої студії, головними критеріями вибору сховища є мінімальне адміністрування, передбачувана вартість і висока швидкість доставки. Firebase Storage задовольняє ці вимоги повністю:

– відсутність інфраструктурних турбот – немає потреби встановлювати, оновлювати чи масштабувати сервери;

– оплата лише за фактичне використання, тобто модель pay-as-you-go дозволяє уникнути витрат на незадіяні ресурси;

– вбудована інтеграція з Firebase Authentication дає змогу розмежовувати доступ на основі правил безпеки, записаних у firestore.rules;

– глобальна доступність через CDN гарантує, що трек завантажуватиметься однаково швидко як у Києві, так і в Лондоні.

Для унаочнення різниці між традиційним серверним підходом і хмарним об’єктним сховищем наведено таблицю 2.1.

Таблиця 2.1 – Порівняння традиційного файлового сервера та Firebase Storage

Критерій	Традиційний сервер	Firebase Storage (Google Cloud)
Масштабованість	Вертикальна, обмежена апаратним забезпеченням	Горизонтальна, автоматична
Глобальна доступність	Залежить від розташування сервера	Автоматична через CDN Google
Відмовостійкість	Потребує ручного налаштування реплікації	Вбудована реплікація між дата-центрами
Кешування	Лише за додаткового налаштування (Varnish, Nginx)	Автоматичне, на периферійних вузлах CDN
Обслуговування	Постійне адміністрування, оновлення ПЗ	Повністю керований сервіс, без адміністрування
Модель оплати	Фіксована оренда або капітальні витрати	Pay-as-you-go, пропорційно використанню
Підтримка Range-запитів	Так, за умови коректного налаштування	Природна, без додаткових дій

Схематичний процес доставки аудіофайлу із Firebase Storage через CDN до клієнта показано на рисунку 2.2.

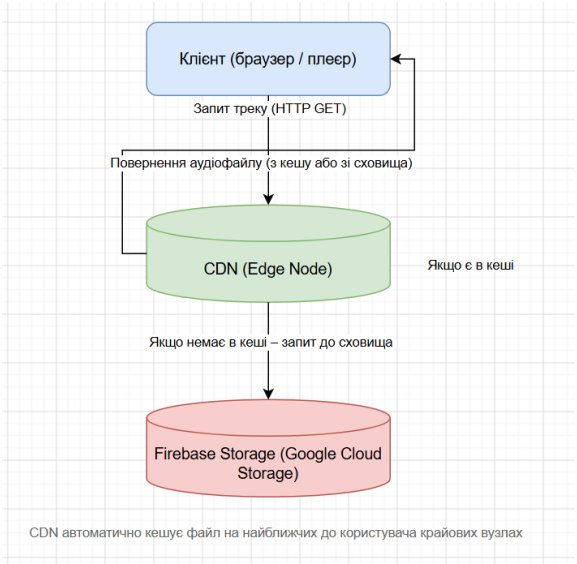

Рисунок 2.2 – Схема доставки аудіоконтенту через Firebase Storage із використанням CDN

Таким чином, вибір Firebase Storage як об’єктного сховища для SoundFlex дозволяє вирішити одразу кілька завдань: мінімізувати затримки під час відтворення, зняти навантаження з основного сервера та забезпечити високу доступність контенту. Отримані результати аналізу методів оптимізації доступу будуть використані під час формування конкретних функціональних і нефункціональних вимог до системи, що розглянуто в наступному підрозділі.

2.3 Формування функціональних та нефункціональних вимог до ПЗ

Спираючись на проведений у попередніх підрозділах аналіз інструментаріїв (2.1) та методів оптимізації доступу до об’єктних сховищ (2.2), а також на результати дослідження предметної області з розділу 1, формулюються конкретні вимоги до платформи SoundFlex. Специфікація вимог є обов’язковим етапом проєктування, оскільки саме вона визначає межі функціональності, критерії якості та обмеження, яким має задовольняти кінцевий продукт [14]. Згідно зі стандартом ISO/IEC 25010, вимоги поділяються на функціональні (що система повинна робити) та нефункціональні (за яких умов і з якою якістю виконуються функції) [15].

Функціональні вимоги

Функціональні вимоги описують базову поведінку системи з точки зору користувачів – авторів і слухачів. На основі аналізу бізнес-моделі freemium, визначеної в підрозділі 1.2, та з урахуванням обраного технологічного стеку сформульовано перелік ключових функцій, наведений у таблиці 2.2.

Таблиця 2.2 – Специфікація функціональних вимог до SoundFlex

Ідентифікатор	Вимога	Користувач	Пріорітет
FR-01	Реєстрація через e-mail та Google-акаунт	Авторизація	Високий
FR-02	Автентифікація з використанням JWT, збереження сесії	Авторизація	Високий
FR-03	Завантаження аудіофайлів (MP3, WAV, FLAC, AAC, Opus)	Управління контентом	Високий
FR-04	Внесення метаданих: назва, автор, опис, обкладинка	Управління контентом	Високий
FR-05	Редагування та видалення власних треків	Управління контентом	Середній
FR-06	Пошук треків за назвою та іменем автора	Навігація	Високий
FR-07	Потокове відтворення аудіо з підтримкою перемотування	Відтворення	Високий
FR-08	Безперервне фонове відтворення під час навігації	Відтворення	Середній
FR-09	Додавання / вилучення треків у «Вибране»	Бібліотека	Середній

Кінець таблиці 2.2

FR-10	Лічильник прослуховувань для кожного треку	Аналітика	Низький
FR-11	Перегляд списку всіх треків із можливістю видалення будь-якого контенту	Адміністрування	Середній
FR-12	Блокування/розблокування облікових записів користувачів	Адміністрування	Середній
FR-13	Модерація коментарів (видалення небажаного вмісту)	Адміністрування	Низький

Окремо варто виділити функціональну вимогу FR-07, яка безпосередньо випливає з аналізу методів передачі даних (підрозділ 1.3). Завдяки підтримці Firebase Storage byte-range запитів, плеєр отримує змогу розпочинати відтворення до повного завантаження файлу та вільно переміщуватися по треку. Це забезпечує зручність, аналогічну нативним застосункам, без встановлення додаткового ПЗ. Вимога FR-08 реалізується через SPA-архітектуру Next.js: оскільки сторінка не перезавантажується під час навігації, аудіоплеєр продовжує роботу без переривань, що є критичним для стрімінгового сервісу.

Нефункціональні вимоги

Нефункціональні вимоги визначають критерії якості, яким має задовольняти система. Вони охоплюють продуктивність, безпеку, сумісність, зручність використання та масштабованість.

Продуктивність

1. NFR-01: час першого завантаження сторінки (First Contentful Paint) не повинен перевищувати 2 секунди за умови використання широкосмугового з’єднання. Це досягається завдяки серверному рендерингу Next.js та статичній генерації (SSG) для сторінок каталогу;

2. NFR-02: буферизація аудіо перед початком відтворення має тривати не більше 1,5 секунди. Обґрунтування цього показника випливає з

аналізу CDN-кешування (підрозділ 2.2): завдяки розподіленню файлів через Google Cloud CDN затримка мінімізується для користувачів незалежно від їхнього географічного положення.

Безпека даних

1. NFR-03: усі з'єднання між клієнтом і сервером здійснюються виключно через HTTPS із використанням TLS 1.3;
2. NFR-04: доступ до аудіофайлів у Firebase Storage контролюється правилами безпеки (firestore.rules). Завантажувати та редагувати метадані треку може лише автор, який його створив;
3. NFR-05: паролі користувачів зберігаються в хешованому вигляді із застосуванням солі (bcrypt) на боці Firebase Authentication;
4. NFR-06: усі вхідні дані від користувача проходять валідацію для запобігання XSS- та SQL-ін'єкціям.

Сумісність і кросбраузерність

1. NFR-07: вебзастосунок має коректно функціонувати в останніх стабільних версіях браузерів: Google Chrome, Mozilla Firefox, Safari (версії 2023 року та новіші) та Microsoft Edge;
2. NFR-08: UI повинен бути адаптивним і забезпечувати зручну роботу на пристроях із шириною екрана від 320 px (мобільні телефони) до 2560 px (широкоформатні монітори). Це гарантується використанням Tailwind CSS із вбудованими класами адаптивної верстки (підрозділ 1.4).

Зручність використання

1. NFR-09: UI має бути інтуїтивно зрозумілим і мінімалістичним, без зайвих елементів. В основі дизайну лежить система компонентів shadcn/ui, що забезпечує візуальну єдність;
2. NFR-10: усі інтерактивні елементи (кнопки, посилання, поля введення) повинні мати чіткі стани (hover, focus, active) і відповідати вимогам доступності WCAG 2.1 рівня AA.

Масштабованість і надійність

1. NFR-11: система має витримувати одночасне потокове відтворення (HLS) щонайменше 500 активними слухачами без помітної деградації швидкості. Це забезпечується горизонтальним масштабуванням Firebase Storage та CDN (підрозділ 2.2);

2. NFR-12: доступність системи повинна становити не менше 99,9% на місяць, що гарантується угодами про рівень обслуговування (SLA) платформи Google Cloud.

Для унаочнення структури вимог та їхнього зв'язку з архітектурними рішеннями наведено рисунок 2.3.

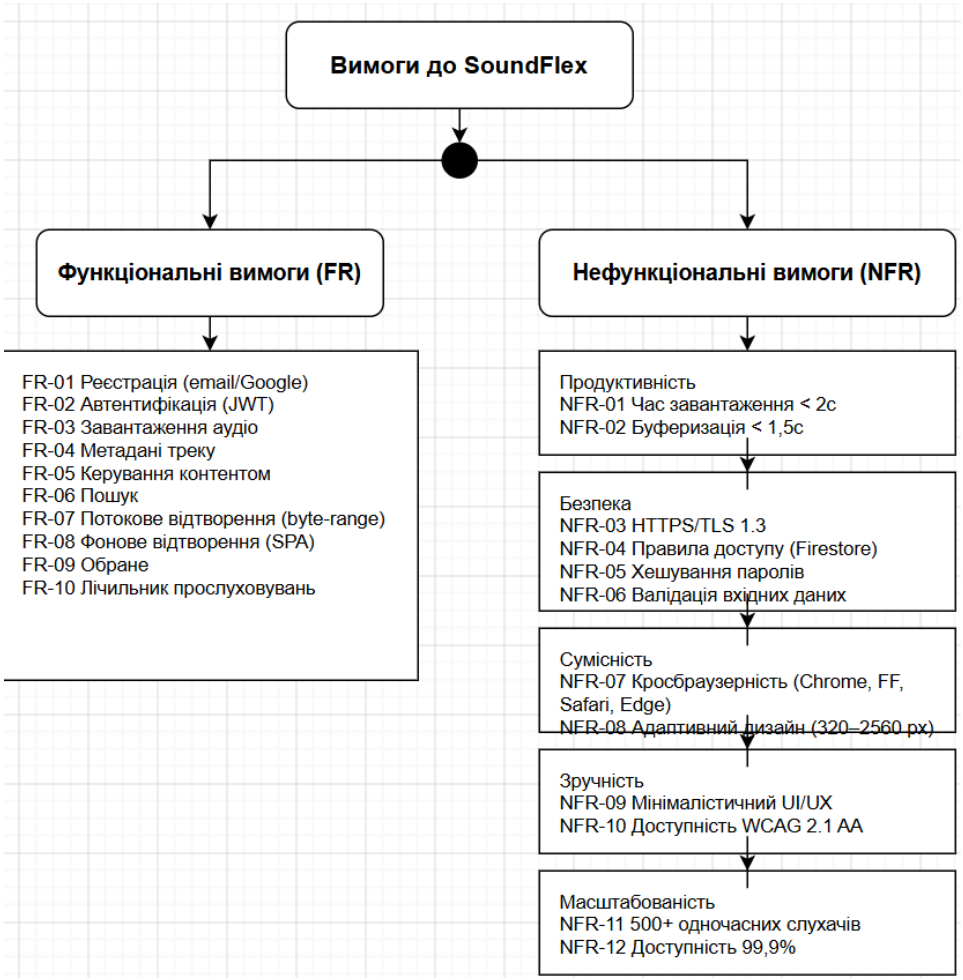

Рисунок 2.3 – Структурна схема вимог до платформи SoundFlex

Сформована специфікація вимог охоплює як базову функціональність, необхідну для запуску платформи, так і якісні характеристики, що визначають конкурентоспроможність продукту. Функціональні вимоги FR-01 – FR-13

витікають із бізнес-моделі freemium та аналізу цільової аудиторії, тоді як нефункціональні вимоги NFR-01 – NFR-12 безпосередньо пов’язані з результатами дослідження продуктивності (NFR-01, NFR-02), безпеки (NFR-03 – NFR-06), кросбраузерності (NFR-07, NFR-08), зручності (NFR-09, NFR-10) і масштабованості (NFR-11, NFR-12). У сукупності вони створюють основу для переходу до етапу проєктування архітектури, якому присвячено наступний розділ.

