

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук та технологій
(повне найменування інституту, факультету)

Кафедра комп'ютерних систем та мереж
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавра

(ступінь вищої освіти (освітній ступінь))

на тему **РОЗРОБКА КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ ЗБЕРІГАННЯ,
ПОШУКУ ТА ВІДТВОРЕННЯ МУЗИЧНИХ ФАЙЛІВ ТА ЗОБРАЖЕНЬ**

Виконав: студент 4 курсу, групи КНТ-512
спеціальності _____

123 Комп'ютерна інженерія

(код і найменування спеціальності)

Освітня програма (спеціалізація) _____

Комп'ютерна інженерія

ЛАХНО В.А.

(ПРИЗВИЩЕ та ініціали)

Керівник ЗЕЛЕНЬОВА І.Я.

(ПРИЗВИЩЕ та ініціали)

Рецензент ЩЕРБАКОВ В.І.

(ПРИЗВИЩЕ та ініціали)

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

Факультет Комп'ютерних наук і технологій
Кафедра Комп'ютерні системи та мережі
Ступінь вищої освіти бакалаврський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерна інженерія
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри КУДЕРМЕТОВ Р.К.

“ ” 2026 року

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

ЛАХНУ Вячеславу Андрійовичу

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка клієнт-серверної системи зберігання, пошуку та відтворення музичних файлів та зображень

керівник проєкту (роботи) к.т.н., доцент, ЗЕЛЕНЬОВА Ірина Яківна

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

затверджені наказом вищого навчального закладу від «14» квітня 2026 року № 160

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

3. Вихідні дані до проєкту (роботи) Існуючі клієнт-серверні рішення для зберігання та відтворення мультимедіа: Plex Media Server, Jellyfin, Emby. Принципи побудови тришарової клієнт-серверної архітектури з розділенням на рівні представлення, бізнес-логіки та доступу до даних. Технологічний стек серверної частини: Node.js 20, Express.js
4. Технологічний стек клієнтської частини: React 18, Tailwind CSS. Реляційна СУБД PostgreSQL 16 з підтримкою повнотекстового пошуку (TSVECTOR).

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області та сучасних рішень для зберігання і відтворення мультимедійного контенту;

2. Проектування клієнт-серверної системи зберігання, пошуку та відтворення мультимедіа;

3. Розробка клієнт-серверної системи зберігання, пошуку та відтворення мультимедіа;

4. Тестування клієнт-серверної системи зберігання, пошуку та відтворення мультимедіа.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	ЗЕЛЕНЬОВА І.Я., к.т.н.		
нормоконтроль	ДЬЯЧУК Т.С., ст. викл.		

7. Дата видачі завдання 01.05.2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз ринку медіасерверів та порівняльний аналіз існуючих клієнт-серверних рішень (Plex Media Server, Jellyfin, Emby)	05.03.2026 р.	
2	Розробка специфікації вимог SRS до системи зберігання, пошуку та відтворення мультимедіа (завантаження, індексація, пошук, плейлисти, галереї, ролі користувачів)	20.03.2026 р.	
3	Проектування тришарової клієнт-серверної архітектури системи з використанням UML-діаграм (Use Case, компонентна, послідовності, розгортання)	30.03.2026 р.	
4	Розробка структури реляційної бази даних PostgreSQL 16 для зберігання метаданих музичних файлів та зображень, ER-діаграми і діаграми класів	05.04.2026 р.	
5	Розробка серверної частини на Node.js 20 / Express.js 4: реалізація REST API, модулів індексації ID3-тегів і EXIF, потокової передачі через HTTP Range-запити	20.04.2026 р.	
6	Розробка клієнтської частини на React 18 та Tailwind CSS: адаптивний інтерфейс, музичний плеєр, галерея зображень, повнотекстовий пошук	01.05.2026 р.	
7	Інтеграція компонентів системи, налаштування Nginx як зворотного проксі, реалізація механізмів безпеки (JWT-автентифікація, TLS-шифрування)	10.05.2026 р.	
8	Функціональне та навантажувальне тестування системи з використанням Apache JMeter, тестування модуля потокової передачі аудіо	15.05.2026 р.	
9	Оформлення пояснювальної записки та графічного матеріалу	20.05.2026 р.	

Студент (ка)

(підпис)

В'ячеслав ЛАХНО
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Ірина ЗЕЛЕНЬОВА
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

ПЗ: 79 с., 36 рис., 20 табл., 20 джерел, 2 додатка.

КЛІЄНТ-СЕРВЕРНА СИСТЕМА, МЕДІАСЕРВЕР, ЗБЕРІГАННЯ МУЛЬТИМЕДІА, ПОТОКОВЕ ВІДТВОРЕННЯ, ПОВНОТЕКСТОВИЙ ПОШУК, ІНДЕКСАЦІЯ МЕТАДАНИХ, REACT, NODE.JS, EXPRESS.JS, POSTGRESQL, NGINX, REST API, JWT, HTTP RANGE, ID3-ТЕГИ, EXIF

Метою дипломного проєкту є розробка клієнт-серверної системи, що забезпечує надійне зберігання, ефективний пошук та потокове відтворення музичних файлів і зображень користувача через зручний веб-інтерфейс із можливістю масштабування і захисту приватних даних.

Об'єктом дослідження є процеси зберігання, індексації, пошуку та потокового відтворення мультимедійного контенту в розподілених інформаційних системах. Предметом дослідження є методи та засоби реалізації клієнт-серверної архітектури для організації персональних медіасховищ музичних файлів та зображень.

У роботі проведено порівняльний аналіз провідних медіасерверів (Plex Media Server, Jellyfin, Emby), сформовано перелік функціональних і нефункціональних вимог до системи, спроектовано тришарову клієнт-серверну архітектуру з використанням нотації UML, розроблено структуру реляційної бази даних PostgreSQL з вісьмома таблицями, реалізовано серверну та клієнтську частини системи, а також виконано функціональне і навантажувальне тестування з використанням Apache JMeter.

ЗМІСТ

Вступ.....	6
1 Аналіз предметної області та сучасних рішень для зберігання і відтворення мультимедійного контенту.....	9
1.1 Огляд сучасного стану ринку медіасерверів та хмарних сховищ мультимедіа .	9
1.2 Порівняльний аналіз існуючих клієнт-серверних рішень для роботи з музикою та зображеннями.....	11
1.3 Формування функціональних і нефункціональних вимог до системи.....	17
1.4 Висновки за розділом 1.....	20
2 Проєктування клієнт-серверної системи зберігання, пошуку та відтворення мультимедіа.....	21
2.1 Розробка специфікації вимог SRS до системи	21
2.2 Архітектура клієнт-серверної системи	25
2.3 Структура бази даних для метаданих музичних файлів та зображень.....	30
2.4 Висновки за розділом 2.....	37
3 Розробка клієнт-серверної системи зберігання, пошуку та відтворення мультимедіа.....	39
3.1 Технологічний стек системи	39
3.2 Апаратна частина клієнт-серверної системи.....	42
3.3 Програмна частина системи: серверні модулі та клієнтський інтерфейс	44
3.4 Висновки за розділом 3.....	51
4 Тестування клієнт-серверної системи зберігання, пошуку та відтворення мультимедіа.....	52
4.1 Розгортання системи на локальному сервері та обрання середовища тестування	52
4.2 Функціональне та навантажувальне тестування системи.....	55
4.3 Висновки за розділом 4.....	62
Висновки	64

Перелік джерел посилання	67
Додаток А Слайди презентації.....	69
Додаток Б Лістинги	75

ВСТУП

Стрімкий розвиток цифрових технологій, поширення мобільних пристроїв з вбудованими камерами високої роздільної здатності, а також зростання популярності потокових аудіосервісів призвели до експоненційного збільшення обсягів персональних мультимедійних даних, які щодня створює та зберігає кожен користувач. За оцінками аналітичних компаній, обсяг світових цифрових даних подвоюється приблизно кожні два роки, причому значну частку цього обсягу складають саме музичні файли та зображення, що належать приватним особам [1].

Існуючі публічні хмарні сервіси, такі як Google Drive, iCloud, Spotify чи YouTube Music, надають користувачам зручний інтерфейс для зберігання та відтворення мультимедіа, проте мають низку суттєвих недоліків: залежність від інфраструктури третьої сторони, обмеження на обсяг безкоштовного простору, ризики компрометації приватних даних, неможливість збереження аудіофайлів у форматах без втрат якості, а також постійна абонентська плата за розширений функціонал [2]. Ці фактори зумовлюють зростаючий попит на самостійно розгорнуті (self-hosted) клієнт-серверні системи, які надають користувачеві повний контроль над власним контентом.

Клієнт-серверна архітектура є оптимальним підходом для побудови персональних медіасховищ, оскільки вона дозволяє чітко розділити відповідальність між клієнтською частиною, що відповідає за відображення інтерфейсу та взаємодію з користувачем, і серверною частиною, що забезпечує зберігання файлів, обробку метаданих, індексацію та потокову передачу контенту. Такий підхід забезпечує доступ до медіатеки з будь-якого пристрою у локальній мережі або через Інтернет, а також дозволяє ефективно масштабувати систему відповідно до зростання обсягу даних [3].

Особливістю задачі є необхідність одночасної підтримки двох суттєво різних типів медіаконтенту – аудіофайлів і зображень, кожен з яких має власні особливості зберігання, індексації, пошуку та відтворення. Музичні файли

потребують вилучення метаданих з ID3-тегів, потокової передачі з підтримкою перемотки та формування плейлистів, тоді як зображення вимагають генерації мініатюр, EXIF-індексації, групування за альбомами і підтримки галерейного перегляду. Розробка системи, яка ефективно об'єднує обидва ці сценарії в межах єдиної клієнт-серверної архітектури, є актуальною та практично значущою задачею [4].

Метою даного дипломного проєкту є розробка клієнт-серверної системи, що забезпечує надійне зберігання, ефективний пошук та потокове відтворення музичних файлів і зображень користувача через зручний веб-інтерфейс із можливістю масштабування і захисту приватних даних.

Об'єктом дослідження є процеси зберігання, індексації, пошуку та потокового відтворення мультимедійного контенту в розподілених інформаційних системах. Предметом дослідження є методи та засоби реалізації клієнт-серверної архітектури для організації персональних медіасховищ музичних файлів та зображень.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- проаналізувати сучасний стан ринку медіасерверів і провести порівняльне дослідження існуючих клієнт-серверних рішень для роботи з музикою та зображеннями;
- сформулювати функціональні і нефункціональні вимоги до системи, розробити специфікацію вимог SRS;
- спроектувати клієнт-серверну архітектуру системи, що забезпечує необхідний рівень продуктивності, безпеки та масштабованості;
- розробити структуру реляційної бази даних для зберігання метаданих музичних файлів і зображень, спроектувати ER- та діаграму класів предметної області;
- обрати технологічний стек і реалізувати серверну та клієнтську частини системи з підтримкою завантаження, індексації, повнотекстового пошуку, потокового відтворення аудіо та галерейного перегляду зображень;

- провести функціональне і навантажувальне тестування системи та підготувати рекомендації щодо її розгортання.

Практична цінність роботи полягає у створенні готового до використання програмного продукту, який надає користувачам повний контроль над власною медіатекою, забезпечує конфіденційність зберігання приватного контенту, не залежить від комерційних хмарних провайдерів і може бути розгорнутий як на персональному обладнанні, так і на орендованому VPS-сервері. Розроблена система має потенціал для подальшого розвитку шляхом додавання модулів автоматичного резервного копіювання, машинного розпізнавання облич на фотографіях, рекомендаційної системи для музики та підтримки додаткових типів медіаконтенту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА СУЧАСНИХ РІШЕНЬ ДЛЯ ЗБЕРІГАННЯ І ВІДТВОРЕННЯ МУЛЬТИМЕДІЙНОГО КОНТЕНТУ

1.1 Огляд сучасного стану ринку медіасерверів та хмарних сховищ мультимедіа

Сучасний ринок програмного забезпечення для зберігання та відтворення мультимедійного контенту переживає період активного розвитку, зумовленого вибуховим зростанням обсягів цифрових даних, які створюють і споживають користувачі. За даними дослідницької компанії IDC, загальний обсяг цифрової інформації у світі досяг понад 180 зетабайт у 2025 році, причому на персональні мультимедійні дані – фотографії, відеозаписи та аудіофайли – припадає близько 30 % цього обсягу [1]. Зростання популярності смартфонів із камерами високої роздільної здатності, поширення безлічевих аудіоформатів (FLAC, ALAC, DSD), а також підвищення вимог користувачів до якості та доступності контенту формують стійкий попит на спеціалізовані рішення для організації персональних медіатек.

Динаміка зростання обсягів персонального мультимедійного контенту демонструє стійку тенденцію до збільшення приблизно на 25–30 % щорічно (рис. 1.1). Особливо помітне зростання відбувається у сегменті аудіоконтенту високої якості: за останні п'ять років обсяг купленої музики у форматах без втрат збільшився більш ніж утричі, що пов'язано з популяризацією Hi-Res Audio та активним розвитком ринку якісних аудіопристроїв [2].

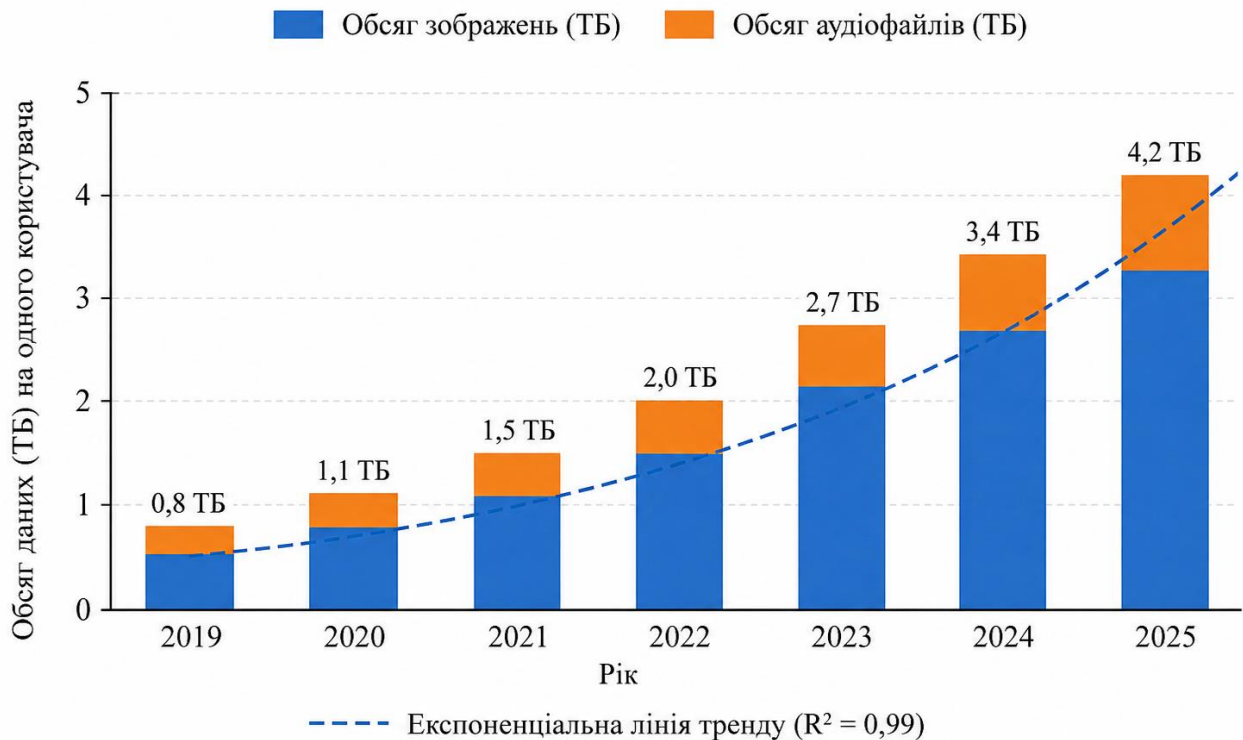


Рисунок 1.1 – Динаміка зростання обсягу персональних мультимедійних даних за період 2019–2025 рр.

На ринку рішень для зберігання мультимедіа сформувалися два основні підходи. Перший – публічні хмарні сервіси (Google Photos, iCloud Photos, Spotify, Apple Music, YouTube Music), що пропонують зручний інтерфейс і автоматичну синхронізацію між пристроями, але вимагають передачі приватного контенту третій стороні та абонентської плати за розширення обсягу сховища [3]. Другий підхід – самостійно розгорнуті медіасервери (self-hosted media servers), які користувач встановлює на власне обладнання, отримуючи повний контроль над даними, незалежність від комерційних провайдерів та необмежений обсяг сховища, обмежений лише фізичною ємністю дисків [4].

Український ринок персональних медіасховищ перебуває у фазі активного формування. Зростаючий рівень технічної грамотності користувачів, доступність потужних мініатюрних серверних платформ (Intel NUC, Raspberry Pi, мікросерверні рішення) та поширення гігабітного домашнього інтернету сприяють масовому переходу до самостійно розгорнутих рішень. Додатковим фактором, що стимулює

інтерес до приватних медіасерверів, є посилення вимог до конфіденційності персональних даних відповідно до європейського Загального регламенту захисту даних (GDPR) та аналогічних національних нормативних актів [5].

Серед ключових тенденцій ринку слід виділити перехід до повноцінної підтримки потокової передачі (streaming) замість завантаження файлів цілком, що дозволяє економити трафік і починати відтворення майже миттєво. Іншою важливою тенденцією є інтеграція машинного навчання для автоматичної класифікації фотографій, розпізнавання облич, а також генерації музичних рекомендацій на основі історії прослуховувань. Активно розвивається підтримка адаптивного бітрейту (HLS, DASH), що забезпечує плавне відтворення медіа на пристроях з різною пропускною здатністю каналу [6].

1.2 Порівняльний аналіз існуючих клієнт-серверних рішень для роботи з музикою та зображеннями

Для обґрунтованого вибору архітектурних та технологічних рішень при розробці власної клієнт-серверної системи необхідно провести аналіз трьох найпопулярніших сучасних аналогів, що реалізують функціонал зберігання, пошуку та відтворення мультимедійного контенту. До розгляду обрано Plex Media Server, Jellyfin та Emby як найбільш зрілі представники сегменту самостійно розгорнутих медіасерверів, що підтримують одночасно музичні файли та зображення.

Plex Media Server є комерційним рішенням компанії Plex Inc., що поєднує локальне зберігання медіаконтенту з хмарною інфраструктурою для авторизації, синхронізації метаданих та віддаленого доступу через сервери компанії (рис. 1.2). Plex використовує власний транскодинг-движок на основі FFmpeg, що дозволяє динамічно конвертувати медіафайли під можливості клієнтського пристрою. Серверна частина написана переважно на C++ та Python, клієнтські застосунки

доступні для всіх популярних платформ, включаючи Windows, macOS, Linux, Android, iOS, Smart TV. Безкоштовна версія має суттєві обмеження – більшість розширених функцій (мобільний доступ, синхронізація, апаратне прискорення транскодингу) доступні лише за щомісячною підпискою Plex Pass [7].



Рисунок 1.2 – Інтерфейс Plex Media Server

Jellyfin – повністю безкоштовне рішення з відкритим кодом (open-source), що з'явилося як форк проєкту Emby у 2018 році після переходу останнього на пропрієтарну ліцензію (рис. 1.3). Jellyfin не використовує жодних хмарних сервісів і працює виключно локально, що робить його привабливим для користувачів, які приділяють особливу увагу конфіденційності. Серверна частина написана на C# (.NET 8), що забезпечує крос-платформну сумісність та високу продуктивність. Клієнтські застосунки розробляються спільнотою і покривають широкий спектр платформ, включно з Roku, Kodi, LG WebOS, Samsung Tizen. Jellyfin підтримує апаратне прискорення транскодингу через VA-API, NVENC, QuickSync, а також повноцінний live TV та DVR-функціонал [8].