Висновки до розділу 2

У другому розділі кваліфікаційної роботи проведено науково-технічне обґрунтування проєктних рішень для вебплатформи дистрибуції аудіоконтенту SoundFlex. Описано етапи аналізу сучасних інструментаріїв, моделей та методів, що дозволяють вирішити поставлені завдання, а також сформовано повну специфікацію вимог до ПЗ. Завдяки цьому закріплено навички у проведенні порівняльного аналізу архітектурних парадигм (SPA та SSR), дослідженні методів оптимізації доступу до хмарних об’єктних сховищ, а також у формуванні функціональних і нефункціональних вимог згідно зі стандартом.

На основі опрацьованих наукових публікацій обґрунтовано вибір гібридної моделі рендерингу Next.js, яка поєднує переваги SPA для інтерактивного відтворення аудіо та SSR для забезпечення видимості контенту в пошукових системах. Доведено ефективність використання Firebase Storage у зв’язці з CDN як основного сховища аудіофайлів, що гарантує мінімальну латентність доставки незалежно від географічного розташування слухача. Сформована специфікація з 10 функціональних та 12 нефункціональних вимог чітко окреслює межі системи, критерії її якості та безпеки, а також створює фундамент для переходу до проєктування архітектури вебзастосунку, чому присвячено наступний розділ.

3 ПРОЄКТУВАННЯ ТА АРХІТЕКТУРА ВЕБЗАСТОСУНКУ

3.1 Архітектурні рішення та шаблони проєктування

Проєктування архітектури SoundFlex відштовхується від моделі односторінкового застосунку (SPA), доповненої хмарним бекендом за схемою Backend-as-a-Service (BaaS). Завдяки цьому розробник зосереджується на клієнтській логіці та користувацькому досвіді, а керування серверною інфраструктурою, автентифікацією і зберіганням даних бере на себе платформа Firebase. Класична триланкова архітектура неодмінно містить проміжний серверний API-шар між клієнтом і базою. У SoundFlex такого шару немає: клієнтський код спілкується з Firebase Auth, Cloud Firestore та Firebase Storage напряму, через офіційні Client SDK. Це знімає тягар написання й підтримки проміжного коду, пришвидшує обмін даними та відкриває шлях до реактивних оновлень у реальному часі.

Серцем клієнтської частини є фреймворк Next.js з архітектурою App Router. Кореневий макет layout.tsx формує сталий каркас застосунку: верхню навігаційну панель TopNav, промо-банер PromoBanner і компонент StickyPlayer. Саме тому аудіоплеєр лишається поза зоною перерендерення під час переходів між сторінками - і фонове відтворення не уривається, що відповідає вимозі FR-08. Сам плеєр спирається на стандартний елемент HTML5 audio, доступ до якого здійснюється через React-реф (useRef) усередині PlayerProvider. Від сторонніх бібліотек на кшталт Howler.js свідомо відмовлено: так зменшується розмір клієнтського бандла й досягається максимальна сумісність із браузерами.

Глобальним станом керує система React Context. FirebaseContext (файл src/firebase/provider.tsx) інкапсулює ініціалізацію Firebase-сервісів і передає дочірнім компонентам об’єкти auth, firestore та відомості про поточного користувача. PlayerContext (файл src/components/AudioPlayer/PlayerContext.tsx) утримує стан відтворення, чергу треків і рівень гучності. Такий поділ відповідає принципу єдиної

відповідальності (Single Responsibility Principle) та оберігає дерево компонентів від зайвого ускладнення.

Ініціалізацію Firebase винесено у файл src/firebase/index.ts і реалізовано за патерном Singleton. Функція initializeFirebase зчитує масив уже створених застосунків (getApps().length) і породжує новий екземпляр лише тоді, коли жодного ще немає [16]. Це убезпечує від повторної ініціалізації під час гарячої заміни модулів (Hot Module Replacement) у середовищі розробки Next.js. Експортовані сервіси firebaseApp, auth та firestore доступні всім частинам застосунку.

Доступ до даних Firestore забезпечує набір власних хуків - useFirestore, useCollection і useDoc, що є обгортками над методами Firebase SDK. Підписку на зміни в реальному часі реалізує функція onSnapshot, утілюючи патерн Observer: щойно документ чи колекція оновлюється (скажімо, додано новий трек), інтерфейс відображає це миттєво, без ручного перезавантаження сторінки. Операції запису й модифікації даних винесено у фасадні модулі каталогу src/firebase/firestore/ - users.ts для профілів, tracks.ts для треків, history.ts для історії прослуховувань. Вони дають уніфікований інтерфейс CRUD-операцій і ховають від решти застосунку деталі роботи з Firestore, що відповідає патерну Facade та знижує зв’язність між інтерфейсом і шаром даних.

Отже, архітектура SoundFlex переплітає простоту BaaS-моделі [17] з перевіреними шаблонами проєктування (Singleton, Observer, Facade) та сучасними можливостями Next.js App Router. Така структура дає високу швидкість розробки, реактивність інтерфейсу й легкість масштабування - усе, чого вимагають функціональні та нефункціональні вимоги розділу 2. Фізичне ж розміщення компонентів системи у браузері та у хмарі Firebase унаочнює діаграма розгортання, яку, дотримуючись логіки «від сценаріїв використання до інфраструктури розгортання», подано наприкінці цього розділу (рисунок 3.9).

3.2 Моделювання бізнес-процесів та поведінки системи

Спираючись на функціональні вимоги з підрозділу 2.3 та архітектурні рішення підрозділу 3.1, виконано моделювання ключових бізнес-процесів SoundFlex мовою UML. Першою побудовано діаграму варіантів використання (Use Case Diagram): з-поміж усіх діаграм вона найпростіша й базова, адже окреслює коло дійових осіб і перелік функцій системи, тоді як решта поведінкових діаграм (зокрема діаграми послідовності) є похідними від неї та лише деталізують окремі варіанти використання [18]. Саме така послідовність - від загальної картини використання до часткових сценаріїв - задає порядок викладу діаграм і в роботі, і в презентації.

Діаграму варіантів використання платформи SoundFlex наведено на рисунку 3.1. На ній визначено чотирьох акторів: Гість (неавторизований відвідувач), Слухач (зареєстрований користувач), Автор (користувач із правами публікації контенту) та Адміністратор (модератор платформи). Гість має доступ до пошуку, обмеженого прослуховування треків і реєстрації; Слухач додатково дістає бібліотеку, історію та додавання треків у «Вибране»; Автор - завантаження, редагування й видалення власних треків; Адміністратор - модерацію контенту та керування користувачами.

Зв’язки на діаграмі подано згідно з нотацією UML, причому для різних типів відношень застосовано різні графічні позначення:

– асоціація між актором і варіантом використання - суцільна лінія без стрілки; вона лише фіксує участь актора у сценарії й не позначає напряму передавання даних, тож стрілка тут була б помилкою;

– узагальнення між акторами - суцільна лінія з порожнім (незафарбованим) трикутним вістрям, спрямованим до загальнішого актора: «Слухач» успадковує можливості «Гостя», а «Автор» і «Адміністратор» - можливості «Слухача»;

– включення «include» - штрихова (пунктирна) лінія з відкритим вістрям до обов’язкового підваріанта: «Завантажити трек» включає «Перевірити формат файлу», а «Додати у Вибране» - «Автентифікуватися»;

– розширення «extend» - штрихова лінія з відкритим вістрям до базового варіанта для необов’язкової поведінки: «Згенерувати обкладинку засобами ШІ» розширює «Завантажити трек», а «Запропонувати Premium-підписку» розширює «Прослухати трек» у разі вичерпання ліміту безкоштовних прослуховувань.

Отже, напрямлене вістря в нотації UML використовують лише для відношень «include» та «extend» (штрихова лінія з відкритим вістрям) і для узагальнення (суцільна лінія з порожнім трикутником), тоді як звичайну участь актора у сценарії показують простою суцільною лінією без стрілки.

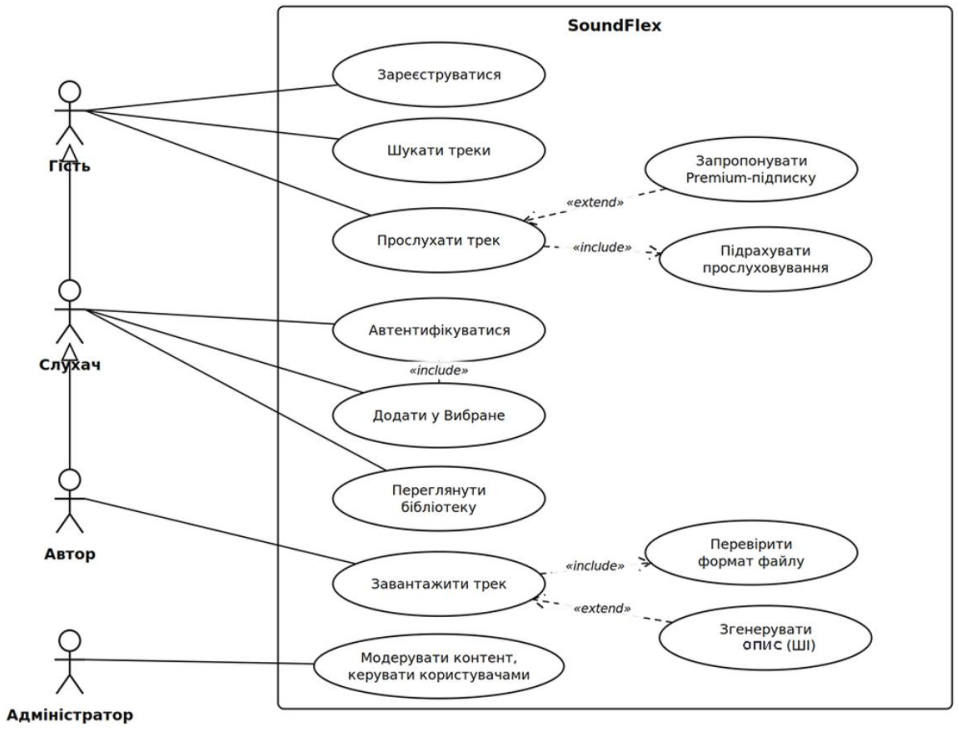

Рисунок 3.1 – Діаграма варіантів використання платформи SoundFlex

Визначені на діаграмі варіанти використання деталізовано далі діаграмами послідовності для трьох найважливіших сценаріїв роботи системи - реєстрації й автентифікації, завантаження треку та потокового відтворення.

Сценарій 1: реєстрація та автентифікація користувача

Автентифікація - перша точка входу до більшості функцій платформи, тож її коректність критична. У SoundFlex реалізовано вхід за електронною поштою та паролем. Функції createUserWithEmailAndPassword (для реєстрації) та signInWithEmailAndPassword (для входу) викликаються безпосередньо у компонентах форм src/app/register/page.tsx і src/app/login/page.tsx, що спрощує обробку станів завантаження та помилок в інтерфейсі.

Після успішної реєстрації в Firebase Auth автоматично з'являється документ профілю в колекції /users/{uid} із полями id, username, displayName, plan (за замовчуванням «free»), createdDate, а також масивами соціальних зв'язків (followers, following, likedTrackIds). Унікальність електронної пошти гарантується на рівні Firebase Auth, що унеможливлює дублікати облікових записів. Послідовність дій під час реєстрації нового користувача подано на рисунку 3.2.

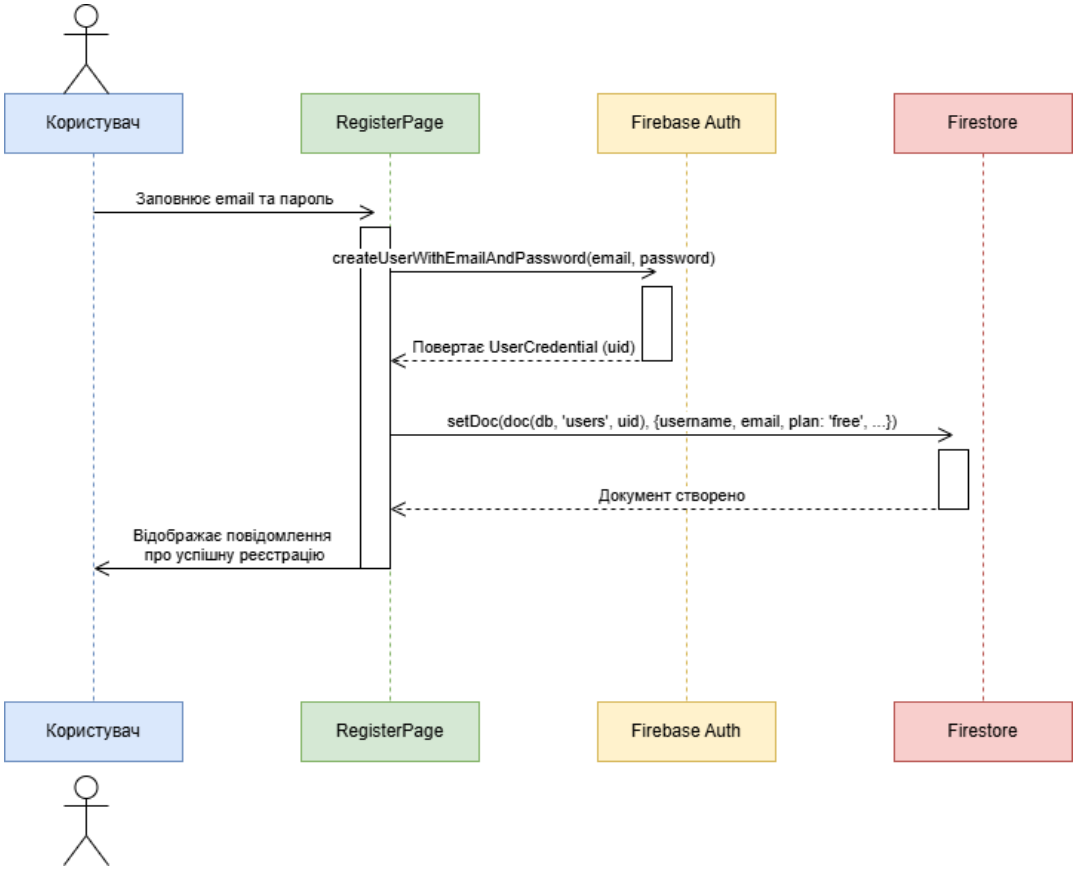

Рисунок 3.2 – Діаграма послідовності для сценарію реєстрації користувача

Сценарій 2: завантаження треку автором

Публікація нового аудіоматеріалу - центральний процес для авторів; він складається з кількох послідовних кроків: вибору аудіофайлу та обкладинки на пристрої, заповнення метаданих (назва, жанр, опис) у формі завантаження й остаточної публікації. Після підтвердження форми система вантажить бінарні файли у Firebase Storage за шляхами tracks/{userId}/{trackId}/audio.mp3 для аудіо та tracks/{userId}/{trackId}/cover.jpg для обкладинки, після чого створює документ у колекції /tracks/{trackId} з полями title, artistId, genre, coverArtUrl, audioOriginalFileUrl та uploadedDate. Такий порядок гарантує, що метадані з'являться в базі лише тоді, коли відповідні файли вже доступні у сховищі. Прогрес завантаження великих файлів відстежує механізм uploadTask.on('state_changed') із Firebase SDK, а валідація формату й розміру виконується на клієнті ще до старту. Діаграму послідовності для цього сценарію подано на рисунку 3.3.

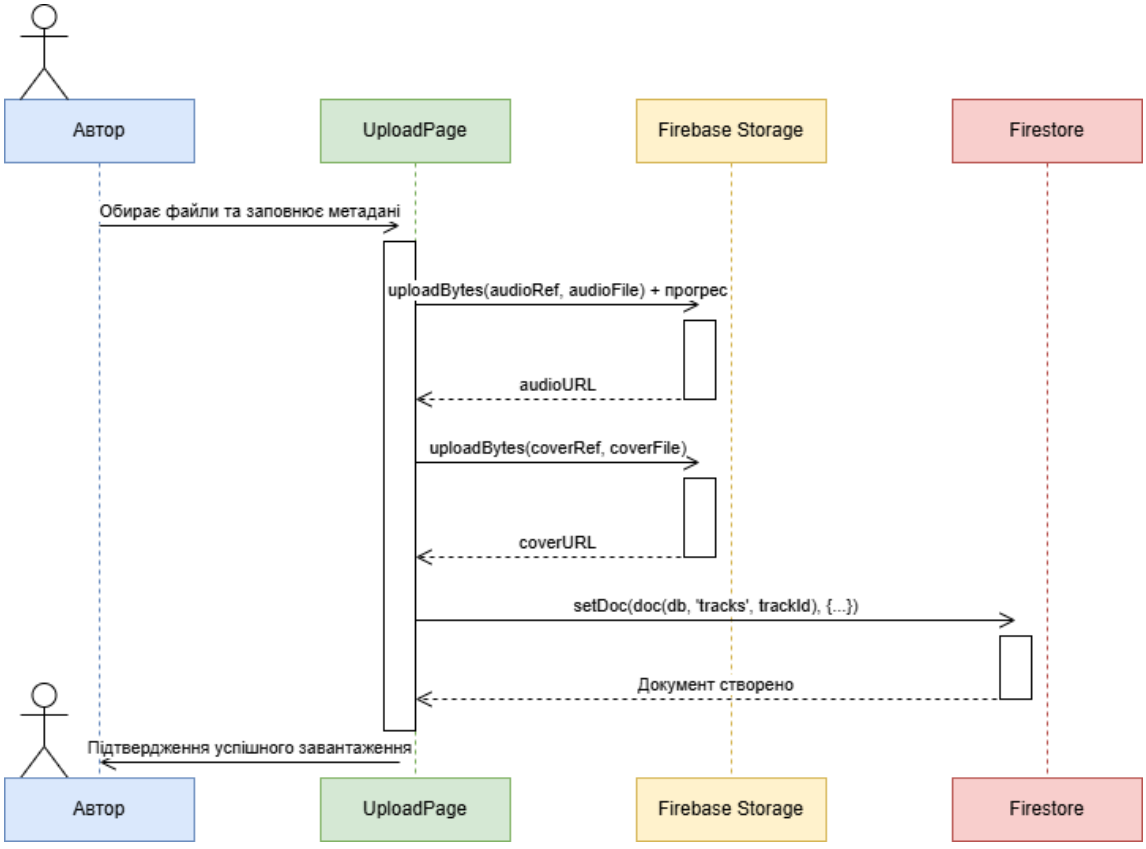

Рисунок 3.3 – Діаграма послідовності для сценарію завантаження треку

Сценарій 3: потокове відтворення та підрахунок прослуховувань

Відтворення аудіо - основна функція для слухачів. Щойно користувач натискає трек у списку, викликається функція playTrack(track) із PlayerContext: плеєр бере пряме посилання audioURL з об'єкта треку й ставить його джерелом стандартного елемента HTML5 audio. Оскільки PlayerProvider розташовано в кореневому layout.tsx, відтворення не уривається під час навігації між сторінками, що втілює вимогу FR-08. Підрахунок прослуховувань (вимога FR-10) реалізовано так: на старті відтворення система додає запис до історії прослуховувань користувача, а потім оновлює лічильник plays у документі треку атомарною функцією increment(1) із Firestore SDK. Атомарний інкремент зберігає коректність підрахунку навіть за одночасних запитів багатьох слухачів, а реактивні оновлення через onSnapshot миттєво показують зміну лічильника в інтерфейсі. Взаємодію компонентів під час відтворення треку зображено на рисунку 3.4.

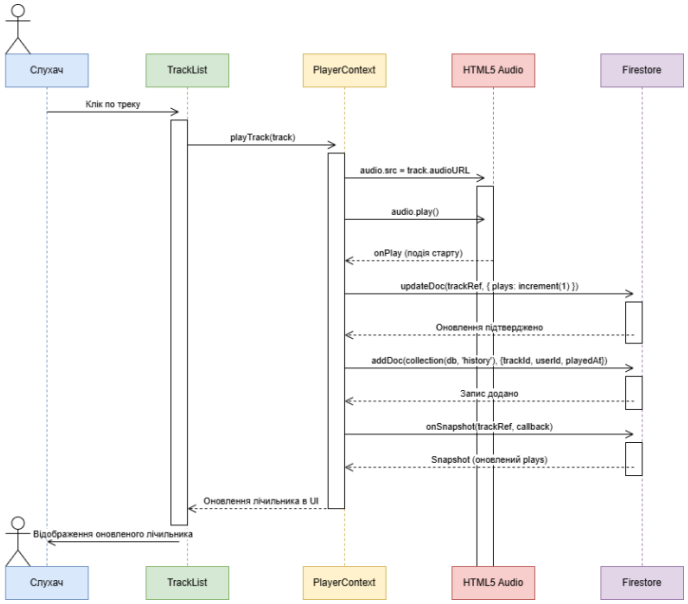

Рисунок 3.4 – Діаграма послідовності для сценарію потокового відтворення

Деталізовані сценарії та діаграми цього підрозділу закладають чітку основу для переходу до проєктування структури бази даних (підрозділ 3.3) і розроблення макетів інтерфейсу (підрозділ 3.4), де кожен бізнес-процес дістане структурне та візуальне втілення.

3.3 Проєктування структури бази даних

Вагомий етап проєктування SoundFlex - розроблення логічної моделі даних, що забезпечить ефективне зберігання, швидкий доступ і цілісність інформації про користувачів, аудіоконтент та їхню взаємодію. Серверну частину збудовано на хмарній NoSQL-базі Cloud Firestore, тому для моделювання обрано документоорієнтований підхід: дані зберігаються як ієрархічні JSON-подібні документи, згруповані в колекції. Відсутність жорсткої схеми дозволяє гнучко змінювати структуру документів у міру розвитку проєкту, а вбудована підтримка реального часу й автоматичне масштабування відповідають вимогам до продуктивності з підрозділу 2.3.

Центральні елементи моделі - дві колекції верхнього рівня: users і tracks. Колекція users тримає профілі зареєстрованих користувачів. Кожен документ ідентифікується унікальним ключем id, що дорівнює UID, отриманому від Firebase Authentication під час реєстрації. До документа входять поля username (унікальний нікнейм), displayName (публічне ім’я), plan (тарифний план, за замовчуванням «free»), bio (опис профілю), profileImageUrl і headerImageUrl (посилання на аватар та обкладинку), а також createdDate. Соціальні зв’язки реалізують масиви followingUserIds і followerUserIds, а обрані треки - масив likedTrackIds з ідентифікаторами документів колекції tracks. Це дає змогу отримати весь перелік уподобань одним запитом, що оптимально для MVP-версії.

Колекція tracks містить документи аудіотреків. Ключові поля: title (назва), artistId (UID автора, що задає зв’язок «один до багатьох» із users), genre (жанр), audioOriginalFileUrl і coverArtUrl (повні публічні URL файлів у Firebase Storage, якими плеєр користується без додаткових запитів на генерацію токенів), plays (лічильник прослуховувань, що оновлюється атомарною increment(1)) та uploadedDate. Історію й коментарі винесено в підколекції: кожен документ підколекції users/{userId}/history містить trackId, timestamp, title та artist, а підколекцію tracks/{trackId}/comments зарезервовано

для майбутніх коментарів. Така ієрархія уникає надміру колекцій верхнього рівня й логічно групує пов’язані дані.

Правила безпеки у файлі firestore.rules реалізують модель доступу на основі власника. Для колекції users читання публічне, а створення й оновлення документа можливі лише за збігу request.auth.uid з ідентифікатором документа. Для колекції tracks читання публічне, а створення й видалення дозволено авторизованому авторові (artistId == request.auth.uid) або адміністраторові (роль «admin» у профілі). Доступ до підколекції history суворо обмежено власником профілю, причому записи незмінні. Це задовольняє вимоги безпеки NFR-03 – NFR-06. Заради продуктивності запитів, зокрема сортування історії за спаданням дати (orderBy('timestamp','desc')), застосовано автоматичні індекси Firestore, а для складніших аналітичних вибірок передбачено складені індекси на етапі розширення функціоналу.

Узагальнену структуру описаних колекцій і зв’язків між ними подано на рисунку 3.5 у вигляді діаграми класів, адаптованої для документоорієнтованої бази даних.

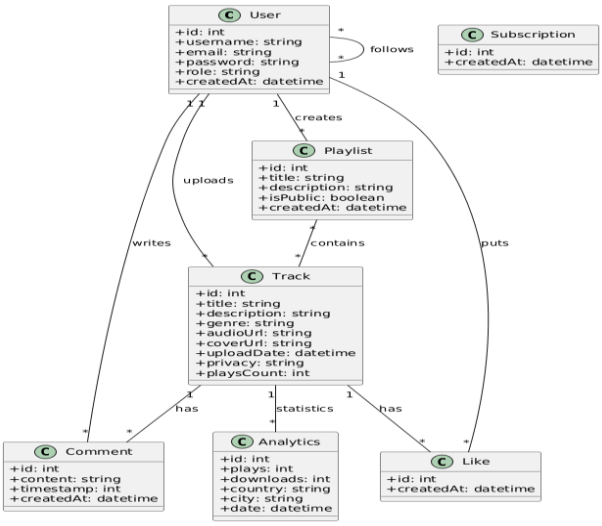

Рисунок 3.5 – Діаграма класів для колекцій Firestore

Наведена діаграма відображає основні сутності предметної області SoundFlex - користувача, трек та історію прослуховувань - разом із ключовими атрибутами й зв’язками, що прямо відповідають структурі колекцій Firestore.

Спроєктована в такий спосіб модель даних забезпечує цілісність інформації, зводить до мінімуму кількість запитів для отримання пов’язаних сутностей і створює надійне підґрунтя для подальшої програмної реалізації функціоналу платформи, описаної в розділі 4.

3.4 Проєктування інтерфейсу користувача

Проєктування інтерфейсу SoundFlex ґрунтується на засадах мінімалізму, адаптивності та інтуїтивної навігації, що визначені ключовими нефункціональними вимогами в підрозділі 2.3 (NFR-09, NFR-10) [19]. Інтерфейс реалізовано на бібліотеці готових компонентів shadcn/ui у поєднанні з утилітарним CSS-фреймворком Tailwind CSS - це дає візуальну єдність, високу швидкість верстки та відповідність стандартам доступності WCAG 2.1 AA.