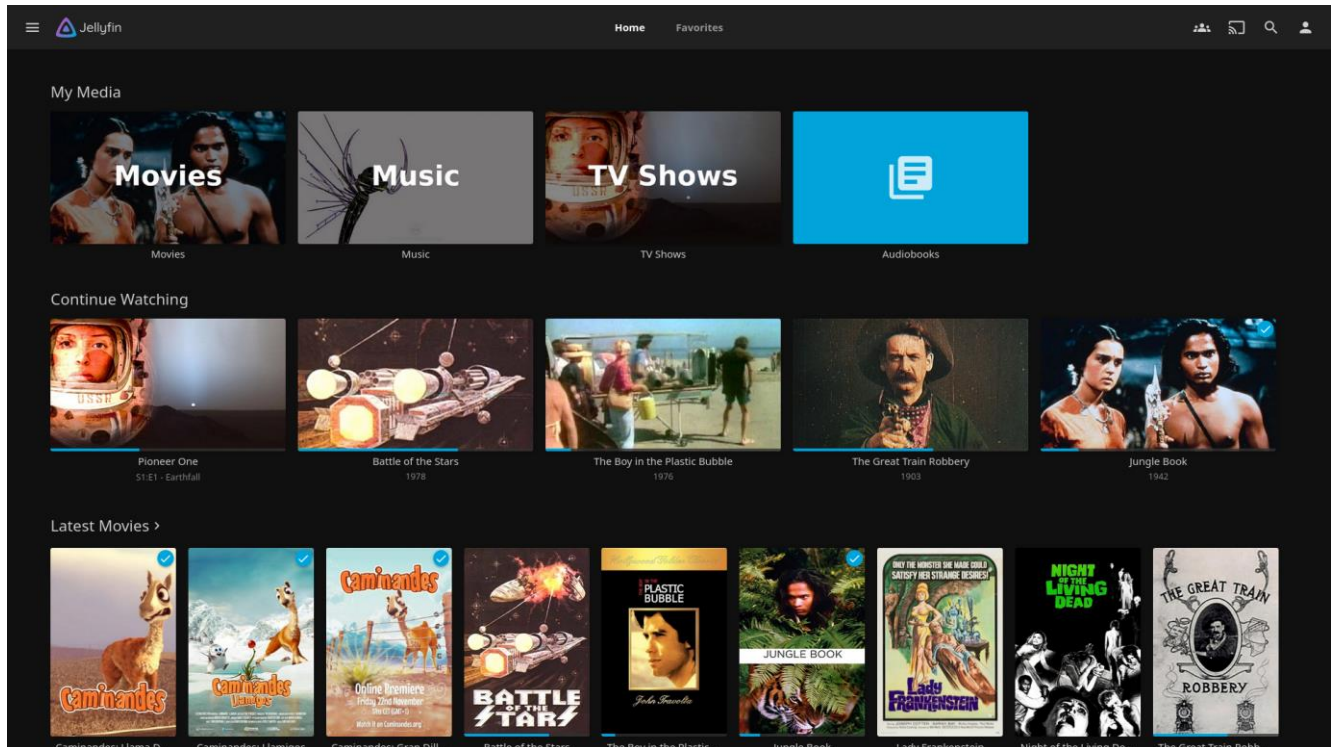


Рисунок 1.3 – Інтерфейс Jellyfin Media Server

Emby – гібридне рішення, що пропонує безкоштовну базову версію з обмеженим функціоналом та платну Premiere-підписку для розблокування повного набору можливостей (рис. 1.4). Серверна частина написана на C# і має модульну архітектуру з підтримкою плагінів від спільноти. Emby надає розширені можливості налаштування метаданих, потужну систему батьківського контролю, а також підтримку DLNA для пристроїв без власного клієнта. Однак переведення Emby на пропрієтарну модель розповсюдження у 2018 році суттєво зменшило підтримку з боку відкритої спільноти на користь Jellyfin [9].

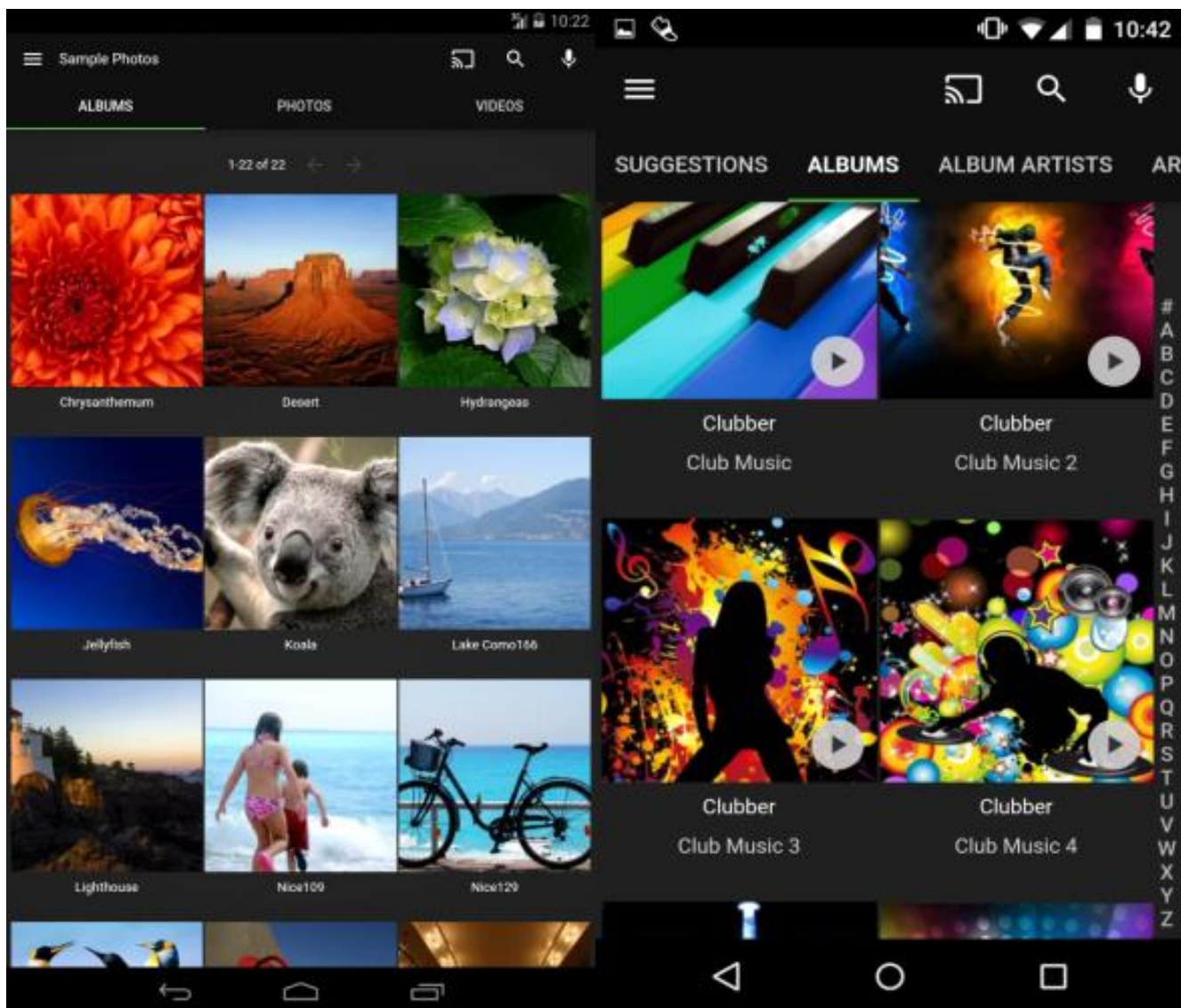


Рисунок 1.4 – Інтерфейс Emby Media Server

Функціональне порівняння трьох систем-аналогів за ключовими критеріями представлено у таблиці 1.1, технічне порівняння – у таблиці 1.2.

Аналіз даних таблиці 1.1 показує, що всі три системи забезпечують базовий функціонал зберігання та відтворення музики і зображень, однак суттєво відрізняються за моделлю розповсюдження, ступенем залежності від зовнішніх сервісів та доступністю розширених можливостей. Plex демонструє найбільш зрілий продукт із продуманим UX, але вимагає постійної взаємодії з хмарними серверами компанії та платної підписки для повного функціоналу. Jellyfin є найбільш збалансованим рішенням з точки зору приватності та вартості, проте поступається конкурентам у функціях машинного навчання. Emby займає проміжну

позицію між Plex та Jellyfin за рівнем функціональності та гнучкістю налаштування.

Таблиця 1.1 – Функціональне порівняння аналогів

Критерій	Plex	Jellyfin	Emby
Підтримка музики	так	так	так
Підтримка зображень	так	так	так
Повнотекстовий пошук	розширений	базовий	розширений
Транскодинг на льоту	так (платно)	так	так (платно)
Адаптивний стримінг (HLS)	так	так	так
Розпізнавання облич	ні	через плагін	так (платно)
Мобільні застосунки	платно	безкоштовно	платно
Ліцензія	Proprietary	GPL v2 (open-source)	Proprietary
Залежність від хмари	обов'язкова	відсутня	часткова
Вартість повного функціоналу	≈5 USD/міс	безкоштовно	≈4,99 USD/міс

Таблиця 1.2 – Технічне порівняння аналогів

Параметр	Plex	Jellyfin	Emby
Мова серверу	C++ / Python	C# (.NET 8)	C# (.NET 8)
База даних	SQLite (вбудована)	SQLite / PostgreSQL	SQLite (вбудована)
Обробка медіа	FFmpeg (модифікований)	FFmpeg	FFmpeg
Мінімальна RAM	2 ГБ	1 ГБ	2 ГБ
Підтримувані ОС сервера	Windows, Linux, macOS, FreeBSD, NAS	Windows, Linux, macOS, Docker	Windows, Linux, macOS, Docker
API для розробників	закритий REST API	відкритий REST + WebSocket	REST API
Протоколи стримінгу	HLS, DASH, прямий	HLS, DASH, прямий	HLS, DASH, прямий
Підтримка плагінів	обмежена	повна	повна

Як видно з таблиці 1.2, Jellyfin та Emby технологічно дуже близькі між собою через спільне історичне коріння – обидва написані на C# для .NET 8 та використовують SQLite як основну базу даних. Plex має дещо специфічнішу архітектуру з модифікованою версією FFmpeg та власним протокольним стеком. Особливістю Jellyfin є наявність повністю відкритого REST API з підтримкою WebSocket, що робить його привабливою основою для побудови сторонніх інтеграцій.

Спільними недоліками всіх трьох розглянутих рішень є надмірна функціональна складність, орієнтована переважно на відеоконтент, що призводить до великого розміру встановлюваного дистрибутиву (від 200 МБ до 1 ГБ), високих

вимог до апаратних ресурсів та довгого часу первинного індексування бібліотеки. Для користувачів, яким потрібно лише зберігати, шукати та відтворювати музику і зображення, ці рішення є надлишковими. Крім того, налаштування Plex та Emby потребує реєстрації у зовнішніх сервісах, а інтерфейс Jellyfin успадкував від Emby складну систему налаштувань, що ускладнює його використання непідготовленим користувачем.

Таким чином, незважаючи на наявність зрілих рішень на ринку, існує практична потреба у створенні легковагової клієнт-серверної системи, спеціалізованої саме на роботі з музичними файлами та зображеннями, що поєднує переваги розглянутих аналогів (відкритість Jellyfin, простоту інтерфейсу Plex, гнучкість Emby) і не успадковує їхніх недоліків.

1.3 Формування функціональних і нефункціональних вимог до системи

На основі проведеного аналізу ринку та порівняльного дослідження аналогів сформовано перелік ключових функціональних та нефункціональних вимог, які мають бути реалізовані у розроблюваній клієнт-серверній системі. До функціональних вимог належить можливість завантаження медіафайлів на сервер, автоматичної індексації метаданих з ID3-тегів аудіофайлів та EXIF-даних зображень, повнотекстового пошуку за назвою, виконавцем, альбомом та тегами, потокового відтворення аудіо з підтримкою перемотки, формування плейлистів і альбомів, перегляду галереї зображень з генерацією мініатюр, а також управління правами користувачів через систему ролей.

До нефункціональних вимог належать висока продуктивність із часом відгуку API не більше 500 мс при типових операціях, масштабованість до сотень тисяч медіафайлів у бібліотеці одного користувача, безпека з обов'язковим шифруванням з'єднання за TLS та автентифікацією на основі JWT-токенів, підтримка горизонтального масштабування за рахунок винесення файлового

сховища на окремий сервер, а також крос-браузерна сумісність клієнтської частини з усіма популярними браузерами актуальних версій.

Деталізований огляд ключових завдань розробки системи представлено в таблиці 1.3. Таблиця охоплює весь життєвий цикл проєкту – від початкового аналізу потреб користувачів до етапу підтримки готового продукту.

Таблиця 1.3 – Огляд ключових завдань розробки клієнт-серверної системи

Етап	Опис завдань
Аналіз потреб користувачів	Дослідження вимог цільової аудиторії до функціоналу системи зберігання та відтворення персональних медіафайлів, виявлення больових точок існуючих рішень.
Аналіз конкурентних рішень	Порівняльне дослідження Plex, Jellyfin та Emby для визначення сильних і слабких сторін, напрацювання списку функціональних запозичень.
Формулювання SRS	Документування функціональних та нефункціональних вимог у вигляді специфікації Software Requirements Specification.
Вибір технологічного стеку	Обґрунтування вибору мов програмування, фреймворків, СУБД та допоміжних інструментів з урахуванням вимог до продуктивності та масштабованості.
Проектування архітектури	Розробка архітектурної схеми системи, визначення компонентів, протоколів взаємодії та точок розширення.

Продовження таблиці 1.3

Етап	Опис завдань
Проектування БД	Розробка схеми реляційної бази даних, ER-діаграми та діаграми класів предметної області для метаданих музики та зображень.
Розробка серверної частини	Реалізація REST API, модулів індексації медіа, повнотекстового пошуку та потокової передачі контенту.
Розробка клієнтської частини	Реалізація вебінтерфейсу з адаптивним дизайном, музичного плеєра та галереї зображень.
Тестування	Функціональне, інтеграційне та навантажувальне тестування системи для перевірки відповідності SRS та оцінки продуктивності.
Розгортання	Підготовка інструкцій з розгортання системи на цільовому сервері, налаштування зворотного проксі, TLS та механізмів моніторингу.
Підтримка та розвиток	Виправлення помилок, додавання нового функціоналу за результатами зворотного зв'язку від користувачів, оновлення залежностей.

Дані таблиці 1.3 ілюструють поетапний підхід до розробки клієнт-серверної системи, у якому кожен наступний етап спирається на результати попереднього. Особлива увага приділяється етапам проектування архітектури та бази даних, оскільки прийняті на цих стадіях рішення суттєво впливають на можливості подальшого масштабування і супроводу системи.

Цільовою аудиторією системи є приватні користувачі, які мають значні особисті колекції музики та фотографій (від десятків гігабайт до кількох терабайт), цінують конфіденційність власних даних та володіють базовими навичками

встановлення серверного програмного забезпечення. Додатковими сегментами користувачів є невеликі творчі колективи (фотостудії, домашні аудіозаклади), яким потрібен централізований сервер для спільного доступу до робочих матеріалів, а також технічно підкуті користувачі, що шукають альтернативу комерційним хмарним сервісам через мотиви захисту приватності або економії коштів.

1.4 Висновки за розділом 1

У рамках першого розділу проведено комплексний аналіз предметної області та сучасних рішень для зберігання, пошуку та відтворення мультимедійного контенту. Встановлено, що ринок персональних медіасховищ перебуває у фазі стабільного зростання, зумовленого збільшенням обсягів цифрових даних користувачів та підвищенням вимог до конфіденційності.

Виконано детальне порівняльне дослідження трьох провідних аналогів – Plex Media Server, Jellyfin та Emby. Аналіз показав, що Plex є найбільш зрілим комерційним продуктом, але вимагає взаємодії з хмарною інфраструктурою та платної підписки. Jellyfin є оптимальним вибором з точки зору відкритості та приватності, але має обмежений набір функцій машинного навчання. Emby займає проміжну позицію між двома першими рішеннями. Спільними недоліками всіх розглянутих систем є надмірна функціональна складність та орієнтація переважно на відеоконтент.

За результатами аналізу обґрунтовано необхідність розробки спеціалізованої клієнт-серверної системи, орієнтованої виключно на роботу з музичними файлами та зображеннями. Сформовано перелік функціональних і нефункціональних вимог до системи, що включає завантаження файлів, автоматичну індексацію метаданих, повнотекстовий пошук, потокове відтворення, формування плейлистів та галерей, систему ролей користувачів, високу

продуктивність, безпеку на основі TLS і JWT, а також можливість горизонтального масштабування.

2 ПРОЄКТУВАННЯ КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ ЗБЕРІГАННЯ, ПОШУКУ ТА ВІДТВОРЕННЯ МУЛЬТИМЕДІА

2.1 Розробка специфікації вимог SRS до системи

Розробка специфікації вимог до програмного забезпечення (Software Requirements Specification, SRS) є фундаментальним етапом проєктування клієнт-серверної системи зберігання, пошуку та відтворення мультимедіа. SRS-документ формалізує функціональні та нефункціональні вимоги, описує сценарії взаємодії користувачів із системою, а також встановлює критерії приймання, за якими буде оцінюватися готова система. На основі сформованих у попередньому розділі вимог проведено моделювання предметної області з використанням мови графічного моделювання UML (Unified Modeling Language).

Першим кроком при проєктуванні системи є визначення акторів – суб'єктів взаємодії з системою, що мають різні набори дозволених операцій. У розроблюваній системі виділено три основні ролі акторів. Узагальнена характеристика ролей акторів представлена у таблиці 2.1.

Для візуалізації функціональних можливостей системи з точки зору акторів побудовано діаграму варіантів використання (Use Case Diagram), що наведена на рис. 2.1. Ця діаграма дає змогу однозначно визначити межі системи, перелік підтримуваних сценаріїв та зв'язки між ними.

Таблиця 2.1 – Ролі акторів системи

Актор	Опис	Типові операції
Гість	Неавторизований користувач, що відвідав вебсайт системи	перегляд інформаційної сторінки, реєстрація, авторизація
Користувач	Авторизована особа, що володіє власною медiateкою	завантаження файлів, пошук, відтворення, формування плейлистів, перегляд галереї
Адміністратор	Особа, що має повний доступ до управління системою	управління користувачами, моніторинг сервера, налаштування системи

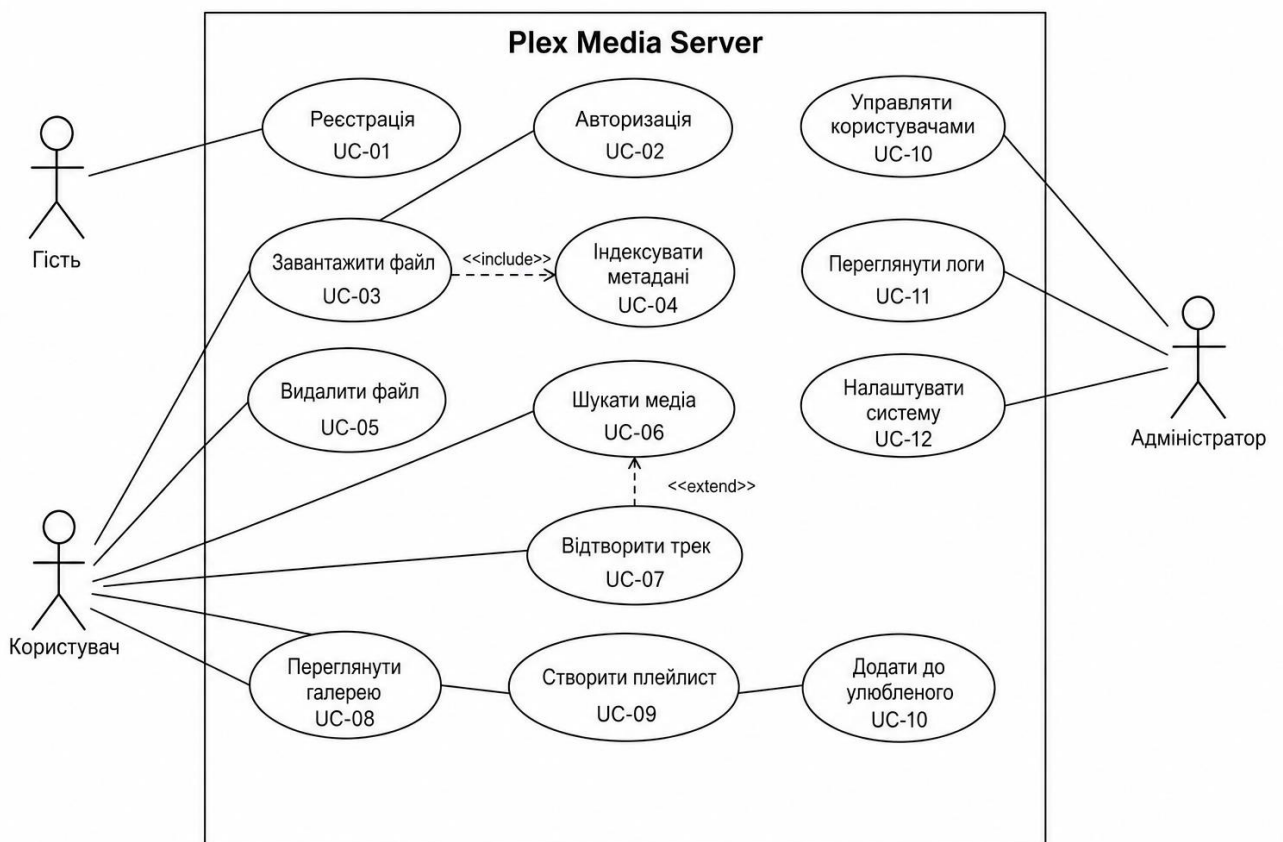


Рисунок 2.1 – Діаграма варіантів використання системи

Особливістю розробленої діаграми варіантів використання є чітке розмежування ролей: гість має доступ лише до варіантів реєстрації та авторизації, користувач отримує доступ до основного функціоналу роботи з медіатекою, а адміністратор має повний контроль над системою, включно з управлінням обліковими записами користувачів. Стереотипи «<<include>>» позначають обов'язкове виконання включеного варіанту використання (наприклад, при завантаженні файлу обов'язково виконується індексація метаданих), стереотипи «<<extend>>» – опціональне розширення (наприклад, відтворення треку може бути викликане з варіанту пошуку).

Для деталізації найбільш складного процесу системи – завантаження та індексації медіафайлу – побудовано діаграму діяльності (Activity Diagram), представлену на рис. 2.2. Діаграма відображає послідовність дій, точки прийняття рішень, паралельні гілки виконання та можливі альтернативні сценарії.

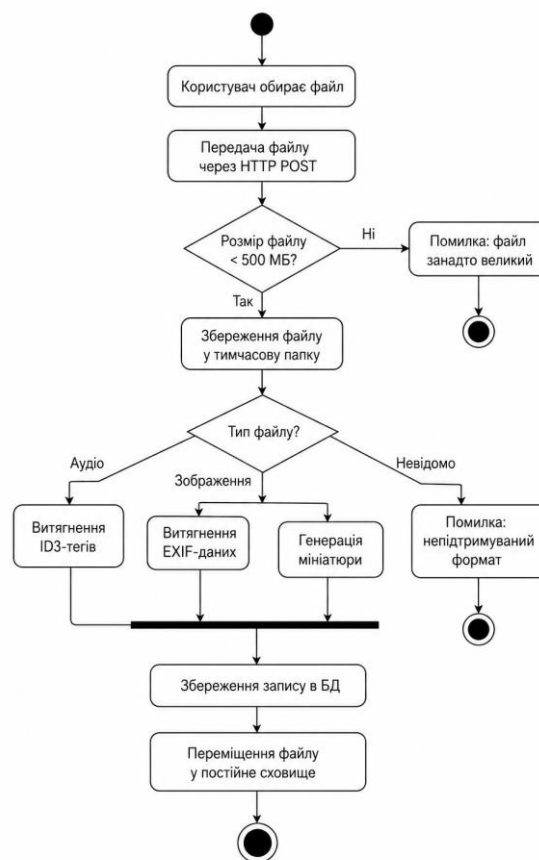


Рисунок 2.2 – Діаграма діяльності для процесу завантаження та індексації медіафайлу

Для опису поведінки інтерактивного компонента системи – музичного плеєра – використано діаграму станів (State Machine Diagram), що наведена на рис. 2.3. Плеєр у будь-який момент часу перебуває в одному з визначених станів, переходи між якими відбуваються у відповідь на події – натискання кнопок користувача або системні події (закінчення треку, мережева помилка тощо).

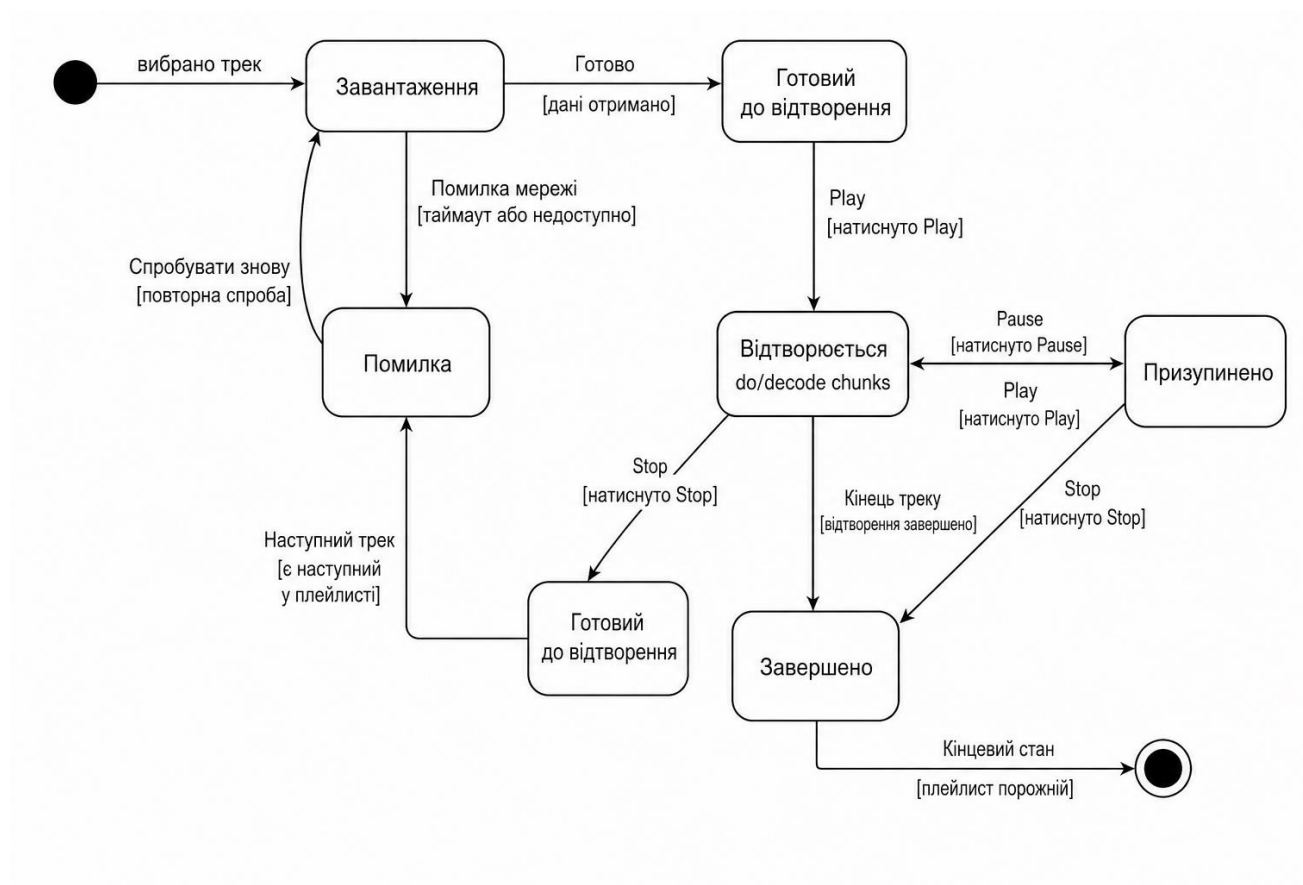


Рисунок 2.3 – Діаграма станів музичного плеєра

Розроблена діаграма станів забезпечує однозначне розуміння поведінки плеєра у всіх можливих сценаріях, що особливо важливо при реалізації обробки помилок мережі та забезпеченні безперервного відтворення черги треків. Подібний підхід до проєктування дозволяє уникнути типових помилок, пов'язаних з неузгодженою поведінкою інтерфейсу при паралельних діях користувача.

2.2 Архітектура клієнт-серверної системи

Архітектура комп'ютерної системи має визначальне значення для забезпечення продуктивності, масштабованості та надійності розроблюваного програмного забезпечення. При проектуванні клієнт-серверних систем зберігання та відтворення мультимедіа потенційно можуть бути застосовані кілька архітектурних підходів – моноліт, мікросервіси, тришарова клієнт-серверна модель, безсерверна (serverless) архітектура. Порівняння цих підходів за ключовими параметрами наведено у таблиці 2.2.

Таблиця 2.2 – Порівняння архітектурних підходів

Параметр	Моноліт	Тришарова клієнт-серверна	Мікросервіси	Serverless
Складність розробки	низька	середня	висока	середня
Масштабованість	обмежена	горизонтальна	гранулярна	автоматична
Розгортання	просте	просте	складне	делеговане
Підтримка стрімінгу	так	так	так	обмежено
Витрати на хостинг	низькі	низькі	високі	за використанням
Підходить для self-hosted	так	так	ні	ні

На основі даних таблиці 2.2 для розроблюваної системи обрано тришарову клієнт-серверну архітектуру, що забезпечує оптимальне співвідношення між

простотою розробки, можливостями масштабування та підходящістю для самостійного розгортання користувачем. Тришарова модель передбачає чітке розділення на рівень представлення (Presentation Layer), рівень бізнес-логіки (Business Logic Layer) та рівень доступу до даних (Data Access Layer), що забезпечує модульність та можливість незалежного оновлення кожного шару.

Загальна архітектура розроблюваної клієнт-серверної системи представлена на рис. 2.4. Архітектура передбачає розділення системи на чотири основні компоненти: клієнтська частина (вебзастосунок у браузері), серверна частина (REST API сервер з модулями індексації та стримінгу), реляційна база даних метаданих та файлове сховище для зберігання власне медіафайлів.

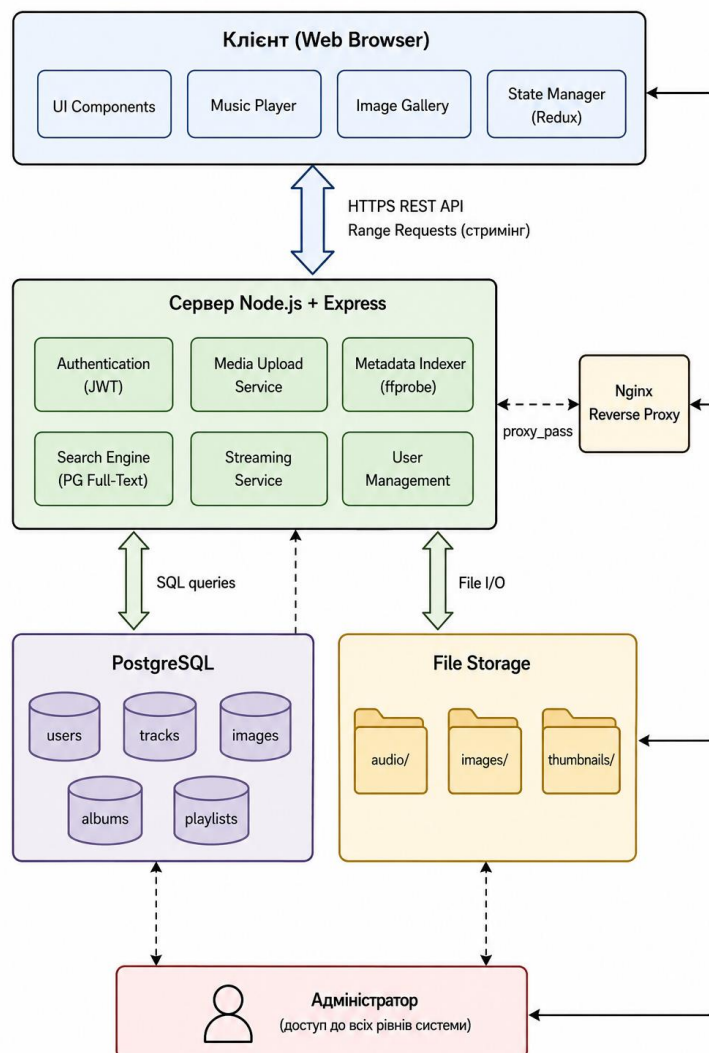


Рисунок 2.4 – Загальна архітектура клієнт-серверної системи

Для більш деталізованого опису внутрішньої структури системи розроблено компонентну діаграму (Component Diagram), наведену на рис. 2.5. Діаграма показує всі програмні компоненти системи, надані ними інтерфейси та залежності між компонентами.

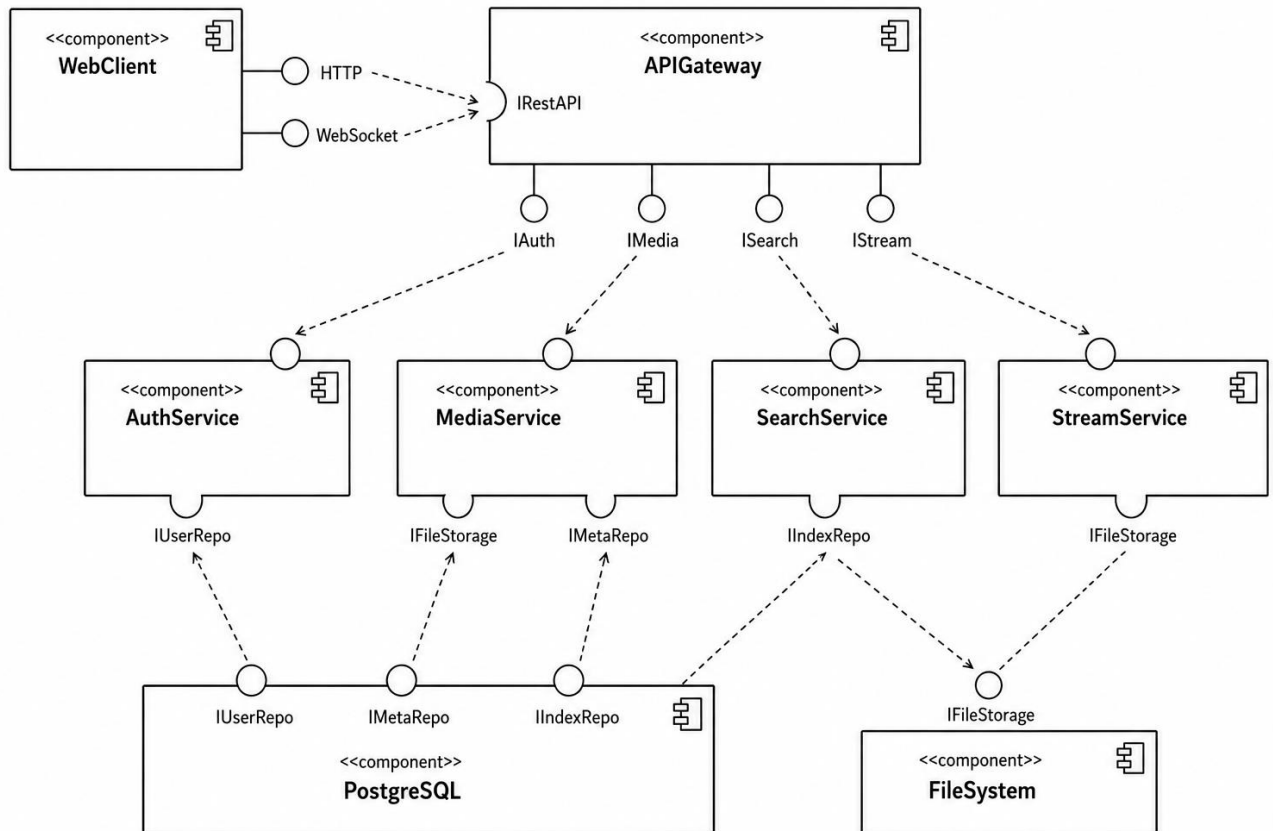


Рисунок 2.5 – Компонентна діаграма системи

Для деталізованого опису ключових сценаріїв взаємодії компонентів системи побудовано діаграми послідовності (Sequence Diagrams). На рис. 2.6 представлено діаграму послідовності для сценарію пошуку та потокового відтворення треку – одного з найважливіших сценаріїв роботи системи.

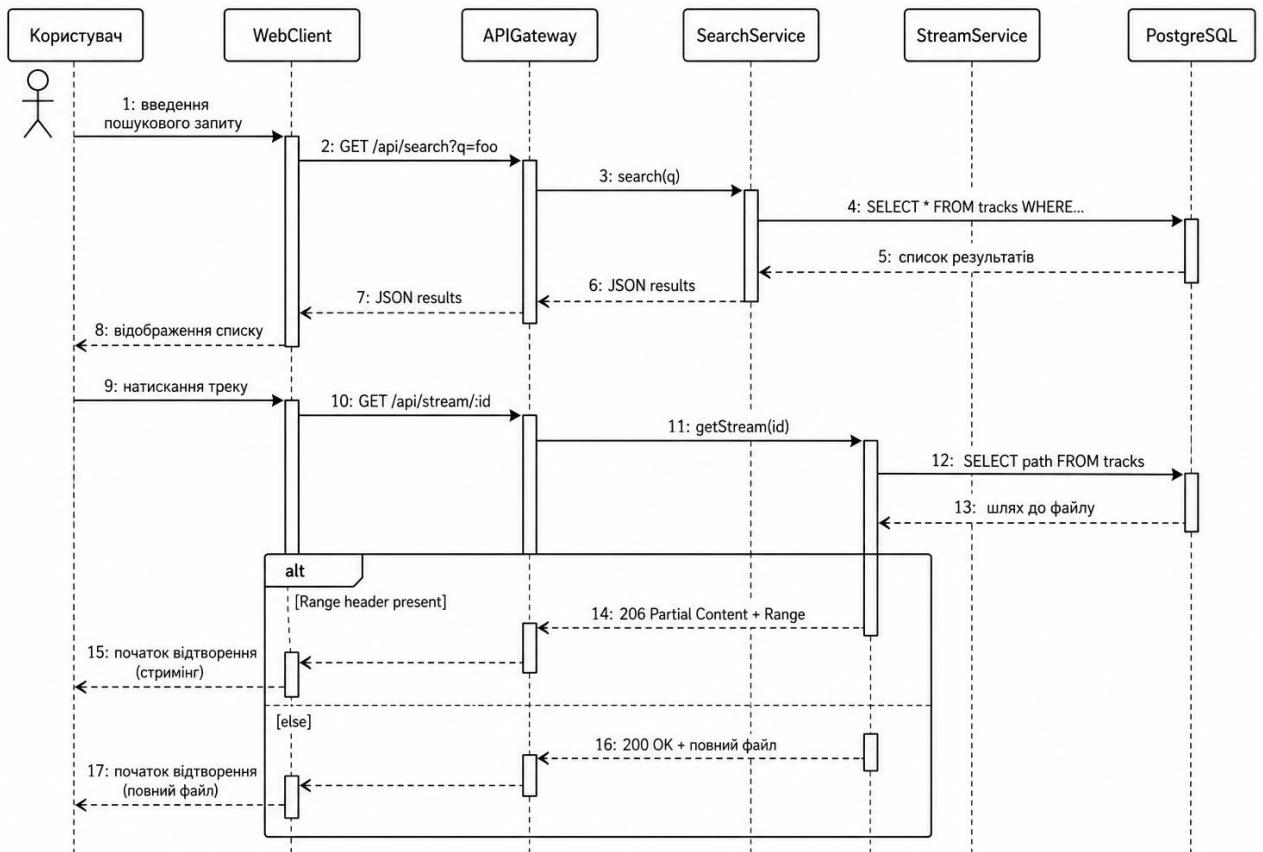


Рисунок 2.6 – Діаграма послідовності для пошуку та потокового відтворення треку

Другим важливим сценарієм є завантаження нового медіафайлу до системи з подальшою індексацією. Діаграму послідовності для цього сценарію наведено на рис. 2.7. Сценарій включає кілька асинхронних операцій – власне передача файлу, паралельне витягнення метаданих, генерація мініатюри (для зображень) та запис у базу даних.