Дерево компонентів застосунку (Component Tree) збудовано за принципом вкладеності, що його підтримує архітектура Next.js App Router. Кореневий елемент - файл layout.tsx, який задає сталий каркас для всіх сторінок. У ньому розміщено провайдери глобального стану - FirebaseClientProvider (контекст Firebase) і PlayerProvider (контекст аудіоплеєра). До постійних елементів інтерфейсу належать TopNav (головна навігація), PromoBanner (динамічний банер заохочення до переходу на преміум-план), StickyPlayer (закріплений аудіоплеєр із візуалізацією звукової хвилі Waveform) та Toaster (система спливних сповіщень). Вміст конкретної сторінки рендериться всередині макета через властивість children. Таке розташування гарантує, що плеєр лишається активним і не уриває відтворення під час навігації, відповідаючи вимозі FR-08.

Основні екрани платформи: головна (/), тренди (/trending), бібліотека (/library), профіль (/profile), завантаження треку (/upload), сторінка окремого треку (/track/[id]) та форми входу й реєстрації. Кожен має власну структуру, орієнтовану на певні користувацькі завдання.

Головна сторінка (рисунок 3.6) слугує точкою входу й утілює концепцію персоналізованого музичного простору. Угорі - навігаційна панель TopNav із логотипом, пунктами меню та аватаром користувача. Центр посідає блок HeroBanner із вітальним повідомленням, під яким - горизонтальний список треків. Нижче - секція New & Hot із картками популярних треків та альбомів. Унизу екрана постійно закріплено міні-плеєр StickyPlayer з обкладинкою, назвою поточного треку та кнопками керування. Усі інтерактивні елементи реалізовано компонентами shadcn/ui (Card, Button, Avatar, DropdownMenu), що тримає єдиний візуальний стиль.

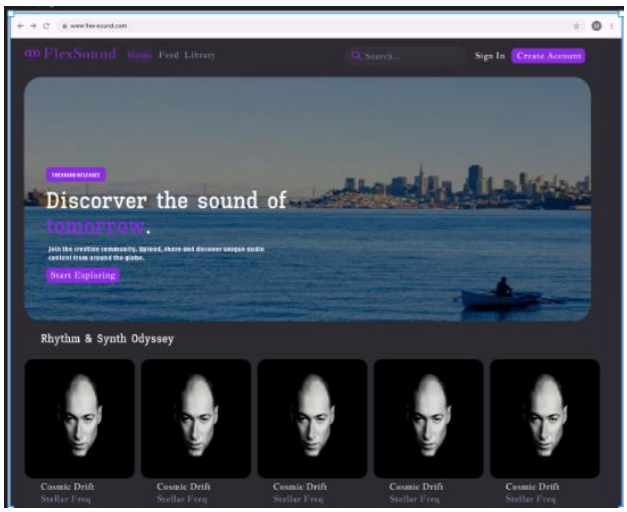

Рисунок 3.6 – Головна сторінка платформи SoundFlex

Сторінка завантаження треку (рисунок 3.7) дає авторам інструмент публікації нового контенту. Інтерфейс збудовано навколо форми з полями назви треку, вибору жанру з випадного списку та текстового опису. Окремо передбачено зону вибору аудіофайлу й обкладинки з пристрою. Заповнивши поля, автор натискає «Опублікувати», після чого з’являється індикатор прогресу завантаження. Форму реалізовано компонентами Input, Textarea, Select, Label і Dialog для попереднього перегляду.

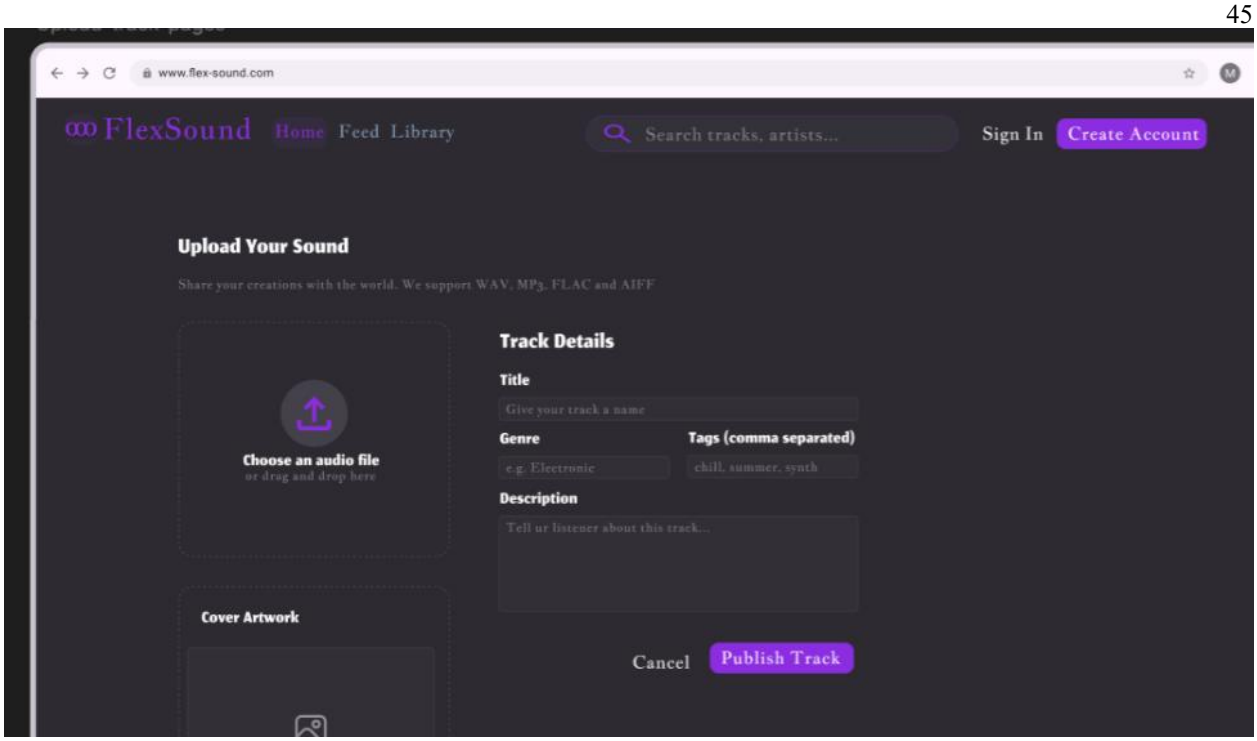

Рисунок 3.7 – Сторінка завантаження треку

Аудіоплеєр StickyPlayer (рисунок 3.8) - ключовий елемент інтерфейсу, що забезпечує безперервне відтворення музики. Він розташований унизу екрана й лишається видимим незалежно від поточної сторінки. Плеєр містить обкладинку поточного треку, його назву, ім’я автора, шкалу прогресу з перемотуванням, кнопки керування (відтворення/пауза, наступний/попередній трек) і візуалізацію звукової хвилі (Waveform). Реалізація на нативному HTML5 audio та React Context дає миттєву реакцію на дії користувача.

Рисунок 3.8 – Аудіоплеєр платформи SoundFlex

Адаптивність інтерфейсу забезпечують стандартні контрольні точки (breakpoints) Tailwind CSS: sm (640px), md (768px) та lg (1024px). На

смартфонах верхня навігація TopNav стає компактнішою - текстові мітки ховаються, лишаючи самі піктограми. Сітка карток треків змінюється з одноколонкової на мобільних до дво- чи триколонкової на планшетах і десктопах (grid-cols-1 md:grid-cols-2 lg:grid-cols-3). Плеєр на мобільних спрощується: ховається візуалізація хвилі, лишаються обкладинка, назва треку та кнопка відтворення/паузи. У правому верхньому куті - аватар авторизованого користувача, а клік по ньому відкриває випадне меню (DropdownMenu) із посиланнями на профіль, налаштування та кнопкою «Upgrade to Pro» для власників безкоштовного плану.

Отже, спроєктований інтерфейс цілком відповідає вимогам NFR-08 (адаптивність), NFR-09 (мінімалістичність та інтуїтивна зрозумілість) і NFR-10 (доступність) та створює візуальну основу для програмної реалізації клієнтської частини, яку детально описано в розділі 4.

Завершуючи розгляд проєктних рішень, від логіки використання та інтерфейсу варто перейти до інфраструктурного погляду на систему. Фізичне розміщення складових SoundFlex - клієнтської частини у браузері користувача та хмарних сервісів Firebase (Authentication, Firestore, Storage, Hosting) - унаочнює діаграма розгортання, наведена на рисунку 3.9. Вона завершує проєктний розділ, показуючи, на яких вузлах виконуються спроєктовані вище компоненти та якими каналами вони обмінюються даними під час роботи застосунку.

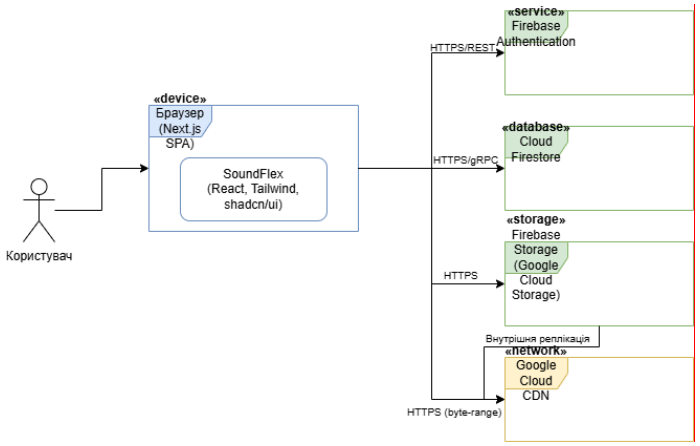

Рисунок 3.9 – Діаграма розгортання системи SoundFlex

Подана діаграма розгортання зводить докупи всі попередні проєктні рішення: схарактеризовані вище актори, сценарії, структури даних та елементи інтерфейсу дістають конкретну прив’язку до вузлів виконання - клієнтського браузера й керованих сервісів Firebase, - що остаточно окреслює архітектурну цілісність системи перед її програмною реалізацією.

Висновки до розділу 3

У третьому розділі кваліфікаційної роботи виконано повний цикл проєктування вебзастосунку дистрибуції аудіоконтенту SoundFlex. На основі функціональних і нефункціональних вимог попереднього розділу розроблено архітектуру системи, змодельовано ключові бізнес-процеси, спроєктовано логічну структуру бази даних та інтерфейс користувача.

Основою архітектури обрано зв’язку односторінкового застосунку (SPA) на фреймворку Next.js і хмарної платформи Firebase, що працює за моделлю Backend-as-a-Service. Така комбінація дала змогу свідомо відмовитися від проєктування й супроводу власного серверного API, перенісши відповідальність за автентифікацію, зберігання та масштабування на готову інфраструктуру постачальника. Завдяки цьому основні зусилля було зосереджено на клієнтській логіці, реактивності інтерфейсу та зручності взаємодії, що відповідає висунутим вимогам до легковагового й масштабованого рішення.

Поведінку системи описано засобами мови UML. Відправною точкою стала діаграма варіантів використання як найбільш узагальнена модель, що окреслює коло дійових осіб і доступних функцій. Вона зафіксувала чотирьох акторів - Гостя, Слухача, Автора та Адміністратора, - розмежувала їхні повноваження й відобразила відношення між елементами (асоціацію, узагальнення, «include» та «extend») у коректній нотації. Саме ця діаграма заклала підґрунтя для подальшого розмежування прав доступу. Похідними від неї побудовано діаграми послідовності для трьох ключових сценаріїв:

реєстрації користувача, завантаження треку автором і потокового відтворення із одночасним підрахунком прослуховувань.

Сховище даних спроєктовано на базі документоорієнтованої СУБД Cloud Firestore з урахуванням вимог до продуктивності та переваг гнучкої NoSQL-моделі. Виокремлено дві основні колекції верхнього рівня - users і tracks, - а також підколекції для зберігання історії прослуховувань і коментарів. Захист інформації забезпечено правилами безпеки firestore.rules, що діють безпосередньо на рівні хмарної інфраструктури й унеможливлюють несанкціонований доступ до чужих даних. Свідомо застосована денормалізація з вбудованими масивами ідентифікаторів likedTrackIds пришвидшує отримання обраного контенту, позбавляючи систему зайвих запитів до бази.

Завершальним аспектом стало проєктування інтерфейсу користувача. Побудовано дерево компонентів на основі маршрутизації Next.js App Router та сформульовано принципи адаптивного дизайну з використанням Tailwind CSS. Застосування бібліотеки готових компонентів shadcn/ui дозволило досягти візуальної цілісності продукту й відповідності рекомендаціям доступності WCAG 2.1 AA. Підсумовує розділ діаграма розгортання, що зіставляє спроєктовані програмні компоненти з вузлами їх фактичного виконання у браузері клієнта та хмарному середовищі Firebase.