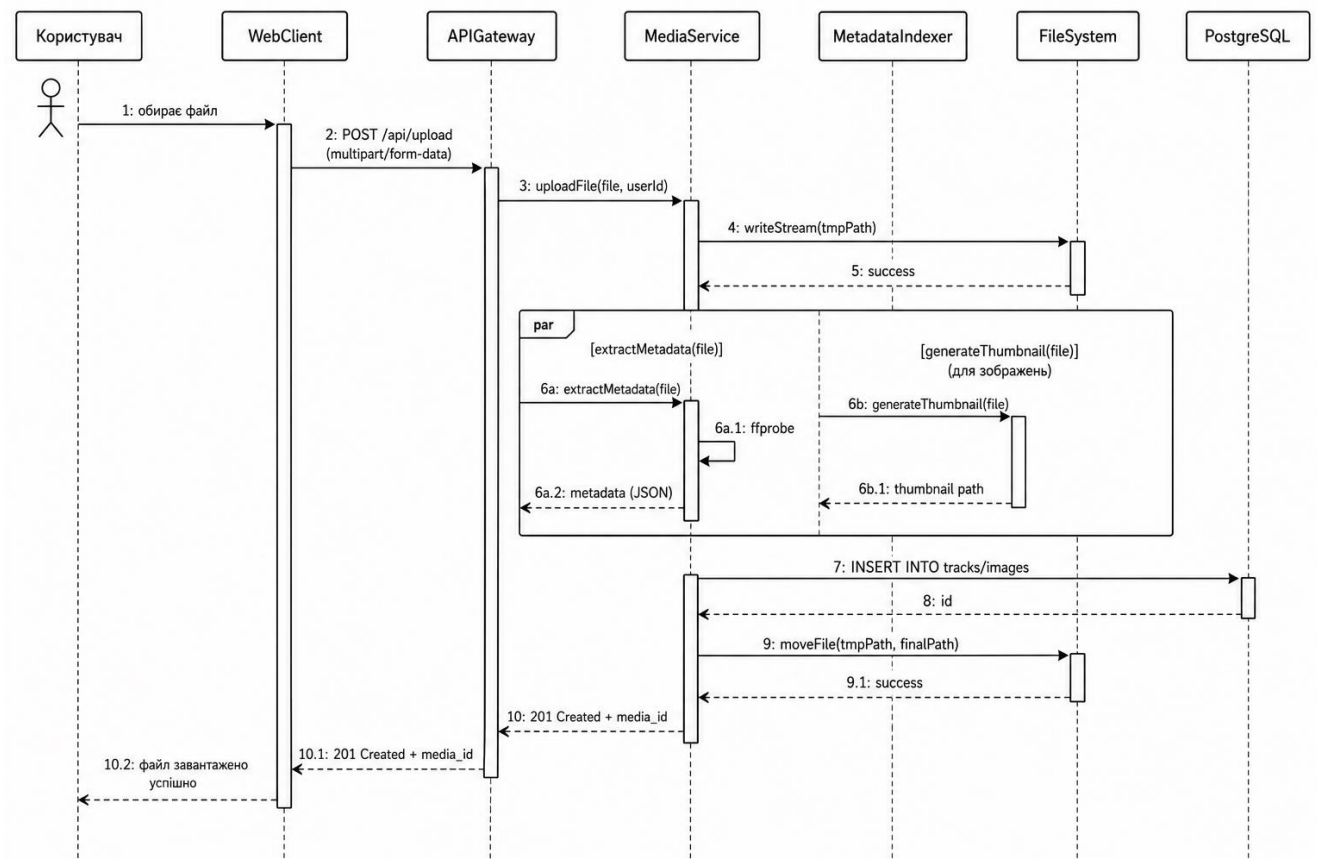


Рисунок 2.7 – Діаграма послідовності для завантаження та індексації медіафайлу

Для опису фізичної організації розгортання компонентів системи на цільовому обладнанні розроблено діаграму розгортання (Deployment Diagram), що наведена на рис. 2.8. Діаграма відображає вузли (фізичні чи віртуальні машини) та артефакти (виконувані файли, компоненти), які виконуються на цих вузлах.

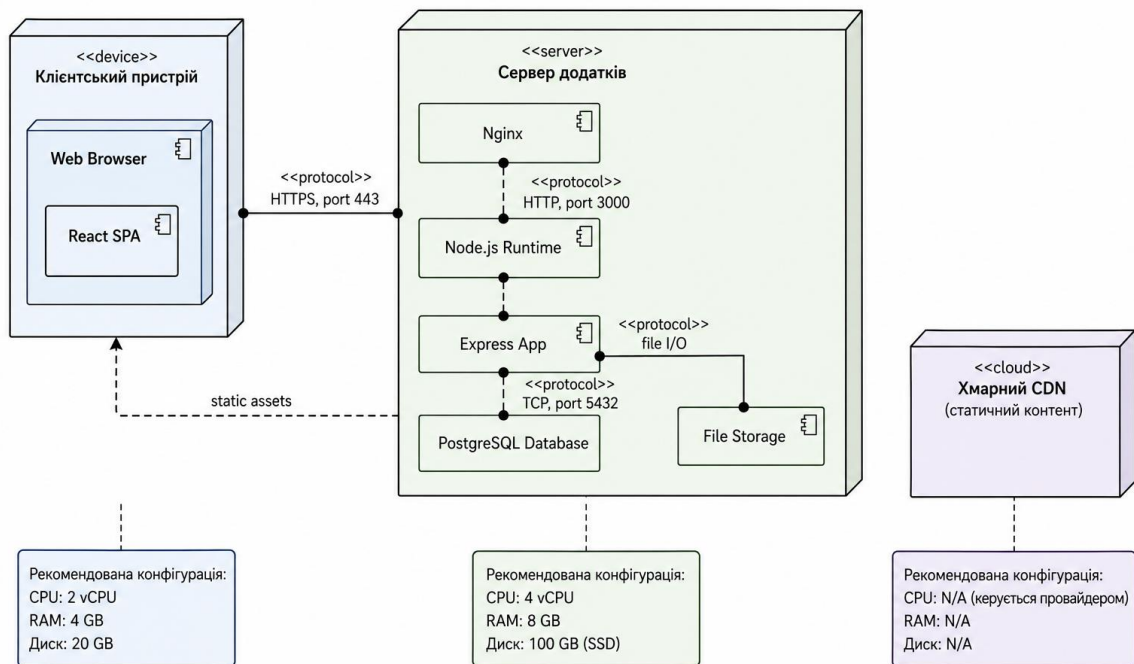


Рисунок 2.8 – Діаграма розгортання системи

Обрана архітектура забезпечує необхідний рівень модульності та розширюваності системи. Кожен сервіс (AuthService, MediaService, SearchService, StreamService) має чітко окреслену зону відповідальності та комунікує з іншими через визначені інтерфейси, що дозволяє у майбутньому замінювати або модифікувати окремі сервіси без зміни решти системи. Використання реляційної бази даних PostgreSQL забезпечує надійне зберігання структурованих метаданих з підтримкою повнотекстового пошуку, а виокремлення файлового сховища у самостійний компонент дозволяє за необхідності винести його на окремий сервер або мережеве сховище NAS.

2.3 Структура бази даних для метаданих музичних файлів та зображень

База даних розроблюваної системи реалізована у вигляді реляційної СУБД PostgreSQL та включає вісім основних таблиць, що зберігають метадані

користувачів, медіафайлів, плейлистів, альбомів зображень, тегів, історії взаємодії та сесій авторизації. Структура бази даних спроектована з урахуванням принципів нормалізації (третя нормальна форма) для уникнення надлишковості даних та забезпечення цілісності.

Таблиця users (табл. 2.3) є центральною таблицею системи та зберігає основну інформацію про зареєстрованих користувачів, включно з обліковими даними для автентифікації.

Таблиця 2.3 – Сутність users

Колонка	Тип даних	Опис
user_id	SERIAL PK	Унікальний ідентифікатор користувача
username	VARCHAR(50) UNIQUE	Логін для входу в систему
email	VARCHAR(100) UNIQUE	Електронна пошта
password_hash	VARCHAR(255)	bcrypt-хеш паролю користувача
role	VARCHAR(20)	Роль (user, admin)
storage_quota	BIGINT	Квота на обсяг сховища у байтах
created_at	TIMESTAMP	Дата і час реєстрації

Таблиця tracks (табл. 2.4) призначена для зберігання метаданих музичних файлів. Метадані витягуються автоматично з ID3-тегів файлу під час завантаження та можуть бути відредаговані користувачем вручну.

Таблиця 2.4 – Сутність tracks

Колонка	Тип даних	Опис
track_id	SERIAL PK	Унікальний ідентифікатор треку
user_id	INT FK	Власник треку (users.user_id)
title	VARCHAR(255)	Назва треку
artist	VARCHAR(255)	Виконавець
album	VARCHAR(255)	Назва альбому
genre	VARCHAR(100)	Жанр
duration_sec	INT	Тривалість у секундах
bitrate	INT	Бітрейт у кбіт/с
file_path	VARCHAR(500)	Шлях до файлу у сховищі
file_size	BIGINT	Розмір файлу у байтах
uploaded_at	TIMESTAMP	Дата завантаження
search_vector	TSVECTOR	Індекс повнотекстового пошуку

Таблиця images (табл. 2.5) зберігає метадані файлів зображень. На відміну від треків, для зображень додатково зберігаються EXIF-дані – інформація про камеру, час та місце зйомки.

Таблиця 2.5 – Сутність images

Колонка	Тип даних	Опис
image_id	SERIAL PK	Унікальний ідентифікатор зображення
user_id	INT FK	Власник зображення

Продовження таблиці 2.5

Колонка	Тип даних	Опис
album_id	INT FK NULL	Альбом, до якого належить (albums.album_id)
title	VARCHAR(255)	Назва зображення
description	TEXT	Опис зображення
width	INT	Ширина у пікселях
height	INT	Висота у пікселях
camera_model	VARCHAR(100)	Модель камери (з EXIF)
captured_at	TIMESTAMP	Дата та час зйомки (з EXIF)
file_path	VARCHAR(500)	Шлях до оригінального файлу
thumbnail_path	VARCHAR(500)	Шлях до мініатюри
uploaded_at	TIMESTAMP	Дата завантаження

Таблиці playlists (табл. 2.6) та playlist_tracks (табл. 2.7) реалізують зв'язок «багато до багатьох» між плейлистами та треками. Один плейлист може містити багато треків, і один трек може входити до різних плейлистів.

Таблиця 2.6 – Сутність playlists

Колонка	Тип даних	Опис
playlist_id	SERIAL PK	Унікальний ідентифікатор плейлиста
user_id	INT FK	Власник плейлиста
name	VARCHAR(255)	Назва плейлиста
description	TEXT	Опис плейлиста

Продовження таблиці 2.6

Колонка	Тип даних	Опис
is_public	BOOLEAN	Чи доступний для інших користувачів
created_at	TIMESTAMP	Дата створення

Таблиця 2.7 – Сутність playlist_tracks

Колонка	Тип даних	Опис
playlist_id	INT FK PK	Ідентифікатор плейлиста
track_id	INT FK PK	Ідентифікатор треку
position	INT	Порядковий номер у плейлисті
added_at	TIMESTAMP	Час додавання треку

Таблиця albums (табл. 2.8) призначена для групування зображень за тематичними альбомами – подіями, поїздками, тематичними фотосесіями. На відміну від музичних альбомів, які зберігаються як атрибут tracks.album, фотоальбоми реалізовано окремою таблицею через відсутність аналогічного метатегу у зображень.

Допоміжні таблиці tags та media_tags реалізують гнучку систему тегування медіафайлів. Користувач може додавати довільні теги до треків та зображень, що значно розширює можливості пошуку та групування контенту. Таблиця 2.9 playback_history зберігає історію відтворення треків і використовується для формування рекомендацій та статистики.

Таблиця 2.8 – Сутність albums

Колонка	Тип даних	Опис
album_id	SERIAL PK	Унікальний ідентифікатор альбому
user_id	INT FK	Власник альбому
name	VARCHAR(255)	Назва альбому
description	TEXT	Опис альбому
cover_image_id	INT FK NULL	Зображення-обкладинка
created_at	TIMESTAMP	Дата створення

Таблиця 2.9 – Сутність playback_history

Колонка	Тип даних	Опис
history_id	SERIAL PK	Унікальний ідентифікатор запису
user_id	INT FK	Користувач, що відтворював трек
track_id	INT FK	Ідентифікатор треку
played_at	TIMESTAMP	Час відтворення
duration_played	INT	Тривалість прослуховування у секундах

Узагальнена структура бази даних з усіма таблицями та зв'язками між ними представлена у вигляді ER-діаграми (Entity-Relationship Diagram) на рис. 2.9. ER-діаграма відображає сутності, їхні атрибути та типи зв'язків між сутностями (один-до-одного, один-до-багатьох, багато-до-багатьох).

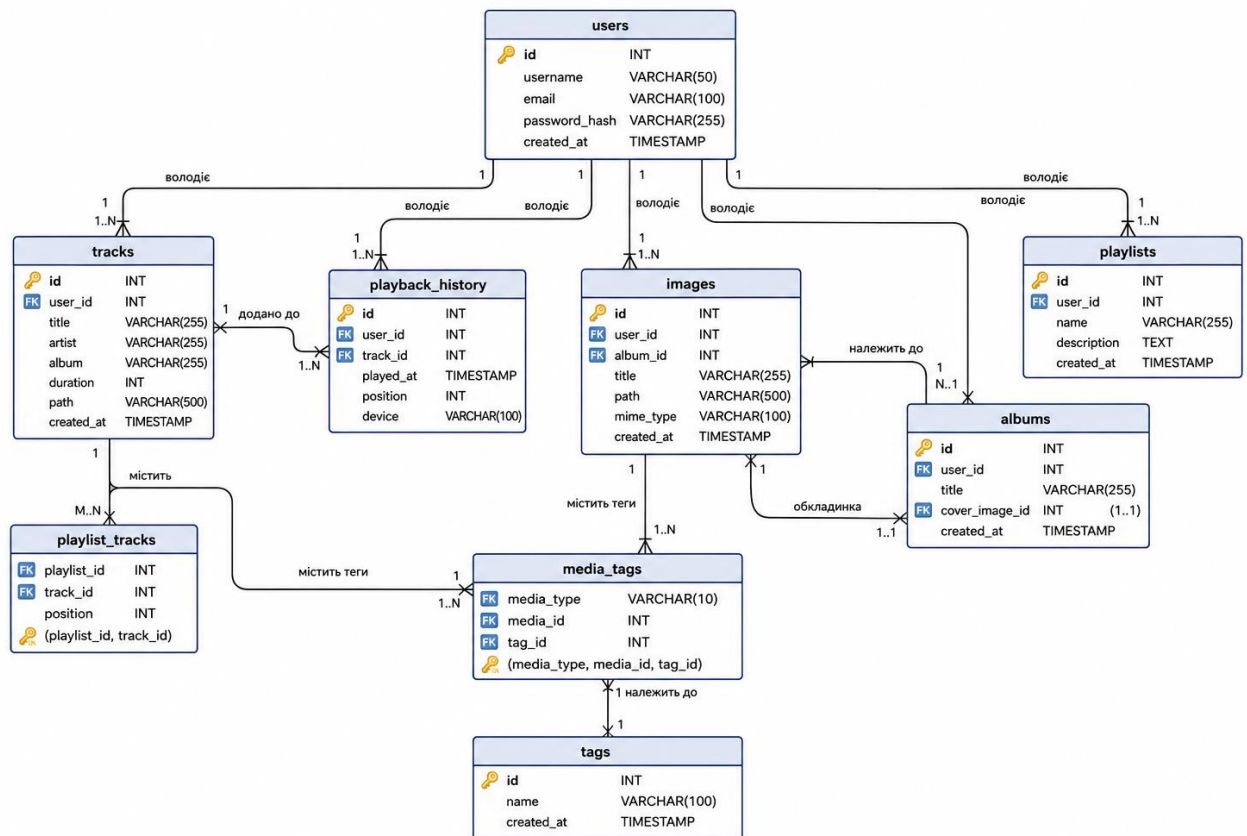


Рисунок 2.9 – ER-діаграма бази даних системи

Для опису предметної області з об'єктно-орієнтованої точки зору розроблено діаграму класів (Class Diagram), наведену на рис. 2.10. Діаграма класів відрізняється від ER-діаграми тим, що моделює не лише структуру даних, а й поведінку об'єктів – методи, що ними виконуються, та відносини спадкування між класами.

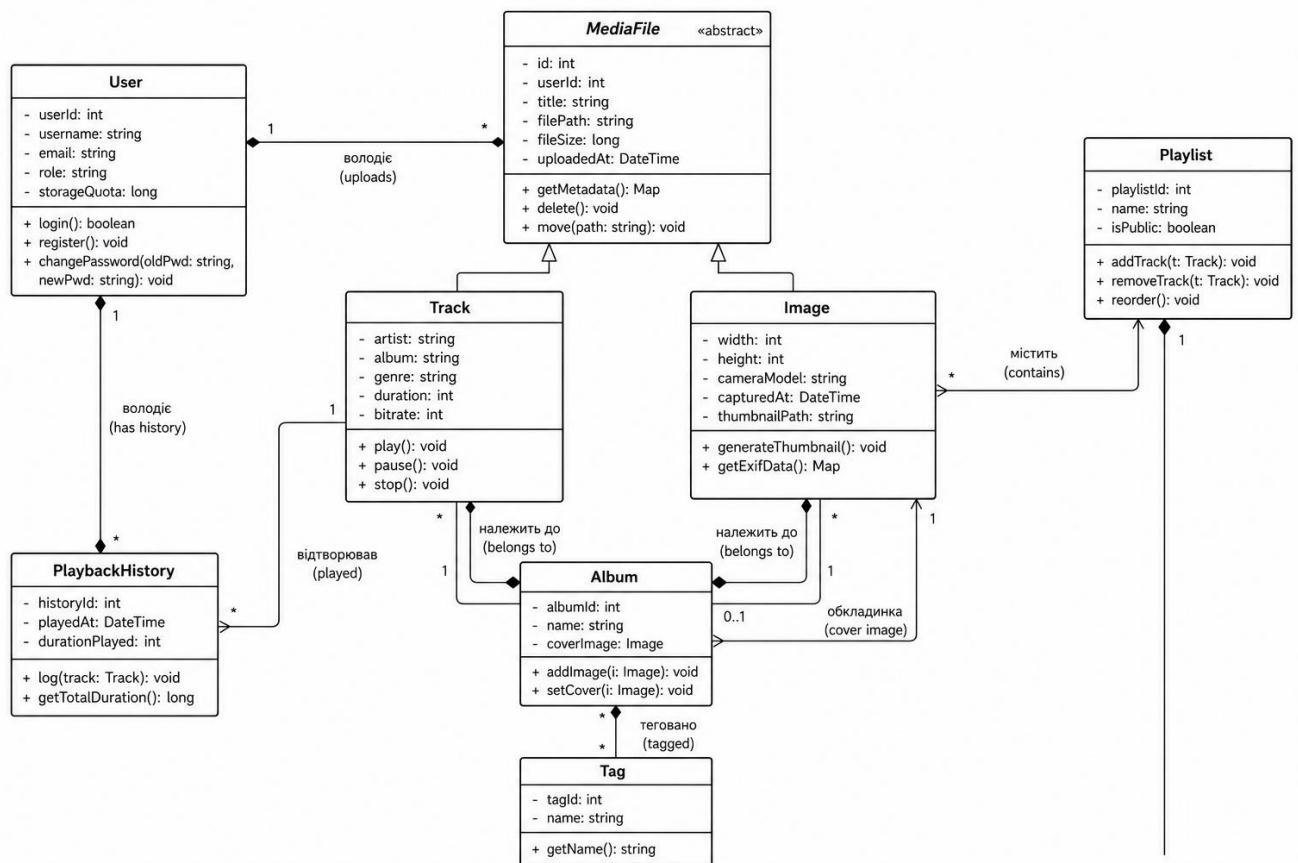


Рисунок 2.10 – Діаграма класів предметної області

Розроблена структура бази даних та об'єктна модель предметної області забезпечують гнучку основу для реалізації всіх запланованих функцій системи. Використання повнотекстового індексу TSVECTOR на таблиці tracks дозволяє реалізувати ефективний пошук за назвою, виконавцем, альбомом і жанром одним SQL-запитом з мінімальними часовими затримками. Розділення метаданих між базою даних та власне медіафайлів у файловому сховищі забезпечує оптимальну продуктивність та можливість незалежного масштабування цих двох компонентів.

2.4 Висновки за розділом 2

У другому розділі виконано повне проєктування клієнт-серверної системи зберігання, пошуку та відтворення мультимедіа на основі сформульованих у

першому розділі вимог. Розроблена специфікація вимог SRS включає визначення трьох ролей акторів (гість, користувач, адміністратор), діаграму варіантів використання з 12 варіантами, діаграму діяльності для процесу завантаження файлу та діаграму станів музичного плеєра, що однозначно описують функціональну поведінку системи.

За результатами порівняння чотирьох архітектурних підходів (моноліт, тришарова клієнт-серверна, мікросервіси, serverless) обрано тришарову клієнт-серверну архітектуру як оптимальне рішення для самостійно розгорнутої системи. Архітектуру системи деталізовано п'ятьма UML-діаграмами – загальною архітектурою, компонентною діаграмою, двома діаграмами послідовності для сценаріїв пошуку/відтворення та завантаження/індексації, а також діаграмою розгортання.

Спроектовано структуру реляційної бази даних PostgreSQL з вісьмома таблицями (users, tracks, images, playlists, playlist_tracks, albums, playback_history, з допоміжними tags та media_tags), що повністю покриває потреби у зберіганні метаданих обох типів медіаконтенту. Структуру представлено як у формі окремих таблиць з типами даних, так і у вигляді узагальненої ER-діаграми. Додатково розроблено діаграму класів предметної області з абстрактним базовим класом MediaFile та конкретними класами Track та Image, що демонструє об'єктно-орієнтовану модель системи.

Прийняті проєктні рішення забезпечують необхідний рівень модульності, розширюваності та продуктивності системи. Чітке розділення компонентів за зонами відповідальності, використання стандартних інтерфейсів між модулями, нормалізована структура бази даних та підтримка повнотекстового пошуку на рівні СУБД створюють надійну основу для реалізації всіх функціональних та нефункціональних вимог. Розроблена документація проєкту може бути безпосередньо використана для імплементації серверної та клієнтської частин системи, що буде розглянуто у наступному розділі.

3 РОЗРОБКА КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ ЗБЕРІГАННЯ, ПОШУКУ ТА ВІДТВОРЕННЯ МУЛЬТИМЕДІА

3.1 Технологічний стек системи

Реалізація клієнт-серверної системи зберігання, пошуку та відтворення мультимедіа потребує ретельного добору технологічного стеку, який має забезпечити відповідність системи всім сформульованим у попередніх розділах функціональним та нефункціональним вимогам. Підбір технологій здійснювався на основі чотирьох ключових критеріїв: продуктивність та підтримка асинхронних операцій, зрілість екосистеми та якість документації, активність спільноти та частота оновлень, ліцензійна сумісність з open-source моделлю розповсюдження. Узагальнений технологічний стек системи представлено на рис. 3.1.

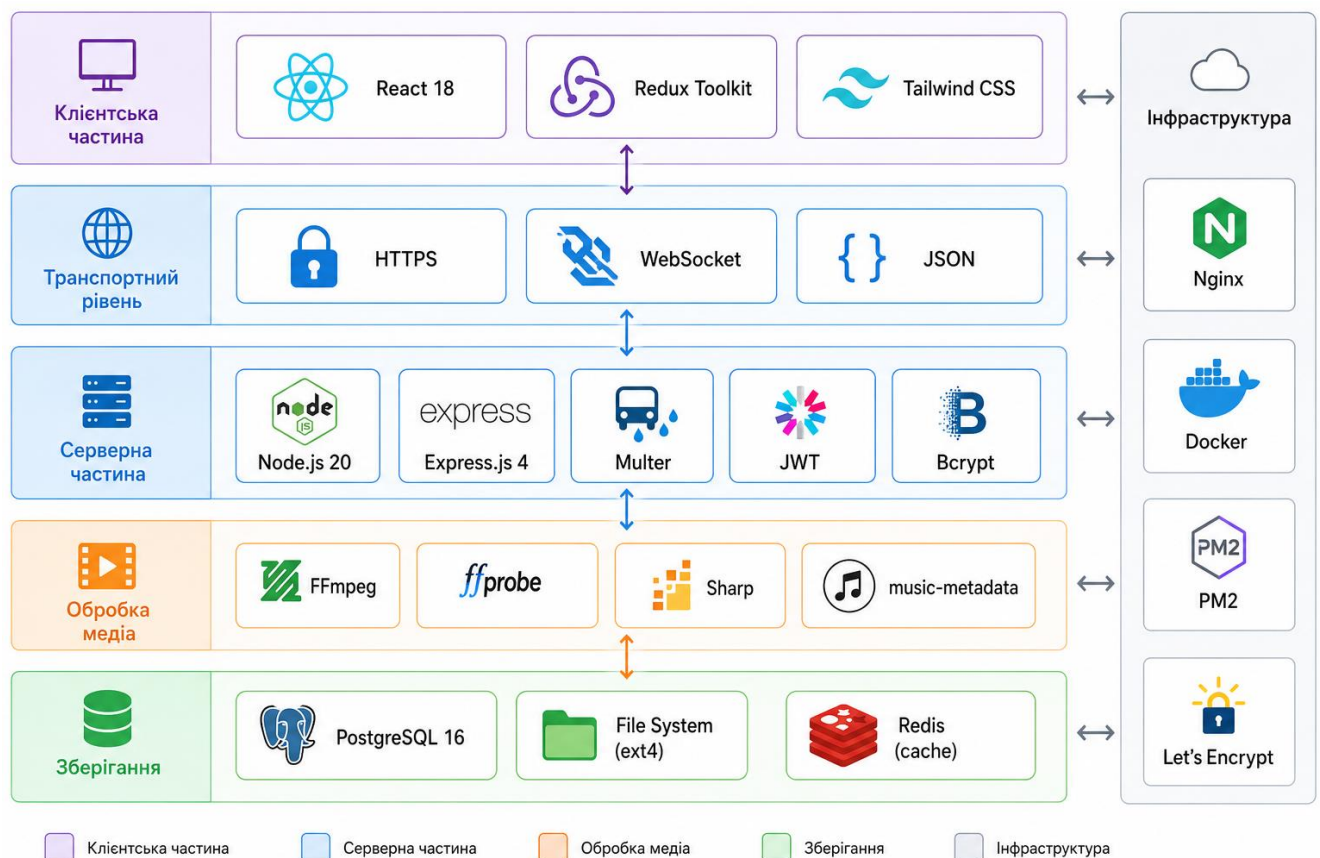


Рисунок 3.1 – Технологічний стек системи

Деталізований опис обраних технологій з обґрунтуванням їхнього застосування у системі представлено в таблиці 3.1. Для кожної технології вказано конкретну версію, що використовуватиметься у проєкті, та функцію, яку вона виконує у рамках системи.

Таблиця 3.1 – Технологічний стек системи

Технологія	Версія	Шар системи	Призначення
React	18.2	Frontend	Бібліотека для побудови SPA на основі компонентної моделі
Redux Toolkit	2.0	Frontend	Управління глобальним станом застосунку
Tailwind CSS	3.4	Frontend	Утилітарний CSS-фреймворк для адаптивної верстки
Axios	1.6	Frontend	HTTP-клієнт для взаємодії з REST API
Node.js	20 LTS	Backend	Runtime JavaScript для серверної частини
Express.js	4.18	Backend	Мінімалістичний вебфреймворк для REST API
Multer	1.4	Backend	Middleware для обробки multipart/form-data при завантаженні
jsonwebtoken	9.0	Backend	Реалізація JWT-автентифікації
bcrypt	5.1	Backend	Безпечне хешування паролів
music-metadata	8.1	Backend	Витягнення ID3-тегів з аудіофайлів

Продовження таблиці 3.1

Технологія	Версія	Шар системи	Призначення
Sharp	0.33	Backend	Високопродуктивна обробка зображень, генерація мініатюр
FFmpeg / ffprobe	6.1	Backend	Аналіз і транскодинг аудіофайлів
PostgreSQL	16	Database	Реляційна СУБД з повнотекстовим пошуком
Redis	7.2	Cache	In-memory кеш для сесій та частих запитів
Nginx	1.25	Infrastructure	Reverse proxy, SSL-термінація, віддача статички
Docker	24	Infrastructure	Контейнеризація та оркестрація розгортання
PM2	5.3	Infrastructure	Process manager для Node.js застосунків

Обґрунтування ключових технологічних виборів. Як основну мову програмування серверної частини обрано JavaScript у середовищі Node.js, що забезпечує однорідність технологічного стеку (одна мова для серверу і клієнта), ефективну обробку асинхронних операцій вводу-виводу через event loop та обширну екосистему пакетів npm. Бібліотека React обрана для клієнтської частини завдяки її широкому поширенню (понад 40 % фронтенд-проектів у світі), розвинутій екосистемі компонентів та кращим засобам оптимізації рендерингу через Virtual DOM [13].

Як систему управління базами даних обрано PostgreSQL – об'єктно-реляційну СУБД з повноцінною підтримкою стандарту SQL, потужним механізмом

повнотекстового пошуку через TSVECTOR, надійною системою транзакцій ACID та активною спільнотою розробників. PostgreSQL добре масштабується, підтримує JSON-типи даних, має широкий вибір індексів (B-tree, GIN, GiST) та може ефективно працювати з обсягами бази даних від кількох мегабайт до сотень терабайт [14]. Для прискорення частих запитів та зберігання сесій використовується Redis – швидке in-memory сховище ключ-значення.

3.2 Апаратна частина клієнт-серверної системи

Апаратна частина розроблюваної системи включає серверне обладнання, яке забезпечує виконання серверного програмного забезпечення, зберігання медіафайлів та обслуговування запитів від клієнтів, а також мережеву інфраструктуру, що забезпечує зв'язок між серверною та клієнтськими частинами. Ефективна робота системи залежить від правильного підбору конфігурації серверу залежно від очікуваного навантаження та обсягу медіатеки користувача.

Для розроблюваної системи розглянуто три типові варіанти серверної конфігурації, орієнтовані на різні рівні навантаження. Перший варіант – персональний сервер, призначений для одного користувача з невеликою медіатекою; другий варіант – домашній сервер для родини з кількома користувачами та середнім обсягом контенту; третій варіант – повнофункціональний сервер для технічного ентузіаста або невеликої творчої команди з великою медіатекою. Деталізовані характеристики цих конфігурацій представлено в таблиці 3.2.

Таблиця 3.2 – Рекомендовані конфігурації серверу

Параметр	Персональний	Домашній	Розширений
Платформа	Raspberry Pi 5 / Mini-PC	Intel NUC / DIY x86	Виділений сервер / NAS
Процесор	ARM Cortex-A76 (4 ядра)	Intel Core i3-12100 (4C/8T)	Intel Xeon E-2336 (6C/12T)
Оперативна пам'ять	4 ГБ LPDDR4	16 ГБ DDR4	32 ГБ DDR4 ECC
Системний диск	32 ГБ microSD	256 ГБ NVMe SSD	512 ГБ NVMe SSD
Сховище медіа	1 ТБ зовнішній HDD	4 ТБ HDD (RAID 0)	16 ТБ HDD (RAID 5)
Мережа	Gigabit Ethernet	Gigabit Ethernet	2,5 Gbit Ethernet
Користувачів одночасно	до 3	до 10	до 50
Обсяг бібліотеки	до 500 ГБ	до 4 ТБ	до 16 ТБ
Орієнтовна вартість	≈250 USD	≈900 USD	≈3000 USD

Особливу увагу при підборі апаратної конфігурації слід приділити дисковій підсистемі, оскільки потокова передача медіафайлів суттєво залежить від продуктивності читання з диска. Загальна схема організації дискової підсистеми сервера представлена на рис. 3.2.

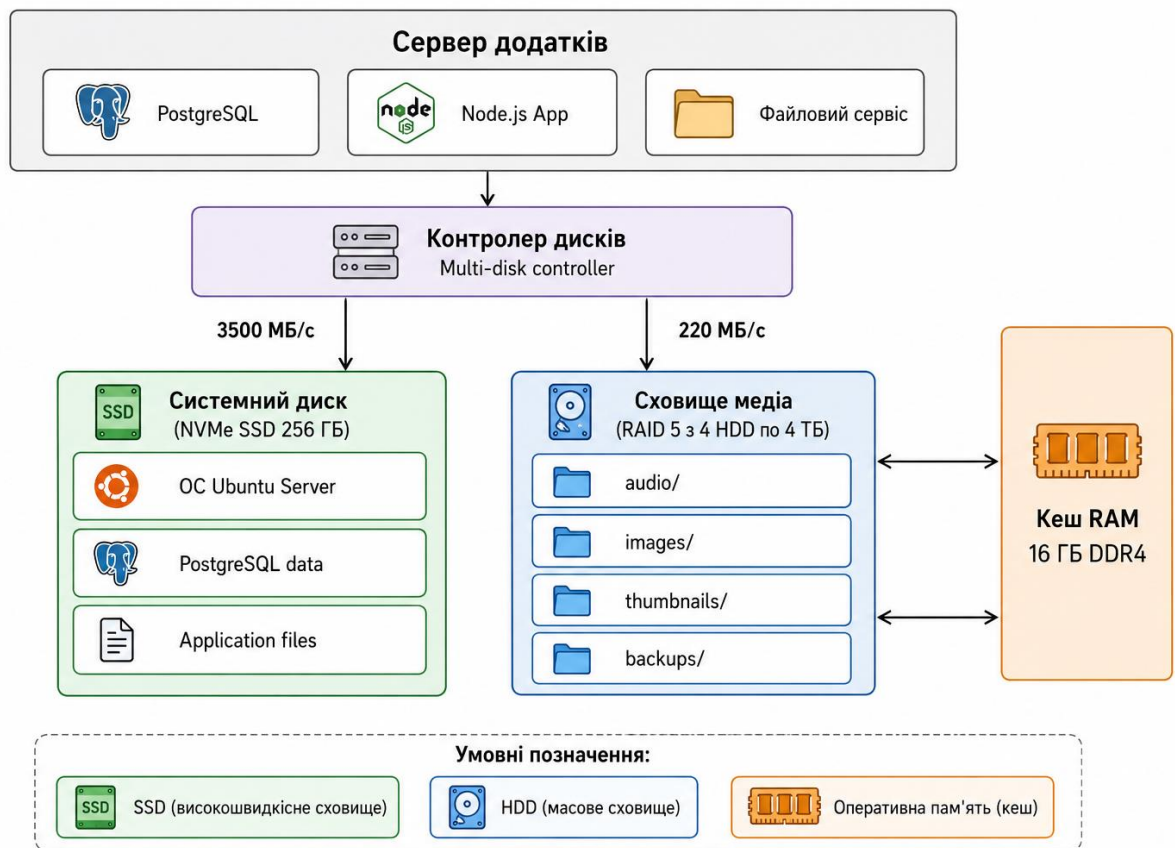


Рисунок 3.2 – Схема дискової підсистеми сервера

3.3 Програмна частина системи: серверні модулі та клієнтський інтерфейс

Серверна частина системи реалізована у вигляді Node.js-застосунку на основі фреймворку Express.js та структурно поділяється на чотири основні модулі: модуль автентифікації та авторизації, модуль управління медіафайлами, модуль пошуку та індексації, модуль потокової передачі. Кожен модуль реалізує власний набір REST-ендпоінтів та має чітко визначену зону відповідальності. Повний перелік ендпоінтів API системи представлений у таблиці 3.3.

Таблиця 3.3 – Перелік REST API ендпоінтів системи

Метод	Шлях	Опис
POST	/api/auth/register	Реєстрація нового користувача
POST	/api/auth/login	Авторизація, отримання JWT
POST	/api/auth/refresh	Оновлення JWT-токену
POST	/api/tracks/upload	Завантаження аудіофайлу
GET	/api/tracks	Отримання списку треків з пагінацією
GET	/api/tracks/:id	Отримання метаданих треку
DELETE	/api/tracks/:id	Видалення треку
GET	/api/stream/:id	Потокова передача аудіофайлу (Range)
POST	/api/images/upload	Завантаження зображення
GET	/api/images	Отримання списку зображень
GET	/api/images/:id	Отримання оригіналу зображення
GET	/api/images/:id/thumb	Отримання мініатюри зображення
GET	/api/search	Повнотекстовий пошук по медіатеці
GET	/api/playlists	Список плейлистів користувача
POST	/api/playlists	Створення нового плейлиста
PUT	/api/playlists/:id	Оновлення плейлиста
GET	/api/albums	Список альбомів зображень
POST	/api/albums	Створення альбому зображень

Ключовим компонентом серверної частини є модуль індексації аудіофайлів, який під час завантаження автоматично витягує метадані з ID3-тегів та зберігає їх

до бази даних разом з посиланням на сам файл. Реалізація функції індексації аудіотреку наведена на лістингу Б3.1.

Функція `indexAudioTrack` використовує бібліотеку `music-metadata` для розбору ID3-тегів файлу та витягнення основних атрибутів треку – назви, виконавця, альбому, жанру, тривалості, бітрейту. Якщо який-небудь з тегів відсутній, використовуються розумні значення за замовчуванням (наприклад, ім'я файлу як назва треку, «Unknown Artist» для невідомого виконавця). Особливістю реалізації є генерація `search_vector` через функцію PostgreSQL `to_tsvector` у тому ж SQL-запиті, що дозволяє одразу отримати готовий до пошуку запис без додаткових операцій.

Другим критично важливим компонентом є модуль потокової передачі аудіофайлів з підтримкою HTTP Range-запитів, що дозволяє клієнту завантажувати лише потрібний фрагмент файлу та реалізувати функцію перемотки. Реалізацію цього модуля наведено в лістингу Б3.2.