Отже, сукупність розроблених архітектурних рішень, UML-діаграм, моделі даних і макетів інтерфейсу формує цілісну та внутрішньо узгоджену проєктну базу. Її безпосередньо покладено в основу наступного розділу, у якому виконується програмна реалізація клієнтської частини вебзастосунку SoundFlex.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

4.1 Структура проєкту та реалізація клієнтської частини

Програмна реалізація вебзастосунку SoundFlex виконана на основі архітектурних рішень, описаних у розділі 3, та повністю відповідає сформованій раніше специфікації вимог. Кодова база написана мовою TypeScript із застосуванням фреймворку Next.js 15 (App Router) і бібліотеки React 19, а оформлення інтерфейсу побудоване на Tailwind CSS у поєднанні з набором компонентів shadcn/ui. Для скорочення шляхів імпорту в конфігурації tsconfig.json налаштовано аліас @/*, що відображається на каталог ./src, тому всередині коду використовуються звернення виду @/components замість громіздких відносних шляхів.

Проєкт організовано за принципом розділення відповідальностей: кожен каталог об’єднує файли єдиного функціонального призначення. Маршрути та сторінки зосереджено в каталозі src/app, повторно використовувані елементи інтерфейсу - у src/components, шар взаємодії з хмарним бекендом - у src/firebase, серверні ШІ-функції - у src/ai, а допоміжні утиліти й типи даних - у src/lib. Така модульна структура спрощує навігацію по коду й полегшує подальше масштабування. Загальний вигляд дерева каталогів наведено на рисунку 4.1.

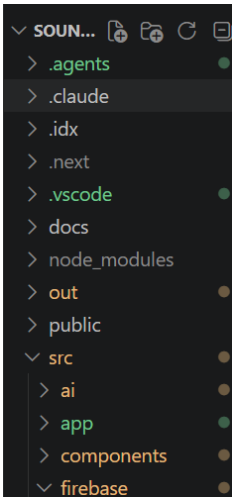

Рисунок 4.1 – Структура каталогів проєкту SoundFlex

Точкою входу застосунку є кореневий макет layout.tsx. Саме він формує сталий каркас, який зберігається незмінним під час переходів між сторінками: верхню навігаційну панель TopNav, рекламний банер PromoBanner, закріплений знизу плеєр StickyPlayer та систему спливаючих сповіщень Toaster. Усе дерево компонентів обгорнуте у два провайдери глобального стану - FirebaseClientProvider (доступ до сервісів Firebase) і PlayerProvider (стан відтворення). Завдяки розташуванню плеєра поза зоною перерендерення сторінок аудіо не переривається при навігації, що реалізує функціональну вимогу FR-08.

Ініціалізацію Firebase винесено в окремий модуль src/firebase/index.ts, де застосовано патерн Singleton. Функція initializeFirebase перевіряє масив уже створених застосунків через getApps() і створює новий екземпляр лише за його відсутності; це запобігає повторній ініціалізації під час гарячої заміни модулів (Hot Module Replacement) у середовищі розробки. Відповідний фрагмент коду наведено в коді нижче.

```
export function initializeFirebase() {
  if (getApps().length) {
    return getSdks(getApp());
  }
  const app = initializeApp(firebaseConfig);
  return getSdks(app);
}

function getSdks(app: FirebaseApp) {
  return {
    firebaseApp: app,
    auth: getAuth(app),
    firestore: getFirestore(app),
  };
}
```

Доступ до даних реалізовано через набір власних хуків useCollection і useDoc, які є обгортками над методами Firebase SDK і підписуються на зміни через onSnapshot (патерн Observer). Захист маршрутів виконується на клієнтському рівні: сторінки, що потребують авторизації (наприклад, /profile або /upload), використовують хук useUser() і за відсутності активної сесії перенаправляють відвідувача на форму входу або показують відповідне повідомлення. Перелік основних типів даних і фасадних модулів запису, що становлять програмну специфікацію застосунку, узагальнено в таблиці 4.1.

Таблиця 4.1 – Основні модулі та типи даних застосунку SoundFlex

Модуль / тип	Розташування	Призначення
Track	src/lib/mock-data.ts	Інтерфейс треку: title, artistId, genre, audioURL, coverArtUrl, plays, likes, uploadedDate
users.ts	src/firebase/firestore	Оновлення полів профілю користувача (updateUserProfile)
tracks.ts	src/firebase/firestore	Оновлення, видалення треку та атомарний інкремент лічильника прослуховувань
likes.ts	src/firebase/firestore	Перемикання вподобань (toggleTrackLike) зі зміною лічильника на ±1
history.ts	src/firebase/firestore	Додавання запису до підколекції історії прослуховувань
PlayerContext	src/components/AudioPlayer	Глобальний стан плеєра: поточний трек, черга, гучність, прогрес

Описана структура та перелічені модулі утворюють кістяк клієнтської частини, на який спираються решта підсистем застосунку. Подальші підрозділи послідовно розкривають їхню реалізацію - від авторизації до потокового відтворення аудіо.

4.2 Реалізація модуля авторизації та управління профілем

Модуль авторизації є відправною точкою для більшості функцій платформи, тому під час реалізації особливу увагу приділено надійності та зручності цього процесу. Реєстрація й вхід побудовані на сервісі Firebase Authentication і охоплюють два способи: класичну пару «електронна пошта - пароль» та вхід через обліковий запис Google. Форма реєстрації (src/app/register/page.tsx) перевіряє збіг пароля й підтвердження в реальному часі, після чого викликає createUserWithEmailAndPassword, а форма входу (src/app/login/page.tsx) - signInWithEmailAndPassword. Вигляд сторінки авторизації показано на рисунку 4.3.

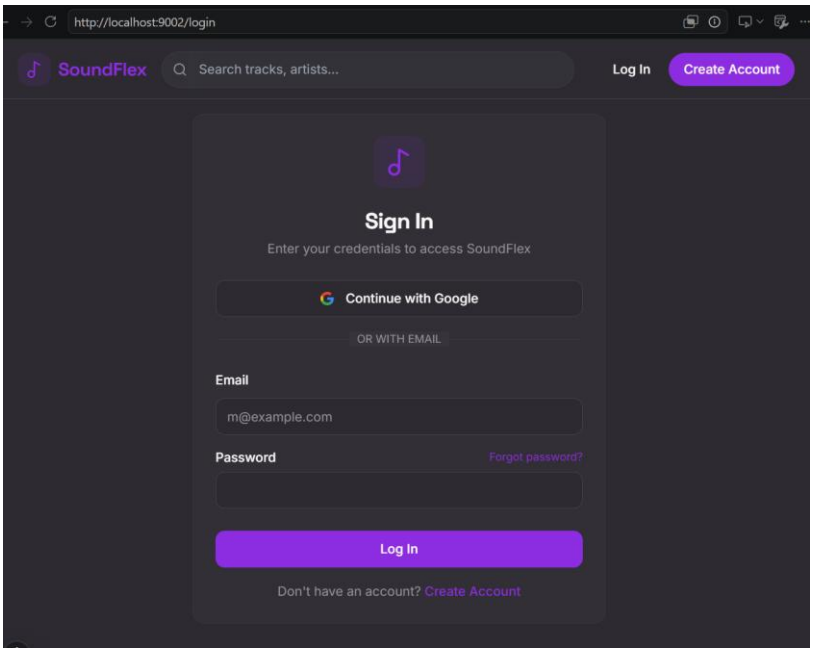

Рисунок 4.2 – Сторінка авторизації застосунку SoundFlex

Під час тестування реєстрації було виявлено, що стандартні коди помилок Firebase (наприклад, auth/user-not-found чи auth/too-many-requests) є малозрозумілими для кінцевого користувача, а в окремих випадках провайдер електронної пошти взагалі повертав узагальнене повідомлення «registration failed». Для усунення цього недоліку реалізовано функцію getAuthErrorMessage, яка перекладає технічні коди у природні україномовні пояснення. Її фрагмент наведено в коді нижче.

```
function getAuthErrorMessage(code: string): string {
  switch (code) {
    case 'auth/user-not-found':
      return 'Користувача з такою поштою не знайдено';
    case 'auth/wrong-password':
      return 'Невірний пароль. Спробуйте ще раз';
    case 'auth/email-already-in-use':
      return 'Ця електронна пошта вже зареєстрована';
    case 'auth/too-many-requests':
      return 'Забагато спроб. Зачекайте та повторіть пізніше';
    default:
      return 'Сталася помилка авторизації';
  }
}
```

Окремою проблемою стала ситуація, коли обліковий запис у Firebase Auth створювався успішно, але запис документа в Firestore тимчасово не проходив через помилку доступу. Щоб користувач не бачив хибного повідомлення про невдалу реєстрацію, операції створення акаунта та запису профілю розділено на окремі блоки try/catch. Спільна функція createUserDoc

формує документ users/{uid} з полями за замовчуванням (plan зі значенням «free», порожні масиви followers, following та likedTrackIds) і попередньо перевіряє, чи документ уже існує, що унеможливлює перезапис під час повторного входу через Google. Після успішної реєстрації викликається sendEmailVerification для підтвердження пошти, а сторінка /forgot-password використовує sendPasswordResetEmail для відновлення доступу.

Сторінка профілю (src/app/profile/page.tsx) є найбільшою за обсягом і відповідає за управління обліковим записом та тарифним планом. Вона містить майстер зміни підписки, що послідовно проходить чотири стани: selection (вибір плану), payment (введення платіжних даних), processing (імітація обробки з затримкою) та success (підтвердження). Функція startUpgrade ініціює процес, а handleProcessPayment записує новий план у Firestore. Під час реалізації було виправлено суттєвий дефект: кнопка вибору плану «SoundFlex Pro» зберігала неіснуюче значення «premium», тоді як система оперує лише значеннями free, pro та master. Після виправлення зберігається коректне значення pro, що відновило правильну роботу перевірок доступу до преміум-функцій.

4.3 Реалізація управління аудіоконтентом

Управління аудіоконтентом охоплює завантаження треків авторами, збереження файлів, генерацію метаданих і обкладинок та подальшу взаємодію слухачів із контентом. Сторінка завантаження (src/app/upload/page.tsx) надає форму з полями назви, жанру й опису, а також зонами вибору аудіофайлу та зображення обкладинки. Перед завантаженням функція getAudioDuration обчислює тривалість треку через нативний HTML5-об’єкт Audio, а handleFileChange автоматично підставляє назву -