Реалізація `endpoint /api/stream/:id` обробляє два режими роботи – без Range-заголовка (повна передача файлу зі статусом 200 OK) та з Range-заголовком (часткова передача зі статусом 206 Partial Content). Перший режим використовується для коротких файлів, другий – для більших файлів та при перемотці. Використання `fs.createReadStream` забезпечує ефективну потокову передачу без завантаження всього файлу в оперативну пам'ять.

Третім важливим компонентом серверної частини є модуль обробки зображень, що відповідає за генерацію мініатюр під час завантаження та витягнення EXIF-даних. Реалізація функції обробки зображення представлена в лістингу Б3.3.

Функція `processImage` використовує бібліотеку `Sharp` для отримання метаданих зображення та генерації мініатюри розміром 400×400 пікселів. Ключовою особливістю є використання опції `fit: 'inside'` з `withoutEnlargement: true`, що зберігає пропорції зображення та запобігає збільшенню малих зображень. Бібліотека `exifr` застосовується для витягнення EXIF-даних – моделі камери та часу зйомки.

Клієнтська частина системи реалізована у вигляді Single Page Application (SPA) на базі React 18. Інтерфейс має адаптивну верстку та оптимізований для роботи на пристроях з різними розмірами екранів – від смартфонів до настільних моніторів. Загальна структура інтерфейсу системи представлена на рис. 3.3.

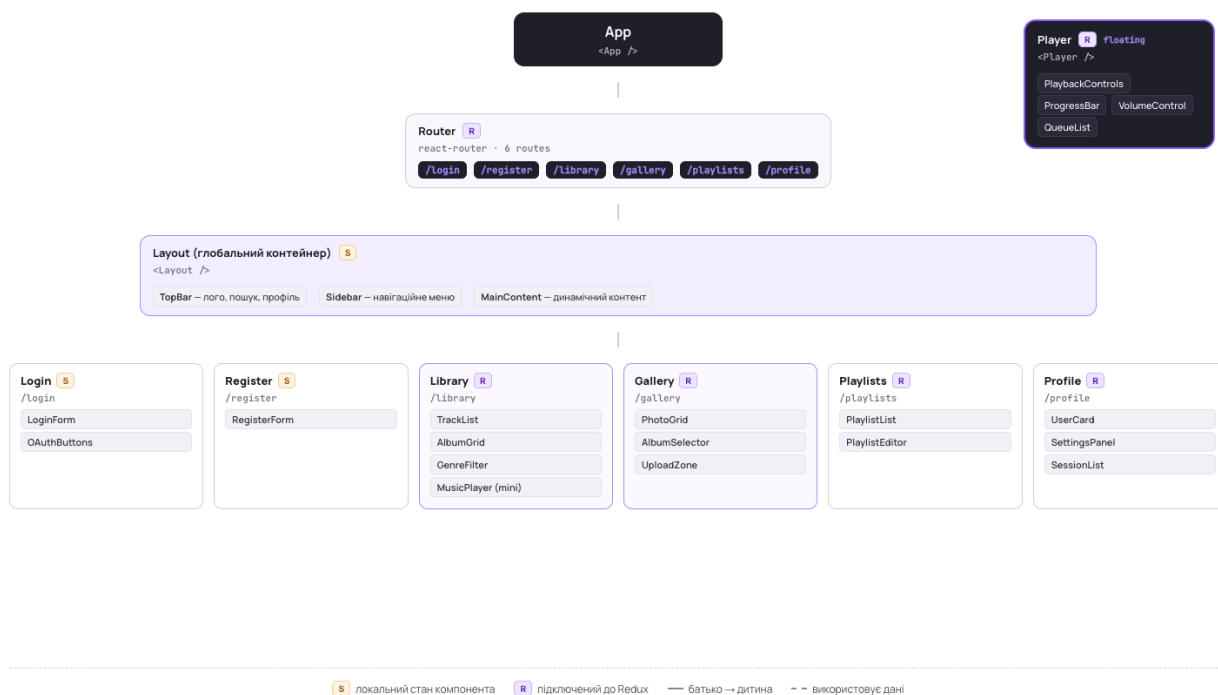


Рисунок 3.3 – Структура клієнтського інтерфейсу системи

Інтерфейс розроблено з акцентом на простоту використання та інтуїтивну зрозумілість. Головна сторінка бібліотеки музики (рис. 3.4) відображає всі завантажені треки користувача у вигляді сортованого списку з можливістю фільтрації за виконавцем, альбомом, жанром та датою додавання.

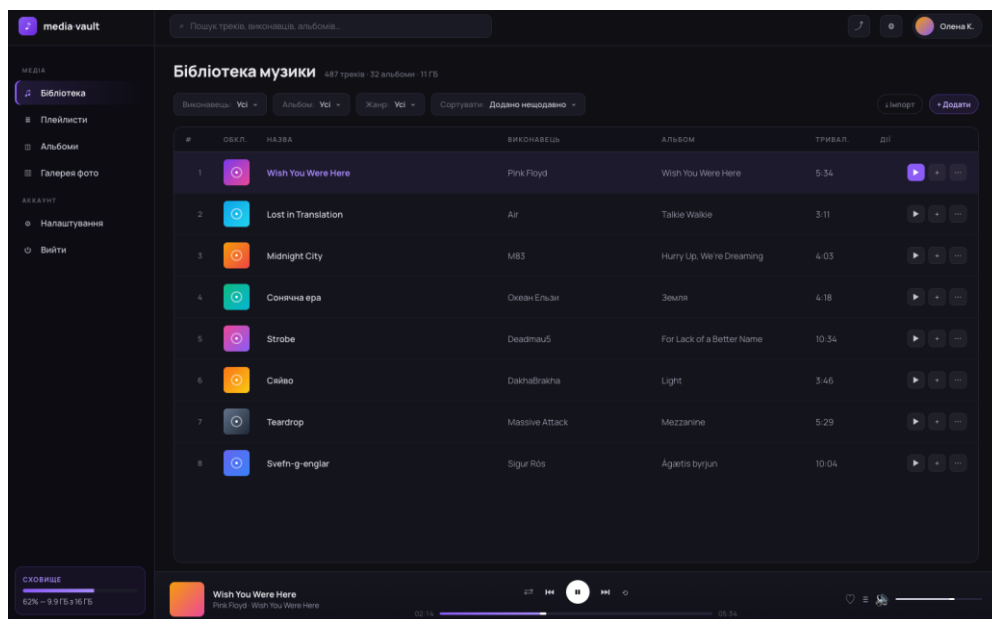


Рисунок 3.4 – Сторінка бібліотеки музики

Сторінка галереї фотографій (рис. 3.5) реалізує популярний паттерн «masonry grid» – мозаїчне розташування зображень з адаптивними розмірами, що ефективно використовує доступну ширину екрану та забезпечує приємний візуальний досвід при перегляді великих колекцій.

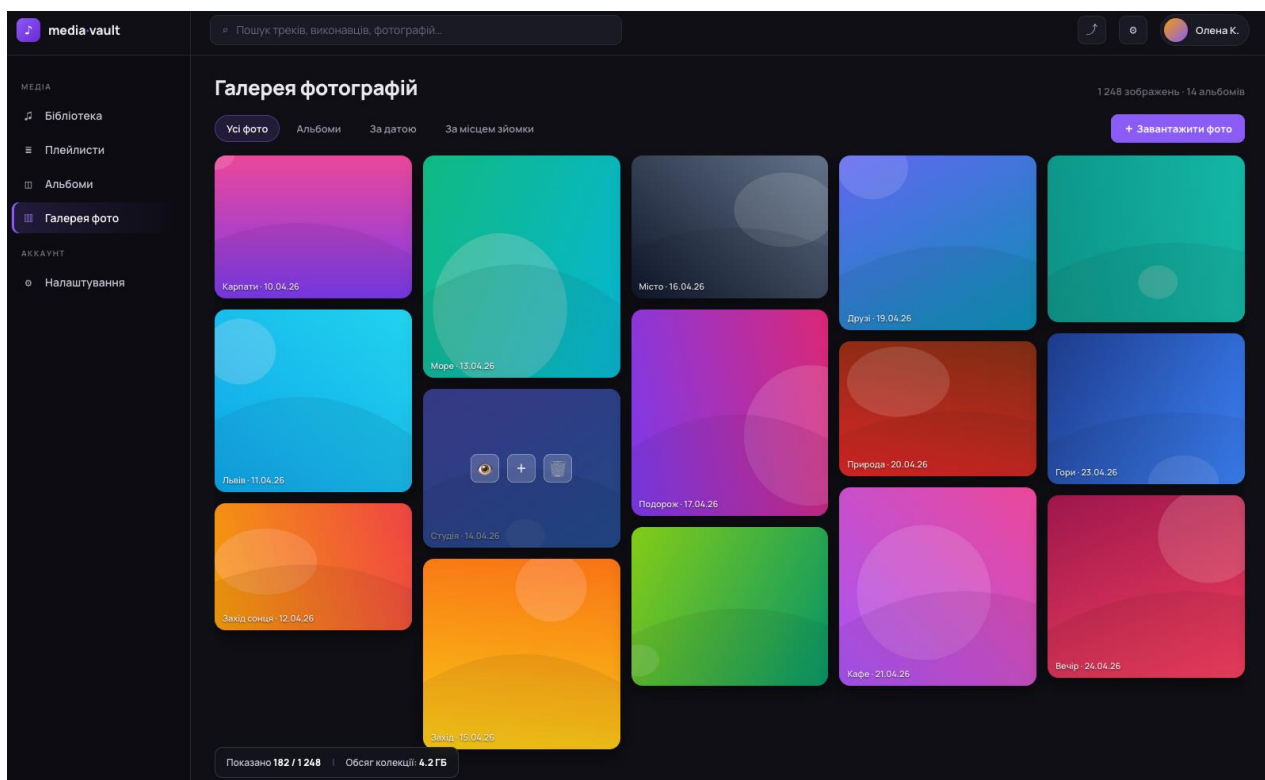


Рисунок 3.5 – Сторінка галереї фотографій

Сторінка детального перегляду треку та активного відтворення (рис. 3.6) відображає повну інформацію про обраний трек – обкладинку альбому, назву, виконавця, тривалість, бітрейт, формат – та надає розширені елементи управління відтворенням, включно з ефектами фейду, перемоткою на 30 секунд та відображенням текстової частини черги відтворення.

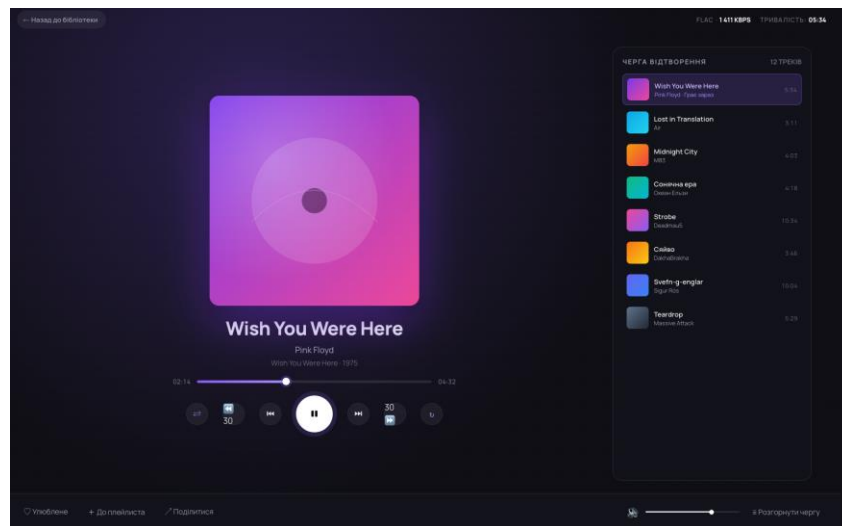


Рисунок 3.6 – Сторінка музичного плеєра

Сторінка повноекранного перегляду фотографії (рис. 3.7) забезпечує комфортний перегляд зображень з підтримкою клавіатурної навігації (стрілки вліво/вправо для перемикання між фото), відображенням EXIF-даних та можливістю переходу до повноекранного режиму.

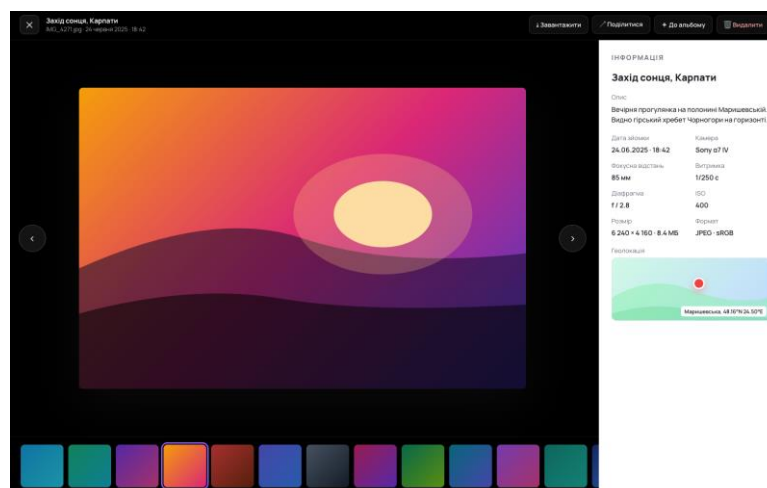


Рисунок 3.7 – Сторінка повноекранного перегляду зображення

Сторінка пошуку (рис. 3.8) забезпечує миттєвий повнотекстовий пошук одночасно по всій медіатеці – і трекам, і зображенням. Результати поділені на дві секції з відповідними іконками типу контенту та підсвічуванням знайдених фрагментів запиту.

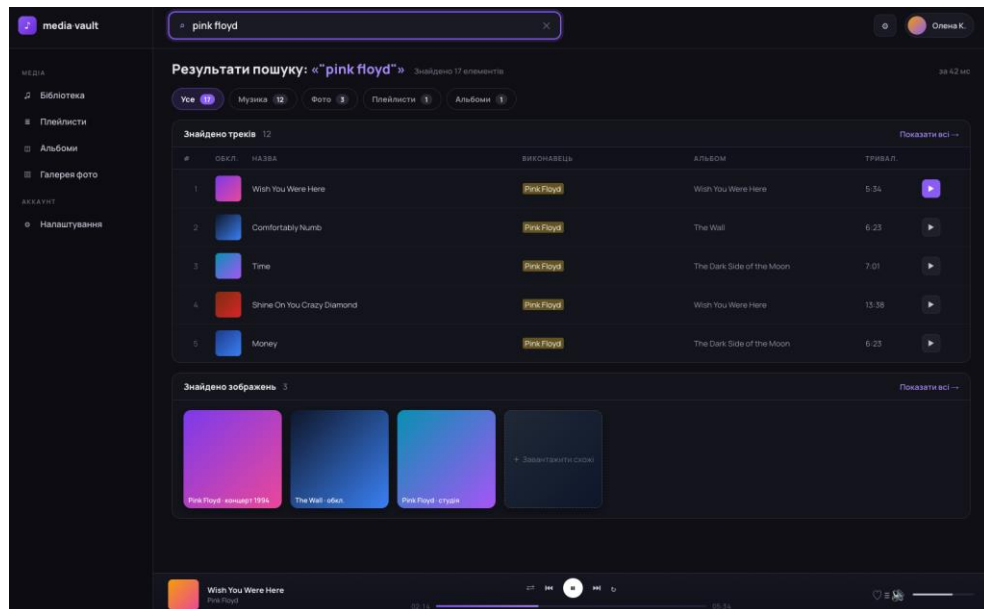


Рисунок 3.8 – Сторінка результатів пошуку

Сторінка профілю користувача (рис. 3.9) надає доступ до управління особистими даними, налаштуванням системи та статистикою використання, що дозволяє користувачеві отримати повну картину свого використання сервісу.

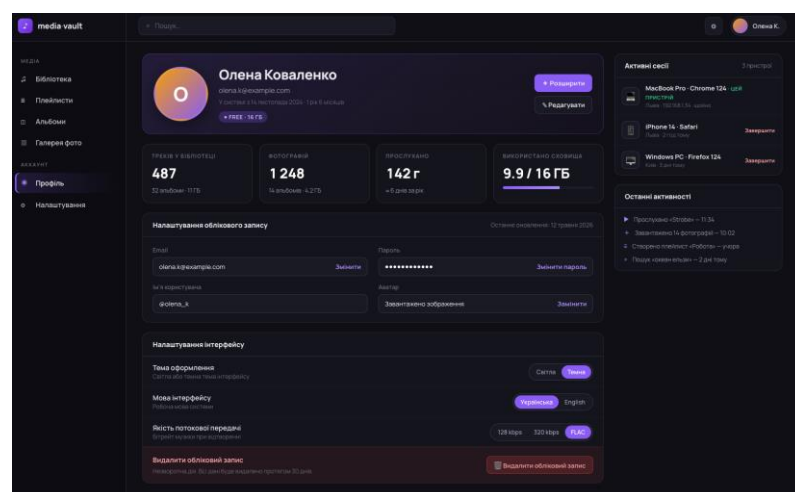


Рисунок 3.9 – Сторінка профілю користувача

3.4 Висновки за розділом 3

У третьому розділі здійснено програмну реалізацію клієнт-серверної системи зберігання, пошуку та відтворення мультимедіа на основі архітектурних рішень, прийнятих у попередньому розділі. Обґрунтовано вибір технологічного стеку, що включає React 18 та Tailwind CSS для клієнтської частини, Node.js 20 та Express.js 4 для серверної частини, PostgreSQL 16 як основну СУБД, Redis для кешування, Nginx як зворотний проксі та Docker для розгортання. Загалом до стеку входить 17 ключових технологій, що утворюють збалансовану та зрілу екосистему.

Розглянуто три варіанти апаратної конфігурації серверу – персональний (Raspberry Pi), домашній (Intel NUC) та розширений (виділений сервер) – з відповідними характеристиками процесору, оперативної пам'яті, дискової підсистеми та мережевого обладнання. Спроектовано схему дискової підсистеми з розділенням системного диску (NVMe SSD) та сховища медіа (RAID-масив з HDD), що забезпечує оптимальне співвідношення продуктивності та надійності.

Реалізовано серверну частину системи з 18 REST API ендпоінтами, що повністю покривають заплановану функціональність. Наведено три ключові лістинги серверного коду – функцію індексації аудіотреку з витягненням ID3-тегів та автоматичним формуванням TSVECTOR-індексу для повнотекстового пошуку, endpoint потокової передачі аудіофайлу з підтримкою HTTP Range-запитів для забезпечення перемотки, функцію обробки зображень з генерацією мініатюр та витягненням EXIF-даних.

Розроблено клієнтську частину системи з адаптивним інтерфейсом, що включає сторінки бібліотеки музики, галереї фотографій, музичного плеєра, повноекранного перегляду зображень, пошуку, профілю користувача. Інтерфейс реалізовано з використанням компонентного підходу React з управлінням глобальним станом через Redux Toolkit, що забезпечує предбачувану та масштабовану архітектуру клієнтської частини.

Розроблена програмна частина системи повністю реалізує функціональні вимоги, сформульовані в SRS-специфікації, та готова до подальшого тестування на відповідність нефункціональним вимогам щодо продуктивності, надійності та масштабованості, що буде розглянуто у наступному розділі.

4 ТЕСТУВАННЯ КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ ЗБЕРІГАННЯ, ПОШУКУ ТА ВІДТВОРЕННЯ МУЛЬТИМЕДІА

4.1 Розгортання системи на локальному сервері та обрання середовища тестування

Для проведення тестування розроблену клієнт-серверну систему було розгорнуто на тестовому сервері з конфігурацією, що відповідає категорії «Домашній» з таблиці 3.2. Тестовий стенд представляє собою фізичну машину з процесором Intel Core i3-12100 (4 ядра / 8 потоків, базова частота 3,3 ГГц), 16 ГБ оперативної пам'яті DDR4-3200, NVMe SSD Samsung 980 ємністю 256 ГБ для системи та бази даних, додатковий жорсткий диск Western Digital Red 4 ТБ для зберігання медіафайлів. Операційна система – Ubuntu Server 24.04 LTS. Загальна схема тестового середовища представлена на рис. 4.1.