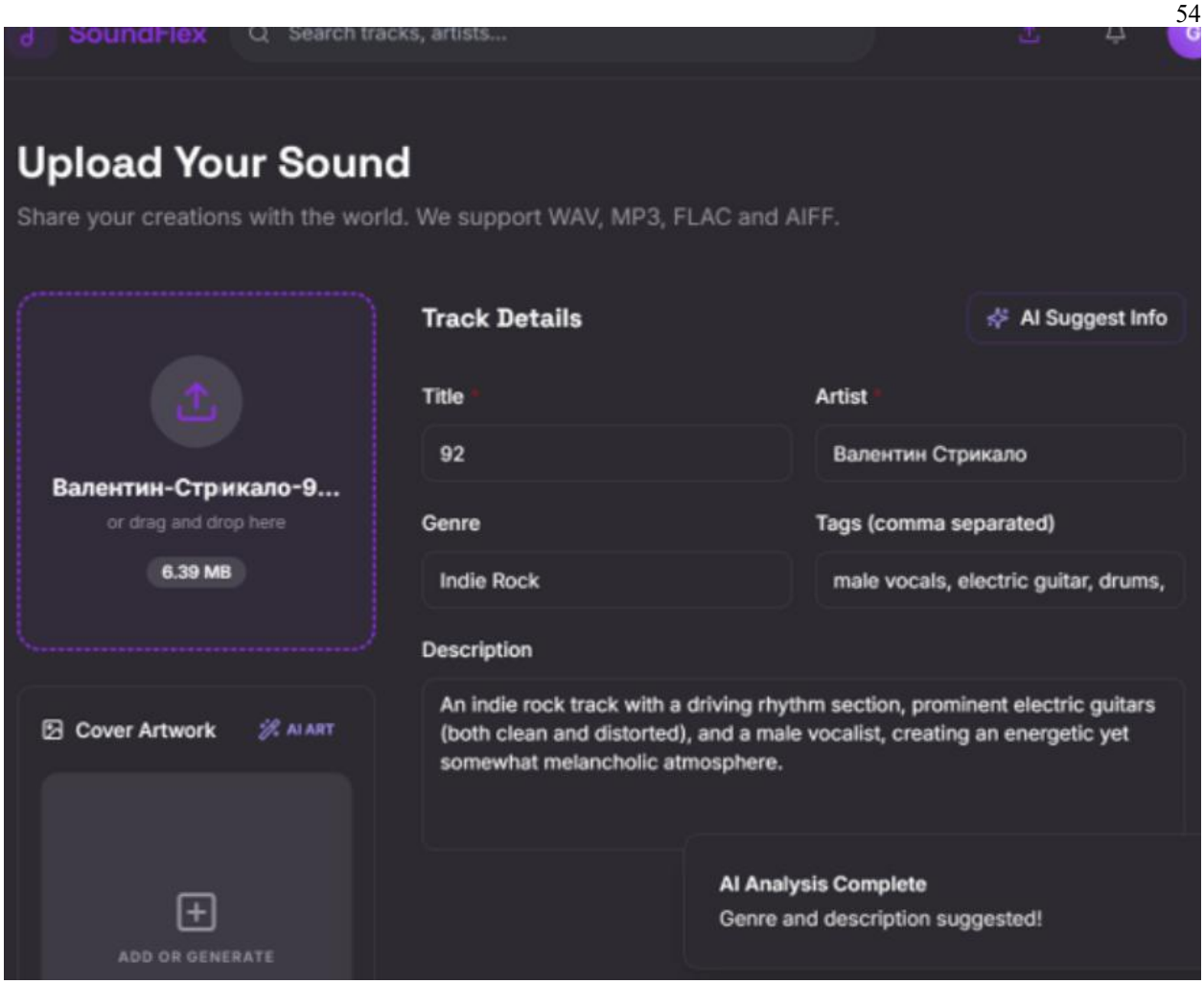

Рисунок 4.3 – Сторінка завантаження треку

У розділі 3 серверне сховище проєктувалося на базі Firebase Storage. Однак на етапі реалізації з’ясувалося, що цей сервіс потребує платного тарифного плану Blaze, тому для MVP-версії застосунку розроблено власний механізм збереження файлів через серверний маршрут Next.js. Обробник /api/upload (src/app/api/upload/route.ts) приймає дані у форматі multipart/form-data, перевіряє MIME-тип за білим списком дозволених форматів, генерує унікальне ім’я файлу на основі мітки часу та зберігає його у каталозі public/uploads через модуль fs/promises. У відповідь повертається публічний URL виду /uploads/audio/1700000000_song.mp3, який Next.js віддає напряму як статичний ресурс. Архітектура застосунку при цьому залишається готовою до переходу на Firebase Storage без зміни клієнтської логіки. Ключовий фрагмент обробника показано в коді нижче.

```
export async function POST(request: Request) {
  const formData = await request.formData();
  const file = formData.get('file') as File;
  const type = formData.get('type') as string;

  if (!ALLOWED_TYPES[type]?.includes(file.type)) {
    return NextResponse.json({ error: 'Недозволений тип файлу' },
                             { status: 400 });
  }
  const safeName = Date.now() + '_' + sanitize(file.name);
  const buffer = Buffer.from(await file.arrayBuffer());
  await writeFile(`./public/uploads/${type}/${safeName}`, buffer);

  return NextResponse.json({ url: `/uploads/${type}/${safeName}` });
}
```

Важливою особливістю модуля завантаження є інтеграція зі штучним інтелектом на базі Genkit. Функція suggestAudioMetadata передає аудіофайл у вигляді data URI до моделі Gemini 2.5 Flash, яка «прослуховує» запис і пропонує жанр, до п’яти тегів та короткий опис. Функція generateTrackCover викликає модель Imagen 4 для створення унікальної обкладинки за назвою й жанром треку, причому промпт явно забороняє додавати текст або цифри на зображення. Для коректної роботи плагіна @genkit-ai/google-genai у файлі .env додано дві змінні - GEMINI_API_KEY та GOOGLE_GENAI_API_KEY, оскільки плагін очікує саме другу назву, а раніше через її відсутність ШІ-функції мовчки не спрацьовували.

Після підготовки даних функція handleUpload виконує повний цикл публікації: завантажує аудіофайл через uploadFileLocally, за потреби зберігає згенеровану обкладинку, формує об’єкт trackData з усіма метаданими та лічильниками й створює документ у колекції tracks через addDoc, після чого додає ідентифікатор нового треку до масиву uploadedTrackIds користувача. Для безкоштовного плану діє обмеження isFreePlanLimit - не більше трьох опублікованих треків. Взаємодію слухачів із контентом реалізовано функцією toggleTrackLike, яка атомарно змінює лічильник likes на ±1 і додає або вилучає ідентифікатор треку з масиву likedTrackIds через arrayUnion та arrayRemove. Для захисту від накрутки у файлі firestore.rules додано окреме правило, що дозволяє авторизованому користувачеві змінювати лічильник лише на одиницю. Сторінки пошуку (/search) і рейтингу (/trending) працюють із реальними даними Firestore: перша виконує клієнтську фільтрацію за назвою,

автором, жанром і тегами, а друга сортує треки за кількістю прослуховувань із посторінковим довантаженням.

4.4 Реалізація аудіоплеєра та механізму потокового відтворення

Аудіоплеєр є центральним елементом застосунку, тому його реалізовано як окремий контекст PlayerContext, доступний з будь-якої точки дерева компонентів. В основі плеєра лежить стандартний HTML5-елемент <audio>, доступ до якого здійснюється через React-реф (useRef) усередині PlayerProvider. Свідома відмова від сторонніх бібліотек на кшталт Howler.js дозволила зменшити розмір клієнтського бандла та забезпечити максимальну сумісність із браузерами. Контекст зберігає поточний трек, чергу відтворення, рівень гучності, прогрес і прапорець якості isLossless, а також надає функції togglePlay, seek, setVolume, skipNext і skipPrevious. Вигляд закріпленого плеєра StickyPlayer наведено на рисунку 4.7.

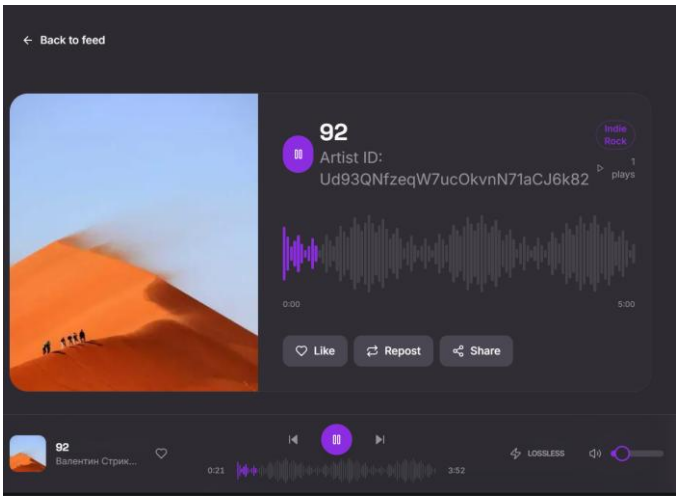

Рисунок 4.4 – Аудіоплеєр платформи SoundFlex

Запуск відтворення відбувається через функцію playTrack. Перед стартом вона перевіряє ліміт безкоштовного плану (не більше десяти треків за сесію, що зберігається у sessionStorage), додає запис до історії прослуховувань через addToHistory та збільшує лічильник прослуховувань треку атомарною функцією incrementTrackPlays. Завдяки реактивним підпискам onSnapshot

зміна лічильника миттєво відображається в інтерфейсі без перезавантаження сторінки. Спрощений фрагмент функції наведено в коді нижче.

```
const playTrack = (track: Track) => {
  if (profile?.plan === 'free' && playCount >= FREE_LIMIT) {
    setShowPremiumModal(true);
    return;
  }
  addToHistory(firestore, user.uid, track);
  incrementTrackPlays(firestore, track.id);

  const src = isLossless ? track.losslessURL : track.audioURL;
  audioRef.current.src = src;
  audioRef.current.play();
  setCurrentTrack(track);
  setIsPlaying(true);
};
```

У плеєрі реалізовано перемикання якості звуку через функцію toggleLossless. На етапі тестування виявилося, що відтворення у форматі FLAC не вмикалося для жодного користувача, оскільки перевірка прав доступу шукала неіснуюче значення плану «premium». Після заміни умови на коректну перевірку plan === «pro» || plan === «master» функція запрацювала правильно. У разі досягнення ліміту безкоштовних прослуховувань або спроби ввімкнути lossless без відповідної підписки з’являється модальне вікно з пропозицією апгрейду. Компонент Waveform візуалізує звукову хвилю треку: він генерує детермінований набір стовпчиків на основі ідентифікатора треку як зерна псевдовипадкових значень, завдяки чому хвиля виглядає однаково при кожному показі того самого запису та підтримує перехід по треку кліком.

Саме відтворення спирається на прогресивне завантаження файлів із підтримкою часткових HTTP-запитів (byte-range), що було обґрунтовано в підрозділі 1.3. Такий підхід дає змогу розпочинати прослуховування ще до повного завантаження файлу та виконувати перемотування, не вимагаючи складної серверної обробки. Окремо створено компонент global-error.tsx, який раніше був відсутній: через це за помилок доступу до Firestore застосунок падав у «білий екран». Новий обробник перехоплює критичні помилки рендеру й показує користувачеві зрозуміле повідомлення разом із кнопкою повторної спроби.

4.5 Тестування вебзастосунку та аналіз результатів

Для перевірки працездатності та відповідності реалізованого функціоналу поставленим вимогам обрано метод ручного функціонального тестування у поєднанні з валідацією нефункціональних характеристик за допомогою інструментів вимірювання продуктивності браузера [20]. Такий підхід є раціональним для проєкту рівня MVP, оскільки дозволяє охопити реальні сценарії взаємодії користувача без значних витрат на побудову повного автоматизованого тестового набору. Додатково коректність ШІ-флоу перевірялася на рівні схем валідації Zod, що описують структуру вхідних і вихідних даних кожної функції Genkit.

На основі специфікації функціональних вимог (підрозділ 2.3) сформовано перелік тест-кейсів, кожен з яких описує початкові умови, виконувані дії та очікуваний результат. Зведені результати функціонального тестування наведено в таблиці 4.2.

Таблиця 4.2 – Результати функціонального тестування

Код	Сценарій	Очікуваний результат	Статус
ТС-01	Реєстрація через e-mail та пароль	Створено акаунт і документ профілю	Пройдено
ТС-02	Вхід через Google-акаунт	Сесію відкрито, профіль створено за потреби	Пройдено
ТС-03	Завантаження аудіофайлу з обкладинкою	Трек збережено, документ додано в Firestore	Пройдено
ТС-04	Генерація метаданих засобами ШІ	Отримано жанр, теги та опис	Пройдено
ТС-05	Потокове відтворення з перемотуванням	Відтворення стартує, перемотування працює	Пройдено
ТС-06	Додавання треку у «Вибране»	Лічильник likes змінено на +1	Пройдено
ТС-07	Пошук за назвою та автором	Відображено релевантні треки	Пройдено
ТС-08	Перевищення ліміту безкоштовного плану	Показано вікно пропозиції апгрейду	Пройдено
ТС-09	Перемикання якості lossless (план pro)	Активовано відтворення у FLAC	Пройдено

Окрім функціональних сценаріїв, проведено вимірювання нефункціональних характеристик, передбачених вимогами NFR-01 та NFR-02. Час першого змістовного відображення сторінки (First Contentful Paint) і затримку буферизації перед стартом відтворення виміряно у вкладці Performance інструментів розробника браузера Google Chrome за умов широкосмугового з'єднання. Усереднені за десятьма вимірюваннями результати подано в таблиці 4.3.

Таблиця 4.3 – Результати вимірювання продуктивності

Показник	Вимога	Виміряне значення
First Contentful Paint (головна)	≤ 2,0 с	1,3 с
Буферизація перед відтворенням	≤ 1,5 с	0,9 с
Перехід між сторінками (SPA)	—	0,2–0,4 с
Час відгуку кнопки відтворення	—	до 0,1 с

Отримані значення підтверджують, що застосунок задовольняє висунуті вимоги до продуктивності: показник First Contentful Paint становить 1,3 с проти граничних 2,0 с, а затримка буферизації - 0,9 с проти 1,5 с. Низький час переходів між сторінками пояснюється клієнтською навігацією SPA та збереженням стану плеєра поза зоною перерендерення. У процесі тестування було виявлено й усунено низку дефектів, що мали критичний вплив на працездатність системи. Серед найсуттєвіших: декоративна (нефункціональна) поведінка кнопок лайків і поля пошуку, підстановка фіктивних даних на сторінках профілю, бібліотеки та рейтингу замість реальних, повністю імітоване завантаження треків, а також помилка збірки через зайвий експорт видаленого модуля. Після виправлень усі перелічені модулі переведено на роботу з реальними даними Firestore.

Підсумкову оцінку якості розробленого ПЗ проведено за атрибутами стандарту ISO/IEC 25010, на який спирається специфікація вимог. Результати оцінювання наведено в таблиці 4.4.

Таблиця 4.4 – Оцінка якості ПЗ за моделлю ISO/IEC 25010

Характеристика	Коментар	Оцінка
Функційна придатність	Реалізовано всі функції високого пріоритету (FR-01 – FR-10)	Висока
Продуктивність	Показники FCP та буферизації кращі за нормативні	Висока
Зручність використання	Мінімалістичний інтерфейс на компонентах shadcn/ui	Висока
Надійність	Додано обробку критичних помилок (global-error.tsx)	Достатня
Безпека	Доступ контролюється правилами firestore.rules та HTTPS	Достатня
Супроводжуваність	Модульна структура, патерни Singleton, Observer, Facade	Висока

Підсумовуючи, проведене тестування підтвердило коректність основних сценаріїв роботи та відповідність застосунку як функціональним, так і нефункціональним вимогам, а виявлені під час перевірки дефекти усунено ще до завершення розробки. Це дає підстави вважати реалізований функціонал готовим до практичного використання, порядок якого описано в наступному підрозділі.