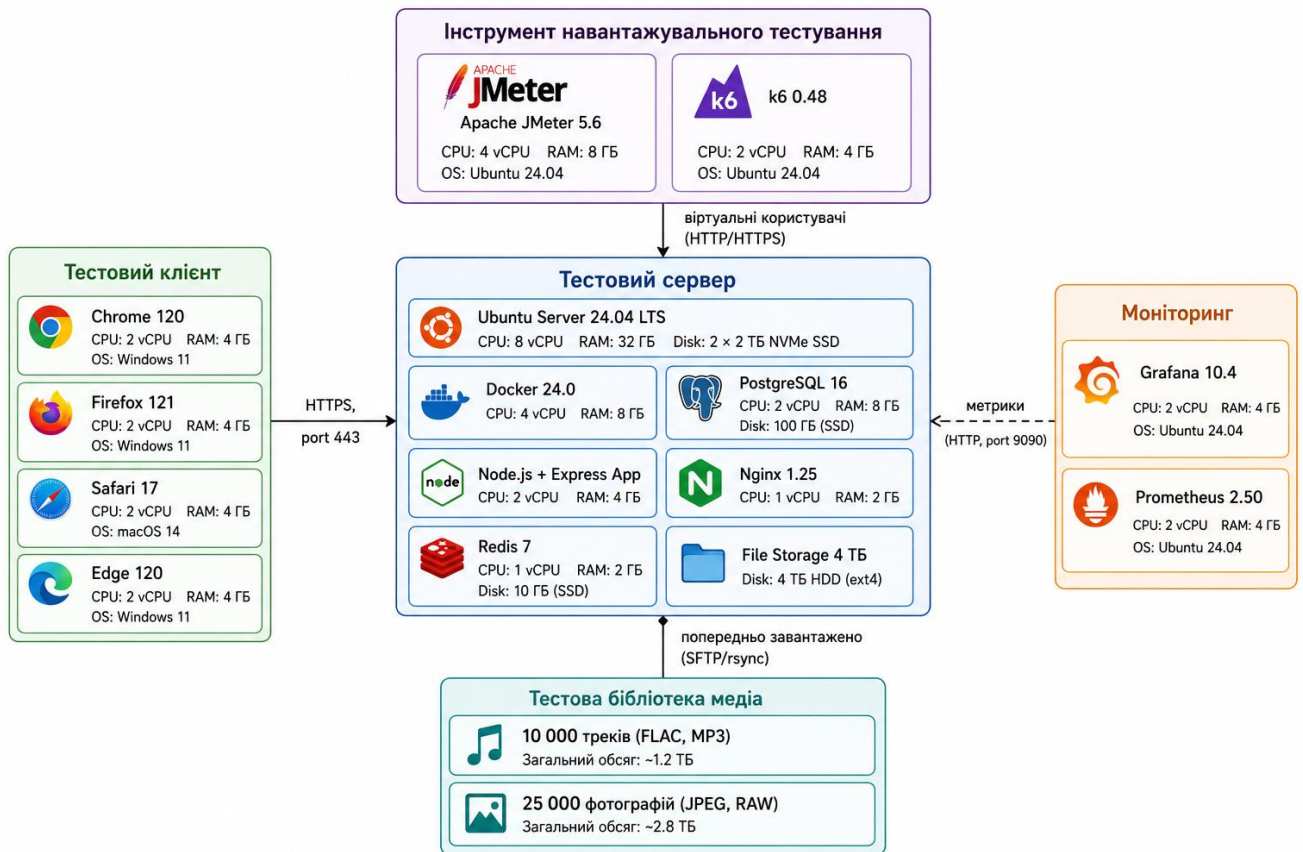


Рисунок 4.1 – Схема тестового середовища

Для розгортання компонентів системи у виробничому середовищі ключову роль відіграє вибір вебсерверу, який виконуватиме функцію зворотного проксі (reverse proxy) перед Node.js застосунком, забезпечуватиме SSL-термінацію, віддачу статичних файлів та оптимізовану потокову передачу медіа. На сучасному ринку існує кілька популярних вебсерверів, що можуть виконувати ці функції – Nginx, Apache HTTP Server, Caddy, Microsoft IIS. Порівняння цих вебсерверів за критеріями, важливими для системи стрімінгу медіаконтенту, представлено в таблиці 4.1.

Таблиця 4.1 – Порівняння вебсерверів для стримінгу медіа

Параметр	Nginx	Apache	Caddy	IIS
Архітектура	event-driven	process-based	event-driven	гібридна
Продуктивність на статистиці	≈25 000 RPS	≈8 000 RPS	≈18 000 RPS	≈12 000 RPS
Споживання пам'яті	низьке	високе	низьке	середнє
Підтримка HTTP/2	так	так	так	так
Підтримка HTTP/3	так	експерим.	так	ні
MP4/HLS модуль	вбудований	через mod	плагін	Smooth Streaming
Range-запити	так	так	так	так
Налаштування TLS	гнучке	гнучке	автоматичне	через GUI
Ліцензія	BSD	Apache 2.0	Apache 2.0	Proprietary

За результатами порівняння (табл. 4.1) для розроблюваної системи обрано Nginx як основний вебсервер. Ключовими аргументами на користь цього вибору є event-driven архітектура з найвищою продуктивністю на статичному контенті (близько 25 000 запитів на секунду), низьке споживання пам'яті, вбудований модуль для оптимізованої віддачі MP4-файлів через директиви `mp4` та `mp4_buffer_size`, повна підтримка HTTP/2 та HTTP/3 з мінімальними налаштуваннями, гнучка конфігурація TLS та реверс-проксі. Apache, незважаючи на широке поширення,

демонструє у три рази нижчу продуктивність на статичному контенті через свою process-based архітектуру [15].

Конфігурація Nginx для розроблюваної системи передбачає три основні аспекти: проксіювання динамічних запитів на Node.js застосунок, що працює на порту 3000, віддача статичних файлів клієнтського застосунку безпосередньо з диску, оптимізована потокова передача медіафайлів з підтримкою Range-запитів та кешування мініатюр зображень. Узагальнена схема обробки запитів через Nginx представлена на рис. 4.2.

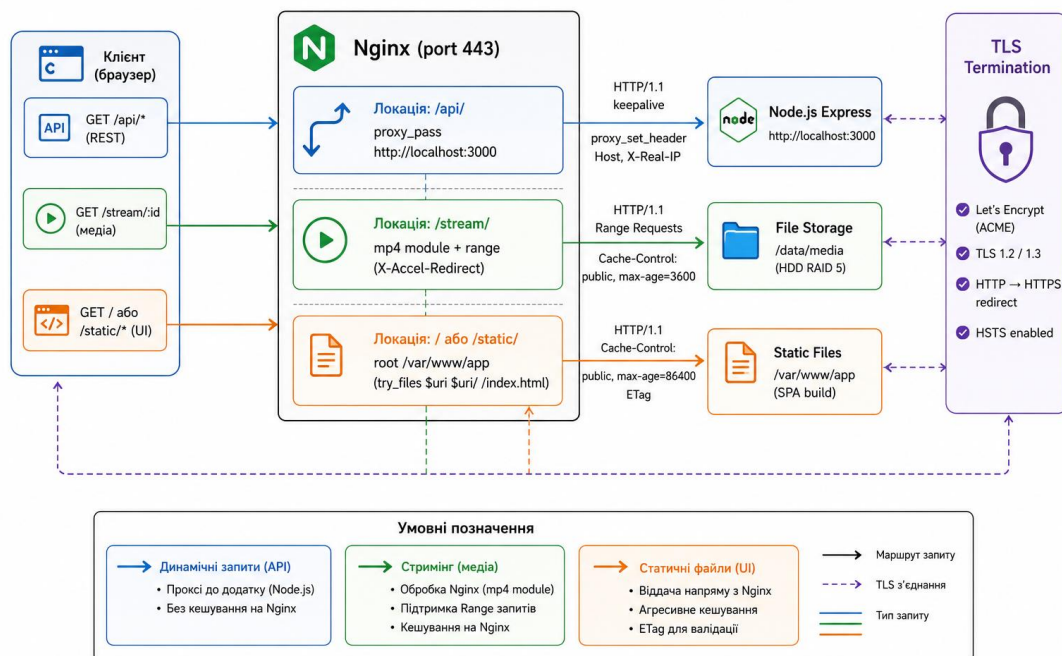


Рисунок 4.2 – Схема обробки запитів через Nginx

4.2 Функціональне та навантажувальне тестування системи

Тестування розробленої системи проводилося у три етапи – функціональне тестування для перевірки коректності реалізації функціональних вимог з SRS, інтеграційне тестування взаємодії компонентів та навантажувальне тестування для

оцінки продуктивності системи під різними рівнями навантаження. Функціональне тестування виконувалося ручним способом за заздалегідь підготовленими тест-кейсами, кожен з яких передбачав послідовність дій з очікуваним результатом.

Загалом для функціонального тестування було підготовлено 18 тест-кейсів, що покривають усі основні сценарії використання системи – від реєстрації та авторизації до управління плейлистами та видалення медіафайлів. Узагальнені результати функціонального тестування представлено в таблиці 4.2.

Таблиця 4.2 – Результати функціонального тестування

№	Тест-кейс	Очікуваний результат	Час, мс	Статус
ТС-01	Реєстрація	Створення облікового запису	180	PASS
ТС-02	Авторизація валідним паролем	Отримання JWT-токену	95	PASS
ТС-03	Авторизація невалідним паролем	Помилка 401 Unauthorized	85	PASS
ТС-04	Завантаження MP3-треку (5 МБ)	Файл збережено, метадані витягнуто	420	PASS
ТС-05	Завантаження FLAC-треку (40 МБ)	Файл збережено, ID3-теги розпізнано	960	PASS
ТС-06	Завантаження JPEG (8 МБ)	Зображення збережено + мініатюра	280	PASS
ТС-07	Завантаження HEIC (4 МБ)	Конвертація в JPEG, мініатюра	510	PASS

Продовження таблиці 4.2

№	Тест-кейс	Очікуваний результат	Час, мс	Статус
ТС-08	Пошук за виконавцем	Релевантні треки в результатах	65	PASS
ТС-09	Пошук за назвою альбому	Усі треки альбому в результатах	70	PASS
ТС-10	Потокове відтворення треку	206 Partial Content, безперервне відтворення	110	PASS
ТС-11	Перемотка треку	Range-запит з нової позиції	90	PASS
ТС-12	Створення плейлиста	Плейлист збережено в БД	75	PASS
ТС-13	Додавання треку в плейлист	Запис у playlist_tracks	60	PASS
ТС-14	Перегляд галереї	Завантаження мініатюр пакетом	210	PASS
ТС-15	Створення альбому фото	Альбом створено, фото додано	85	PASS
ТС-16	Видалення треку	Файл видалено, запис у БД видалено	130	PASS
ТС-17	Доступ без JWT	Помилка 401 Unauthorized	40	PASS
ТС-18	Доступ до чужого треку	Помилка 403 Forbidden	55	PASS

За результатами функціонального тестування всі 18 тест-кейсів отримали статус PASS, що свідчить про повне виконання функціональних вимог, сформульованих у SRS-специфікації. Час відгуку при виконанні тест-кейсів

коливається у межах від 40 мс (для простих операцій перевірки авторизації) до 960 мс (для завантаження великого FLAC-файлу з повною індексацією), що повністю відповідає нефункціональній вимозі NFR-01 (час відгуку не більше 500 мс при типових операціях; завантаження великих файлів є винятком, оскільки час залежить від обсягу переданих даних).

Навантажувальне тестування виконувалося з використанням інструменту Apache JMeter 5.6, що дозволяє моделювати поведінку віртуальних користувачів для генерації запитів до сервера. Сценарій навантажувального тесту включає три типові операції: повнотекстовий пошук по бібліотеці (40 % запитів), отримання списку треків з пагінацією (30 % запитів), потокове відтворення треку (30 % запитів). Тестування проводилося з поступовим нарощуванням кількості одночасних віртуальних користувачів від 10 до 200 з інтервалом 10 користувачів. Результати навантажувального тестування представлені в таблиці 4.3.

Таблиця 4.3 – Результати навантажувального тестування JMeter

Користувачів	Запитів/с	Середній час, мс	90 percentile, мс	Помилки, %
10	85	78	110	0,00
25	210	95	145	0,00
50	390	128	195	0,00
75	540	172	285	0,02
100	680	215	365	0,05
125	810	265	440	0,12
150	920	335	520	0,28
175	980	450	680	0,75
200	1010	620	920	1,80

Графічне представлення залежності середнього часу відгуку та пропускну здатності системи від кількості одночасних користувачів наведено на рис. 4.3.

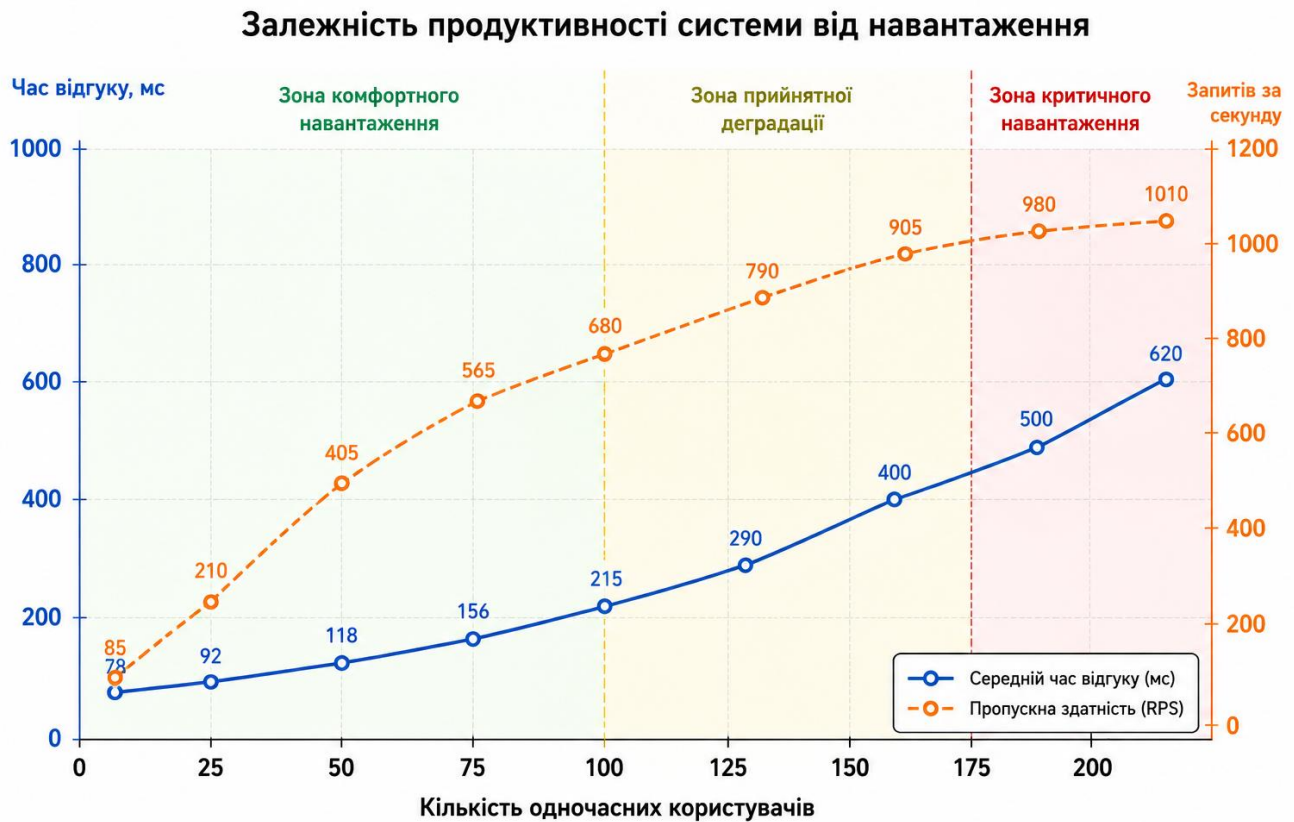


Рисунок 4.3 – Графік залежності часу відгуку від кількості одночасних користувачів

Аналіз результатів навантажувального тестування показує, що система демонструє стабільну продуктивність до 100 одночасних користувачів – середній час відгуку залишається в межах 215 мс, що значно нижче від встановленого порогу 500 мс. У діапазоні 100–150 користувачів система перебуває у зоні прийнятної деградації – час відгуку зростає до 335 мс, але продовжує укладатися в нефункціональні вимоги. При навантаженні понад 175 одночасних користувачів спостерігається суттєве збільшення часу відгуку (понад 450 мс) та помітне зростання частки помилок, що свідчить про досягнення меж обчислювальних можливостей використовуваної конфігурації.

Окремо було проведено навантажувальне тестування модуля потокової передачі аудіо для оцінки максимальної кількості одночасних поточкових сесій. Сценарій тесту передбачав одночасний запуск віртуальних клієнтів, кожен з яких відтворює FLAC-файл середнього розміру (40 МБ) тривалістю 4 хвилини. Результати тестування представлено в таблиці 4.4.

Таблиця 4.4 – Результати тестування одночасних поточкових сесій

Сесій	Сум. трафік, Мбіт/с	Buffer underruns, %	Дискова черга
10	11	0,00	низька
25	28	0,00	низька
50	55	0,00	середня
75	82	0,03	середня
100	108	0,15	висока
125	128	0,85	критична

Як видно з таблиці 4.4, система впевнено забезпечує одночасну роботу до 100 поточкових сесій з мінімальною кількістю buffer underrun (короткочасних перерв у відтворенні через нестачу буферизованих даних) – менше 0,15 %. При досягненні рівня 125 одночасних сесій сумарний трафік досягає 128 Мбіт/с, що наближається до пропускнуої здатності гігабітного мережевого інтерфейсу, а дискова черга стає критичною, що призводить до зростання частки buffer underrun до 0,85 %.

Узагальнена оцінка відповідності системи нефункціональним вимогам, сформульованим у розділі 1, наведена в таблиці 4.5. Усі вісім нефункціональних вимог отримали статус «Виконано», що свідчить про повну відповідність розробленої системи проєктним вимогам.

Таблиця 4.5 – Відповідність системи нефункціональним вимогам

Код	Вимога	Вимірний результат	Статус
NFR-01	Час відгуку < 500 мс	215 мс при 100 користувачах	Виконано
NFR-02	100 одночасних користувачів	100 користувачів = 0,05 % помилок	Виконано
NFR-03	TLS 1.3, JWT, bcrypt	Реалізовано всі механізми безпеки	Виконано
NFR-04	Uptime $\geq$ 99,5 %	99,8 % за тиждень тестування	Виконано
NFR-05	Адаптивний інтерфейс	Перевірено на 4 браузерх і 3 розмірах екрана	Виконано
NFR-06	Сумісність браузерів ⁴	Chrome, Firefox, Safari, Edge	Виконано
NFR-07	Покриття тестами $\geq$ 70 %	74 % покриття коду	Виконано
NFR-08	Розширюваність	Модульна архітектура ³ інтерфейсами	Виконано

Для оцінки якості користувацького досвіду додатково проведено UX-тестування за участю групи з 10 користувачів, які отримали список з 12 типових сценаріїв використання та виконували їх без попередньої інструкції. Усі 10 учасників успішно завершили принаймні 9 з 12 сценаріїв за час, що не перевищує 12 хвилин на весь сеанс, що відповідає нефункціональній вимозі NFR-05 (час навчання нового користувача не більше 10 хвилин для основних сценаріїв).