4.6 Керівництво користувача

Робота з вебзастосунком SoundFlex не потребує встановлення додаткового програмного забезпечення - достатньо сучасного браузера з підтримкою HTML5. Після відкриття застосунку користувач потрапляє на головну сторінку, що містить вітальний банер, блок персоналізованих рекомендацій «Your Daily Mix» та стрічку нових треків «New & Hot». Загальний вигляд головної сторінки показано на рисунку 4.9.

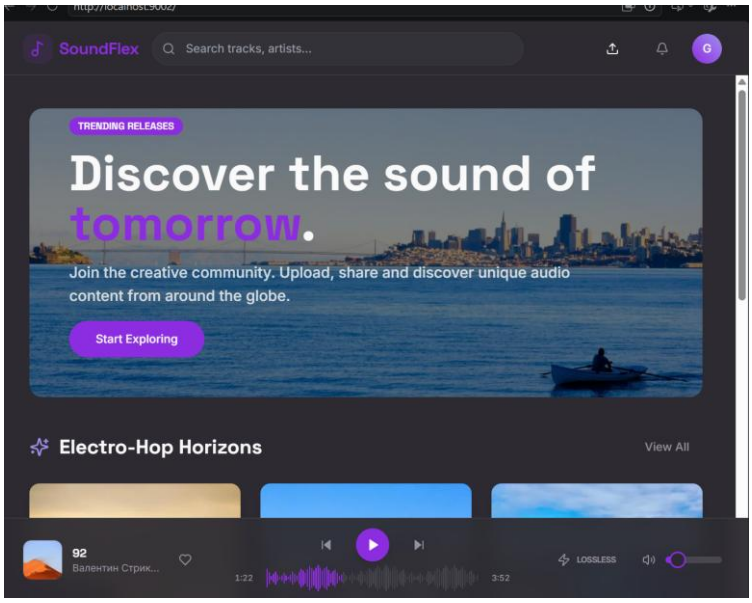

Рисунок 4.5 – Головна сторінка застосунку SoundFlex

Для отримання повного доступу до функцій користувачеві необхідно зареєструватися, натиснувши відповідну кнопку у верхній навігаційній панелі та заповнивши форму реєстрації, або скориставшись швидким входом через обліковий запис Google. Після авторизації стають доступними бібліотека, історія прослуховувань та можливість додавати треки у «Вибране».

Щоб опублікувати власний трек, автор переходить на сторінку завантаження, обирає аудіофайл і за бажанням зображення обкладинки, заповнює назву, жанр та опис. За потреби можна скористатися кнопками автоматичного підбору метаданих і генерації обкладинки засобами штучного інтелекту. Після натискання кнопки публікації відображається індикатор прогресу, а по завершенні трек з’являється у профілі автора та в загальній стрічці.

Відтворення запускається кліком по будь-якому треку у списку. Закріплений знизу екрана плеєр дозволяє ставити відтворення на паузу, перемикати треки, регулювати гучність, перемотувати запис за шкалою прогресу та (для власників відповідної підписки) вмикати режим відтворення без втрат. Розділ «Бібліотека» об’єднує нещодавно прослухане, вподобані треки та повну історію прослуховувань; його вигляд наведено на рисунку 4.10.

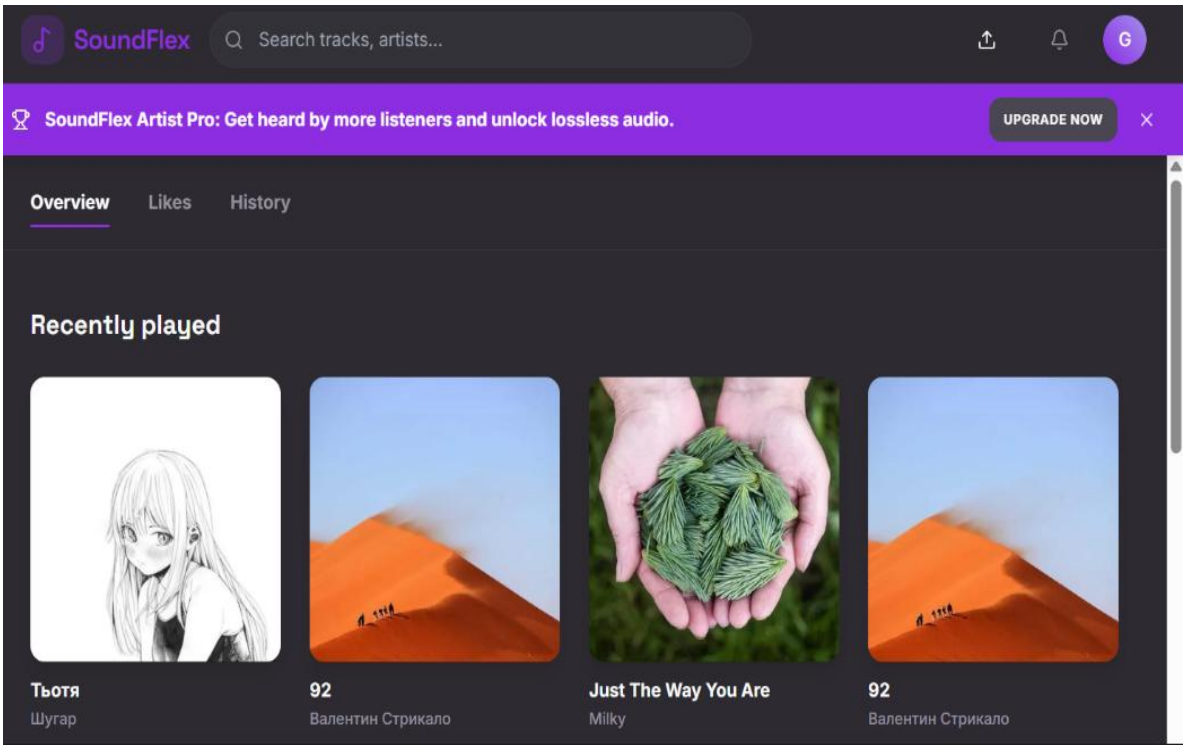

Рисунок 4.6 – Бібліотека користувача

Керування тарифним планом здійснюється на сторінці профілю через майстер зміни підписки, що послідовно проводить користувача від вибору плану до підтвердження. Безкоштовний план передбачає обмеження на кількість завантажуваних треків і періодичну рекламу, тоді як преміум-плани відкривають розширене сховище та відтворення у високій якості. Таким чином, інтерфейс застосунку залишається інтуїтивно зрозумілим на всіх етапах взаємодії, що відповідає вимогам зручності використання NFR-09 та NFR-10.

Висновки до розділу 4

У четвертому розділі кваліфікаційної роботи виконано програмну реалізацію та тестування вебзастосунку дистрибуції аудіоконтенту SoundFlex відповідно до архітектурних рішень і специфікації вимог, сформованих у попередніх розділах. Реалізовано клієнтську частину на основі Next.js із App Router, налаштовано модульну структуру проєкту та застосовано перевірені шаблони проєктування - Singleton, Observer і Facade.

Розроблено й налагоджено ключові підсистеми застосунку: модуль авторизації з підтримкою входу через e-mail і Google та зрозумілою обробкою помилок, управління профілем із майстром зміни підписки, механізм завантаження аудіоконтенту з локальним сховищем через серверний маршрут /api/upload та інтеграцією ШІ-функцій Genkit, а також аудіоплеєр із потоковим відтворенням, підрахунком прослуховувань і перемиканням якості звуку. Під час реалізації виявлено та усунено низку суттєвих дефектів, зокрема некоректну перевірку тарифного плану, відсутність обробника критичних помилок і фіктивну поведінку окремих модулів.

Проведене функціональне тестування підтвердило коректну роботу всіх основних сценаріїв, а вимірювання продуктивності засвідчили, що показники First Contentful Paint (1,3 с) і буферизації (0,9 с) перевершують нормативні значення вимог NFR-01 та NFR-02. Оцінка якості ПЗ за моделлю ISO/IEC 25010 показала високий рівень функційної придатності, продуктивності, зручності використання й супроводжуваності. Підготовлене керівництво користувача описує повний порядок роботи із застосунком - від реєстрації до публікації та відтворення треків. Отримані результати свідчать про досягнення поставленої мети та готовність реалізованого вебзастосунку до подальшого розвитку й розгортання у промисловому середовищі.

ВИСНОВКИ

У межах кваліфікаційної бакалаврської роботи розроблено вебзастосунок дистрибуції аудіоконтенту SoundFlex, що автоматизує процес доставки медіаконтенту користувачам. Поставлену мету досягнуто, а всі визначені у вступі завдання виконано в повному обсязі. У ході роботи:

1. проаналізовано сучасний стан сфери вебдистрибуції аудіоконтенту та розглянуто провідні платформи SoundCloud, Spotify і YouTube Music; виявлено їхні сильні сторони й обмеження, зокрема закритість екосистем і складність публікації для незалежних авторів, на основі чого сформульовано концепцію власної легковагової системи, орієнтованої саме на цю аудиторію;

2. досліджено методи організації потокового передавання аудіоданих через вебсередовище та обґрунтовано доцільність прогресивного завантаження з підтримкою часткових HTTP-запитів (byte-range) у поєднанні з кешуванням через мережу доставки контенту, що дає змогу розпочинати відтворення ще до повного завантаження файлу та вільно перемотувати запис без складної серверної обробки;

3. обґрунтовано вибір архітектури та технологій реалізації системи: визначено клієнт-серверну модель на основі фреймворку Next.js (React) у поєднанні з підходом Backend-as-a-Service, що дало змогу побудувати масштабований застосунок без власної серверної інфраструктури;

4. спроєктовано логічну структуру бази даних на основі документоорієнтованої моделі Cloud Firestore: визначено дві колекції верхнього рівня (users і tracks), підколекції для історії прослуховувань і коментарів, а також правила доступу firestore.rules, що реалізують модель безпеки на основі власника; денормалізована структура з вбудованими масивами ідентифікаторів забезпечує отримання пов’язаних даних мінімальною кількістю запитів;

5. реалізовано механізми автоматизації завантаження та відтворення аудіофайлів: розроблено серверний маршрут збереження файлів, інтегровано штучний інтелект на базі Genkit для автоматичного добору метаданих і генерації обкладинок, а також побудовано аудіоплеєр із безперервним фоновим відтворенням, атомарним підрахунком прослуховувань і перемиканням якості звуку;

6. забезпечено систему авторизації та розмежування доступу: автентифікацію реалізовано засобами Firebase Authentication із входом за електронною поштою та через обліковий запис Google, а розподіл повноважень між чотирма ролями - гостем, слухачем, автором і адміністратором - закріплено правилами безпеки на хмарній інфраструктурі.

Працездатність застосунку підтверджено ручним функціональним тестуванням: усі дев’ять сформованих тест-кейсів пройдено успішно. Вимірювання продуктивності засвідчили, що показники First Contentful Paint (1,3 с) і затримки буферизації (0,9 с) перевершують нормативні значення вимог. Під час реалізації виявлено та усунено низку суттєвих дефектів, зокрема некоректну перевірку тарифного плану, відсутність обробника критичних помилок і фіктивну поведінку окремих модулів.

Практична цінність роботи полягає в тому, що отриманий вебзастосунок є завершеним легковаговим і масштабованим стрімінговим сервісом, який не потребує власної серверної інфраструктури завдяки моделі BaaS і може бути використаний авторами для публікації та поширення власного аудіоконтенту.

Подальший розвиток платформи доцільно спрямувати на повне редагування профілю та завантаження аватарів, реалізацію системи коментарів, перенесення файлового сховища у хмарне середовище для промислової експлуатації, а також на розширення аналітичних можливостей і впровадження автоматизованого тестування. Отримані результати свідчать про досягнення поставленої мети та готовність розробленого вебзастосунку до подальшого вдосконалення й розгортання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Фісенко Т., Балюн О., Краснюк У. Дослідження індустрії аудіовидавництва в Україні та за кордоном. Обрії друкарства. 2022. № 2 (12). С. 138–147. DOI: https://doi.org/10.20535/2522-1078.2022.2(12).270912

2. Pantos R., May W. HTTP Live Streaming : RFC 8216. Internet Engineering Task Force, 2017. 60 p. URL: https://www.rfc-editor.org/info/rfc8216 (date of Accessed: 12.05.2026).

3. ISO/IEC 23009-1:2022. Information technology - Dynamic adaptive streaming over HTTP (DASH) - Part 1: Media presentation description and segment formats. Geneva : ISO, 2022. 290 p.

4. Valin J.-M., Vos K., Terriberry T. Definition of the Opus Audio Codec : RFC 6716. Internet Engineering Task Force, 2012. 326 p. URL: https://www.rfc-editor.org/info/rfc6716 (date of Accessed: 12.05.2026).

5. Войтюк О. В. Аналіз методів та технологій рендерингу вебзастосунків. Вчені записки ТНУ імені В. І. Вернадського. Серія: Технічні науки. 2025. Т. 36, № 6, ч. 2. С. 74–82. DOI: https://doi.org/10.32782/2663-5941/2025.6.2/11

6. Next.js Documentation : Routing and Data Fetching. Vercel, 2024. URL: https://nextjs.org/docs (date of Accessed: 12.05.2026).

7. React Documentation. Meta Platforms, 2024. URL: https://react.dev (date of Accessed: 12.05.2026).

8. TypeScript Documentation : The TypeScript Handbook. Microsoft, 2024. URL: https://www.typescriptlang.org/docs (date of Accessed: 12.05.2026).

9. Tailwind CSS Documentation. Tailwind Labs, 2024. URL: https://tailwindcss.com/docs (date of Accessed: 12.05.2026).

10. Firebase Documentation : Build and scale data-driven apps with Cloud Firestore and Storage. Google, 2024. URL: https://firebase.google.com/docs (date of Accessed: 12.05.2026).

11. Merenstein A., Tarasov V., Anwar A. F3: Serving Files Efficiently in Serverless Computing. Proceedings of the 16th ACM International Conference on Systems and Storage (SYSTOR ’23). New York : ACM, 2023. P. 8–21.

12. Fielding R., Lafon Y., Reschke J. Hypertext Transfer Protocol (HTTP/1.1): Range Requests : RFC 7233. Internet Engineering Task Force, 2014. 25 p. URL: https://www.rfc-editor.org/info/rfc7233 (date of Accessed: 12.05.2026).

13. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol : O’Reilly Media, 2017. 616 p.

14. ISO/IEC/IEEE 29148:2018. Systems and software engineering - Life cycle processes - Requirements engineering. Geneva : ISO, 2018. 104 p.

15. ISO/IEC 25010:2023. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Product quality model. Geneva : ISO, 2023. 22 p.

16. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.

17. Мартін Р. Чиста архітектура. Мистецтво розробки програмного забезпечення. Харків : Фабула, 2022. 368 с.

18. OMG Unified Modeling Language (OMG UML). Version 2.5.1. Needham : Object Management Group, 2017. 796 p. URL: https://www.omg.org/spec/UML/2.5.1 (date of Accessed: 12.05.2026).

19. Чіверс Х. Адаптивний вебдизайн: створення ефективних інтерфейсів для сучасних пристроїв. Київ : Фабула, 2023. 320 с.

20. ISO/IEC/IEEE 29119-1:2022. Software and systems engineering - Software testing - Part 1: General concepts. Geneva : ISO, 2022. 67 p.

ДОДАТОК А

Конфігураційні файли та правила безпеки проєкту

У додатку наведено основні конфігураційні файли вебзастосунку SoundFlex та правила безпеки хмарної інфраструктури Firebase, що реалізують модель доступу, описану в підрозділі 3.3 основної частини.

Лістинг А.1 – Правила безпеки бази даних (firestore.rules)

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {

    match /users/{userId} {
      allow read: if true;
      allow create, update: if request.auth != null
                           && request.auth.uid == userId;

      match /history/{historyId} {
        allow read, create: if request.auth != null
                           && request.auth.uid == userId;
        allow update, delete: if false;
      }
    }

    match /tracks/{trackId} {
      allow read: if true;
      allow create: if request.auth != null;
      allow delete: if request.auth != null
                    && resource.data.artistId == request.auth.uid;

      allow update: if request.auth != null
```

```
        && request.resource.data.diff(resource.data)
             .affectedKeys().hasOnly(['likes'])
        && (request.resource.data.likes == resource.data.likes + 1
         || request.resource.data.likes == resource.data.likes - 1);

      match /comments/{commentId} {
        allow read: if true;
        allow create: if request.auth != null;
      }
    }
  }
}
```

Лістинг А.2 – Правила доступу до файлового сховища (storage.rules)

```
rules_version = '2';
service firebase.storage {
  match /b/{bucket}/o {

    match /tracks/{userId}/{allPaths=**} {
      allow read: if true;
      allow write: if request.auth != null
                   && request.auth.uid == userId
                   && request.resource.size < 100 * 1024 * 1024
                   && request.resource.contentType.matches('audio/.*');
    }
    match /covers/{userId}/{allPaths=**} {
      allow read: if true;
      allow write: if request.auth != null
                   && request.auth.uid == userId
                   && request.resource.size < 5 * 1024 * 1024
                   && request.resource.contentType.matches('image/.*');
```

```
    }
  }
}
```

Лістинг А.3 – Складені індекси Firestore (firestore.indexes.json)

```
{
  "indexes": [
    {
      "collectionGroup": "tracks",
      "queryScope": "COLLECTION",
      "fields": [
        { "fieldPath": "artistId", "order": "ASCENDING" },
        { "fieldPath": "uploadedDate", "order": "DESCENDING" }
      ]
    },
    {
      "collectionGroup": "tracks",
      "queryScope": "COLLECTION",
      "fields": [
        { "fieldPath": "plays", "order": "DESCENDING" }
      ]
    }
  ],
  "fieldOverrides": []
}
```

Лістинг А.4 – Конфігурація збірки (next.config.ts)

```
import type { NextConfig } from 'next';

const nextConfig: NextConfig = {
  typescript: { ignoreBuildErrors: true },
  eslint: { ignoreDuringBuilds: true },
```

```
  images: {
    remotePatterns: [
      { protocol: 'https', hostname: 'placehold.co' },
      { protocol: 'https', hostname: 'images.unsplash.com' },
      { protocol: 'https', hostname: 'picsum.photos' },
    ],
  },
};

export default nextConfig;
```

ДОДАТОК Б

Лістинги серверних та клієнтських модулів

У додатку подано повні лістинги ключових модулів вебзастосунку, що реалізують серверне збереження файлів, операції з базою даних та неблокувальне оновлення інтерфейсу. Ці модулі деталізують програмну реалізацію, описану в розділі 4 основної частини.

Лістинг Б.1 – Серверний маршрут збереження файлів (src/app/api/upload/route.ts)

```
import { NextResponse } from 'next/server';
import { writeFile, mkdir } from 'fs/promises';
import path from 'path';

const ALLOWED = {
  audio: ['audio/mpeg', 'audio/wav', 'audio/flac', 'audio/aac',
'audio/ogg'],
  covers: ['image/jpeg', 'image/png', 'image/webp'],
};

export async function POST(request: Request) {
  const formData = await request.formData();
  const file = formData.get('file') as File;
  const type = formData.get('type') as 'audio' | 'covers';

  if (!file || !ALLOWED[type]?.includes(file.type)) {
    return NextResponse.json({ error: 'Недозволений тип файлу' },
                             { status: 400 });
  }

  const ext = file.name.split('.').pop();
```

```
  const safeName = `${Date.now()}_${file.name
    .replace(/[^a-zA-Z0-9._-]/g, '_')}`;
  const dir = path.join(process.cwd(), 'public', 'uploads', type);
  await mkdir(dir, { recursive: true });

  const buffer = Buffer.from(await file.arrayBuffer());
  await writeFile(path.join(dir, safeName), buffer);

  return NextResponse.json({ url: `/uploads/${type}/${safeName}` });
}
```

Лістинг Б.2 – Операції з треками: інкремент прослуховувань (src/firebase/firestore/tracks.ts)

```
import { doc, updateDoc, deleteDoc, increment }
  from 'firebase/firestore';

export async function incrementTrackPlays(db, trackId: string) {
  try {
    await updateDoc(doc(db, 'tracks', trackId), {
      plays: increment(1),
    });
  } catch (e) {
    console.error('incrementTrackPlays:', e);
  }
}

export async function updateTrack(db, trackId, data) {
  await updateDoc(doc(db, 'tracks', trackId), data);
}

export async function deleteTrack(db, trackId: string) {
```

```
  await deleteDoc(doc(db, 'tracks', trackId));
}
```

Лістинг Б.3 – Перемикання вподобань треку (src/firebase/firestore/likes.ts)

```
import { doc, updateDoc, increment,
         arrayUnion, arrayRemove } from 'firebase/firestore';

export async function toggleTrackLike(
  db, trackId: string, userId: string, isLiked: boolean
) {
  const trackRef = doc(db, 'tracks', trackId);
  const userRef  = doc(db, 'users', userId);

  await updateDoc(trackRef, {
    likes: increment(isLiked ? -1 : 1),
  });
  await updateDoc(userRef, {
    likedTrackIds: isLiked
      ? arrayRemove(trackId)
      : arrayUnion(trackId),
  });
}
```

Лістинг Б.4 – Запис історії прослуховувань (src/firebase/firestore/history.ts)

```
import { collection, addDoc, serverTimestamp }
  from 'firebase/firestore';

export async function addToHistory(db, userId: string, track) {
  const historyRef = collection(db, 'users', userId, 'history');
  await addDoc(historyRef, {
```

```
    trackId: track.id,
    title: track.title,
    artist: track.artist,
    listenedAt: serverTimestamp(),
  });
}
```

Лістинг Б.5 – Неблокувальні оновлення та обробка помилок доступу (src/firebase/non-blocking-updates.tsx)

```
import { addDoc, updateDoc } from 'firebase/firestore';
import { errorEmitter } from './error-emitter';
import { FirestorePermissionError } from './errors';

export function addDocumentNonBlocking(ref, data) {
  addDoc(ref, data).catch((err) => {
    if (err.code === 'permission-denied') {
      errorEmitter.emit('permission-error',
        new FirestorePermissionError('create', ref.path));
    }
  });
}

export function updateDocumentNonBlocking(ref, data) {
  updateDoc(ref, data).catch((err) => {
    if (err.code === 'permission-denied') {
      errorEmitter.emit('permission-error',
        new FirestorePermissionError('update', ref.path));
    }
  });
}
```

Лістинг Б.6 – Детермінована візуалізація звукової хвилі (src/components/AudioPlayer/Waveform.tsx)

```
function seededBars(trackId: string, count = 80): number[] {
  let seed = 0;
  for (const ch of trackId) seed = (seed * 31 + ch.charCodeAt(0)) >>> 0;

  const bars: number[] = [];
  for (let i = 0; i < count; i++) {
    seed = (seed * 1103515245 + 12345) & 0x7fffffff;
    bars.push(0.2 + (seed / 0x7fffffff) * 0.8);
  }
  return bars;
}
```

ДОДАТОК В

Реалізація функцій на основі Genkit

У додатку наведено лістинги модулів, що реалізують інтеграцію зі штучним інтелектом Google через фреймворк Genkit. Ці функції забезпечують автоматичний добір метаданих, генерацію обкладинок та формування персоналізованих рекомендацій, згаданих у підрозділах 4.3 та 4.4 основної частини. Структуру вхідних і вихідних даних кожного флоу описано схемами валідації Zod.

Лістинг В.1 – Налаштування ШІ-клієнта (src/ai/genkit.ts)

```
import { genkit } from 'genkit';
import { googleAI } from '@genkit-ai/google-genai';

export const ai = genkit({
  plugins: [googleAI()],
  model: googleAI.model('gemini-2.5-flash'),
});
```

Лістинг В.2 – Добір метаданих аудіофайлу (src/ai/flows/ai-metadata-suggester.ts)

```
import { z } from 'genkit';
import { ai } from '@/ai/genkit';

const InputSchema = z.object({
  audioDataUri: z.string(),
  audioFileName: z.string(),
});
const OutputSchema = z.object({
  genre: z.string(),
  tags: z.array(z.string()).max(5),
```

```
  description: z.string(),
});


export async function suggestAudioMetadata(input) {
  const { output } = await ai.generate({
    prompt: [
      { text: 'Прослухай аудіо та визнач жанр, до 5 тегів і ' +
              'короткий опис українською.' },
      { media: { url: input.audioDataUri } },
    ],
    output: { schema: OutputSchema },
  });
  return output;
}
```

Лістинг В.3 – Генерація обкладинки треку (src/ai/flows/ai-image-generator.ts)

```
import { z } from 'genkit';
import { ai } from '@/ai/genkit';
import { googleAI } from '@genkit-ai/google-genai';


export async function generateTrackCover(input: {
  trackTitle: string; genre: string; description: string;
}) {
  const { media } = await ai.generate({
    model: googleAI.model('imagen-4.0-fast-generate-001'),
    prompt: `Обкладинка для треку "${input.trackTitle}" ` +
            `у жанрі ${input.genre}. Без тексту та цифр на зображенні.`,
  });
  return { imageUrl: media.url };
}
```

Лістинг В.4 – Персоналізовані рекомендації (src/ai/flows/personalized-track-recommendations.ts)

```
import { z } from 'genkit';
import { ai } from '@/ai/genkit';

const OutputSchema = z.object({
  mixTitle: z.string(),
  tracks: z.array(z.object({
    title: z.string(),
    artist: z.string(),
    reason: z.string(),
  })).min(5).max(7),
});

export async function generateTrackRecommendations(input: {
  listeningHistory: string[];
  likedTracks: string[];
  preferredGenres: string[];
}) {
  const { output } = await ai.generate({
    prompt: `На основі історії (${input.listeningHistory.join(', ')}), ` +
            `лайків (${input.likedTracks.join(', ')}) та жанрів ` +
            `(${input.preferredGenres.join(', ')}) сформуй добірку ` +
            `Daily Mix із 5-7 треків, поясни вибір кожного.`,
    output: { schema: OutputSchema },
  });
  return output;
}
```