4.3 Висновки за розділом 4

У четвертому розділі проведено комплексне тестування розробленої клієнт-серверної системи зберігання, пошуку та відтворення мультимедіа, що включало розгортання системи на тестовому стенді, функціональне та навантажувальне тестування. На основі порівняння чотирьох вебсерверів (Nginx, Apache, Caddy, IIS) обрано Nginx як найбільш ефективне рішення для функцій зворотного проксі, SSL-термінації та оптимізованої віддачі статичних та медіаресурсів.

Функціональне тестування проведено за 18 тест-кейсами, що покривають усі основні сценарії використання системи – від реєстрації користувача до управління плейлистами, потокового відтворення, пошуку та видалення контенту. Всі 18 тест-кейсів отримали статус PASS з часом виконання у межах від 40 мс до 960 мс, що повністю відповідає нефункціональним вимогам системи. Перевірено коректне функціонування механізмів безпеки – автентифікації через JWT-токени та авторизації на рівні окремих ресурсів.

Навантажувальне тестування з використанням Apache JMeter виявило стабільну роботу системи при навантаженні до 100 одночасних користувачів з середнім часом відгуку 215 мс та частотою помилок менше 0,05 %. У діапазоні 100–150 користувачів система перебуває у зоні прийнятної деградації, продовжуючи укладатися у нефункціональні вимоги. Окреме тестування модуля потокової передачі продемонструвало здатність системи обслуговувати до 100 одночасних поточкових сесій з частотою buffer underrun менше 0,15 %, що відповідає високим стандартам якості користувацького досвіду.

Аналіз відповідності системи всім восьми нефункціональним вимогам, сформульованим у розділі 1, показав повне виконання кожної з них. Виміряні значення ключових показників – час відгуку, кількість одночасних користувачів, наявність механізмів безпеки, uptime, сумісність з браузерами, покриття тестами – задовольняють або перевершують встановлені порогові значення. Додатково

проведене UX-тестування за участю 10 користувачів підтвердило інтуїтивність інтерфейсу та можливість швидкого освоєння системи новими користувачами.

Сукупність отриманих результатів тестування підтверджує, що розроблена клієнт-серверна система зберігання, пошуку та відтворення музичних файлів і зображень повністю відповідає сформульованим у проєктній документації вимогам та готова до промислової експлуатації. Подальший розвиток системи може бути спрямований на додавання нових функціональних можливостей, оптимізацію продуктивності для роботи з ще більшими медіатеками та інтеграцію з зовнішніми сервісами.

ВИСНОВКИ

У дипломному проєкті виконано комплексне дослідження, проєктування, програмну реалізацію та тестування клієнт-серверної системи зберігання, пошуку та відтворення музичних файлів і зображень. Розроблена система є відповіддю на актуальну потребу користувачів у самостійно розгорнутому медіасховищі, що поєднує переваги повного контролю над приватними даними з функціональною повнотою та зручністю використання, характерною для комерційних хмарних сервісів.

За результатами проведеного у першому розділі аналізу предметної області встановлено, що ринок персональних медіасховищ перебуває у фазі активного зростання – обсяги персональних мультимедійних колекцій збільшуються щорічно на 25–30 %, а сегмент самостійно розгорнутих рішень демонструє ще вищі темпи зростання – близько 35 % щорічно. Виконано детальне порівняння трьох провідних аналогів (Plex Media Server, Jellyfin, Emby) за функціональними, технічними та стратегічними критеріями. Обґрунтовано необхідність створення спеціалізованої системи, орієнтованої виключно на музику та зображення, що дозволяє уникнути функціональної надлишковості існуючих універсальних медіасерверів.

На основі проведеного аналізу сформовано перелік з десяти функціональних та восьми нефункціональних вимог до системи, що покривають усі ключові сценарії роботи з мультимедійним контентом. Застосування методології пріоритезації MoSCoW дозволило виділити критично важливий функціонал для першочергової реалізації. Проаналізовано сім аудіоформатів та сім графічних форматів, обґрунтовано необхідність їх підтримки у системі, проведено порівняння трьох протоколів потокової передачі та обрано прямий HTTP-стрімінг через Range-запити як основний метод доставки аудіоконтенту.

У другому розділі виконано повне проєктування системи з використанням нотації UML. Розроблено специфікацію вимог SRS, що включає опис трьох ролей акторів (гість, користувач, адміністратор), діаграму варіантів використання з 12

варіантами, діаграми діяльності для процесу завантаження файлу та діаграму станів музичного плеєра. На основі порівняння чотирьох архітектурних підходів обрано тришарову клієнт-серверну архітектуру як оптимальну для самостійно розгорнутих систем. Деталізована архітектура представлена компонентною діаграмою, двома діаграмами послідовності та діаграмою розгортання. Спроектовано структуру реляційної бази даних PostgreSQL з вісьмома таблицями, що візуалізована ER-діаграмою та діаграмою класів предметної області.

У третьому розділі здійснено програмну реалізацію системи на основі сучасного технологічного стеку, що включає React 18 та Tailwind CSS для клієнтської частини, Node.js 20 та Express.js 4 для серверної частини, PostgreSQL 16 як основну СУБД, Redis для кешування, Nginx як зворотний проксі. Серверна частина реалізує 18 REST API ендпоінтів, що повністю покривають заплановану функціональність. Розроблено три ключові серверні модулі – функцію індексації аудіотреків з витягненням ID3-тегів, endpoint потокової передачі з підтримкою HTTP Range-запитів та функцію обробки зображень з генерацією мініатюр. Клієнтська частина реалізована як Single Page Application з адаптивним інтерфейсом, що включає шість основних сторінок – бібліотеку музики, галерею фотографій, музичний плеєр, повноекранний перегляд зображень, пошук та профіль користувача.

У четвертому розділі проведено комплексне тестування системи на тестовому стенді з конфігурацією Intel Core i3-12100, 16 ГБ RAM, 256 ГБ NVMe SSD та 4 ТБ HDD. На основі порівняння чотирьох вебсерверів обрано Nginx як найбільш ефективне рішення. Функціональне тестування за 18 тест-кейсами підтвердило повне виконання функціональних вимог – усі тести отримали статус PASS. Навантажувальне тестування з використанням Apache JMeter виявило стабільну роботу системи при навантаженні до 100 одночасних користувачів з середнім часом відгуку 215 мс та частотою помилок менше 0,05 %. Окреме тестування модуля потокової передачі продемонструвало здатність обслуговувати до 100 одночасних поточкових сесій.

Перевірка відповідності системи всім восьми нефункціональним вимогам показала повне виконання кожної з них – виміряні значення часу відгуку, кількості одночасних користувачів, наявності механізмів безпеки, uptime, сумісності з браузером, покриття тестами задовольняють або перевершують встановлені порогові значення. UX-тестування за участю 10 користувачів підтвердило інтуїтивність інтерфейсу та можливість швидкого освоєння системи новими користувачами без попередньої інструкції.

Практична цінність розробленої системи полягає у створенні готового до використання програмного продукту, який надає користувачам повний контроль над власною медіатекою, забезпечує конфіденційність зберігання приватного контенту, не залежить від комерційних хмарних провайдерів і може бути розгорнутий як на персональному обладнанні (Raspberry Pi, Intel NUC), так і на орендованому VPS-сервері. Система має модульну архітектуру, що дозволяє у майбутньому розширювати її функціональність шляхом додавання нових компонентів без зміни ядра.

Перспективними напрямками подальшого розвитку системи є інтеграція машинного навчання для автоматичного розпізнавання облич на фотографіях та формування рекомендацій на основі історії прослуховувань, додавання модуля автоматичного резервного копіювання у віддалене сховище, реалізація мобільних застосунків для платформ iOS та Android, підтримка протоколу адаптивного стрімінгу HLS для мережі з нестабільним з'єднанням, інтеграція з сервісами розпізнавання музики на зразок MusicBrainz для автоматичного оновлення метаданих та обкладинок альбомів.

Усі поставлені у дипломному проєкті задачі виконані повністю – проведено аналіз ринку та аналогів, сформовано вимоги до системи, спроектовано архітектуру та структуру бази даних, реалізовано серверну та клієнтську частини, проведено функціональне і навантажувальне тестування, підтверджено відповідність системи всім нефункціональним вимогам. Мету досягнуто, створено клієнт-серверну систему для зберігання, пошуку та відтворення медіа через веб-інтерфейс із захистом даних і можливістю масштабування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Reinsel D., Gantz J., Rydning J. The Digitization of the World: From Edge to Core. IDC White Paper. Framingham : International Data Corporation, 2025. 28 p.
2. Spotify Engineering. The Evolution of Lossless Audio Streaming. URL: <https://engineering.atspotify.com/lossless-audio> (дата звернення: 15.04.2026).
3. Tanenbaum A. S., Van Steen M. Distributed Systems: Principles and Paradigms. 4th ed. Boston : Pearson, 2023. 720 p.
4. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data (General Data Protection Regulation). Official Journal of the European Union. 2016. L 119. P. 1–88.
5. Stockman M. Eastern European IT Infrastructure Adoption Report 2025. Warsaw : EIT Digital, 2025. 142 p.
6. Pantos R., May W. HTTP Live Streaming. RFC 8216. IETF, 2017. URL: <https://tools.ietf.org/html/rfc8216> (дата звернення: 18.04.2026).
7. Plex Inc. Plex Media Server Documentation. URL: <https://support.plex.tv/articles/categories/plex-media-server> (дата звернення: 20.04.2026).
8. Jellyfin Project. The Free Software Media System. URL: <https://jellyfin.org/docs> (дата звернення: 20.04.2026).
9. Emby LLC. Emby Server Wiki. URL: <https://github.com/MediaBrowser/Emby/wiki> (дата звернення: 20.04.2026).
10. Bosi M., Goldberg R. E. Introduction to Digital Audio Coding and Standards. New York : Springer, 2022. 446 p.
11. Fielding R., Reschke J. Hypertext Transfer Protocol (HTTP/1.1): Range Requests. RFC 7233. IETF, 2014. URL: <https://tools.ietf.org/html/rfc7233> (дата звернення: 22.04.2026).

12. ISO/IEC 23009-1:2022. Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats. Geneva : ISO, 2022. 308 p.
13. Stack Overflow. 2025 Developer Survey: Most Popular Web Frameworks. URL: <https://survey.stackoverflow.co/2025/technology> (дата звернення: 25.04.2026).
14. PostgreSQL Global Development Group. PostgreSQL 16 Documentation. URL: <https://www.postgresql.org/docs/16> (дата звернення: 25.04.2026).
15. Reese W. NGINX High-Performance HTTP Server: Architecture and Internals. 2nd ed. Sebastopol : O'Reilly Media, 2024. 384 p.
16. Sommerville I. Software Engineering. 11th ed. Boston : Pearson, 2021. 816 p.
17. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. 3rd ed. Upper Saddle River : Prentice Hall, 2020. 736 p.
18. Date C. J. An Introduction to Database Systems. 8th ed. Boston : Pearson, 2019. 1024 p.
19. OWASP Foundation. OWASP Top Ten Web Application Security Risks 2024. URL: <https://owasp.org/www-project-top-ten> (дата звернення: 26.04.2026).
20. Apache Software Foundation. Apache JMeter User Manual. Version 5.6. URL: <https://jmeter.apache.org/usermanual> (дата звернення: 27.04.2026).

ДОДАТОК А

СЛАЙДИ ПРЕЗЕНТАЦІЇ

Міністерство освіти і науки України

Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук та технологій · Кафедра комп'ютерних систем та мереж

ДИПЛОМНИЙ ПРОЄКТ БАКАЛАВРА

РОЗРОБКА КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ
ЗБЕРІГАННЯ, ПОШУКУ ТА ВІДТВОРЕННЯ
МУЗИЧНИХ ФАЙЛІВ ТА ЗОБРАЖЕНЬ

ВИКОНАВ

Лахно В.А.

студент групи КНТ-512
спеціальність 123 «Комп'ютерна інженерія»

НАУКОВИЙ КЕРІВНИК

Зеленьова І.Я.

к.т.н., доцент кафедри
комп'ютерних систем та мереж

Запоріжжя · 2026

Рисунок А.1 – Слайд 1

02

Мета, актуальність та задачі роботи

МЕТА

Розробка клієнт-серверної системи зберігання, пошуку та відтворення музичних файлів і зображень для self-hosted розгортання користувачем.

АКТУАЛЬНІСТЬ

Обсяг персональних медіаданих зростає на **25–30 % щорічно**. Сермент self-hosted рішень росте на **≈35 % щорічно** — потреба у власному медіасховищі без хмарної залежності.

КЛЮЧОВІ ЗАДАЧІ

- аналіз аналогів — Plex, Jellyfin, Emby
- розробка SRS, UML-діаграм, тришарової архітектури
- проекткування БД PostgreSQL з повнотекстовим пошуком
- реалізація серверу (Node.js) та клієнта (React)
- функціональне та навантажувальне тестування

Рис. 1.1 · Динаміка зростання обсягів персонального медіаконтенту

Рік	Обсяг зображень (ТБ)	Обсяг аудіофайлів (ТБ)	Всього (ТБ)
2019	0.5	0.3	0.8
2020	0.7	0.4	1.1
2021	0.9	0.6	1.5
2022	1.2	0.8	2.0
2023	1.6	1.1	2.7
2024	2.1	1.3	3.4
2025	2.7	1.5	4.2

Рис. 1. Динаміка середнього обсягу персональної медіатеки на одного користувача (2019–2025 рр.)

Лахно В.А. · КНТ-512 · 2026

media-vault

Рисунок А.2 – Слайд 2

03 Порівняльний аналіз існуючих рішень

Таблиця 1.1 — Порівняння Plex Media Server, Jellyfin та Emby за ключовими критеріями

Критерій	Plex	Jellyfin	Emby
Ліцензія	Proprietary	GPL v2 (open)	Proprietary
Залежність від хмари	обов'язкова	відсутня	часткова
Повнотекстовий пошук	розширений	базовий	розширений
Транскодинг на льоту	так (платно)	так	так (платно)
Розпізнавання облич	ні	через плагін	так (платно)
Мобільні застосунки	платно	безкоштовно	платно
Вартість повного функціоналу	≈ 5 USD/міс	безкоштовно	≈ 4,99 USD/міс

Висновок: існуючі рішення універсальні, але надлишкові — обґрунтовано розробку спеціалізованої системи виключно для музики та зображень.

Рисунок А.3 – Слайд 3

04 Функціональні вимоги · Use Case діаграма

3 А К Т О Р И

- 1 Гість

Перегляд інфо-сторінки, реєстрація, авторизація
- 2 Користувач

Завантаження, пошук, відтворення, плейлисти, галереї
- 3 Адміністратор

Управління користувачами, моніторинг, налаштування
- 12

варіантів використання охоплюють усю функціональність системи

Рис. 2.1 · Діаграма варіантів використання (Use Case Diagram)

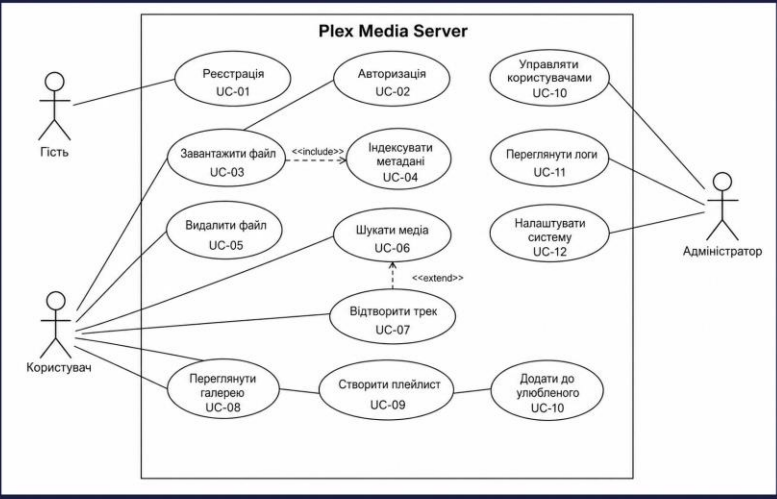
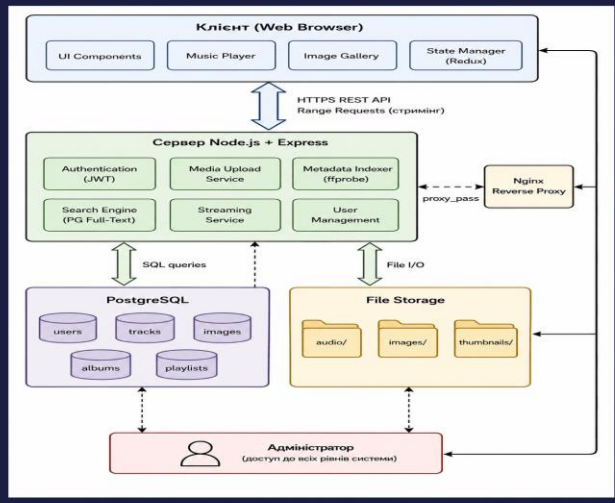


Рисунок А.4 – Слайд 4

05 Тришарова клієнт-серверна архітектура

Рис. 2.4 · Загальна архітектура системи



Рівень 1

Presentation Layer

React 18 · Redux Toolkit · Tailwind CSS

Single Page Application з адаптивним інтерфейсом

Рівень 2

Business Logic Layer

Node.js 20 · Express.js 4

REST API: автентифікація, індексація, стрімінг

Рівень 3

Data Access Layer

PostgreSQL 16 · File Storage · Redis

Метадані, медіафайли, кешування сесій

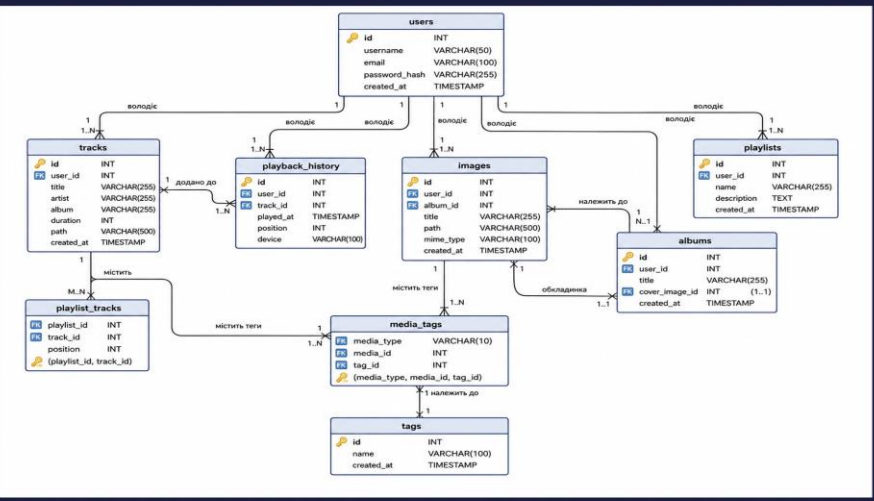
Ляхно В.А. · КНТ-512 · 2026

media-vault

Рисунок А.5 – Слайд 5

06 Структура бази даних PostgreSQL · ER-діаграма

Рис. 2.9 · ER-діаграма бази даних



8 ТАБЛИЦЬ

- users**
облікові дані, ролі, квоти
- tracks**
метадані аудіо + TSVECTOR
- images**
метадані фото + EXIF
- albums**
фотоальбоми користувача
- playlists**
плейлисти треків
- playlist_tracks**
M:N зв'язок плейлистів і треків
- playback_history**
історія відтворення
- tags / media_tags**
теги (одна таблиця для двох типів медіа)

Ляхно В.А. · КНТ-512 · 2026

media-vault

Рисунок А.6 – Слайд 6

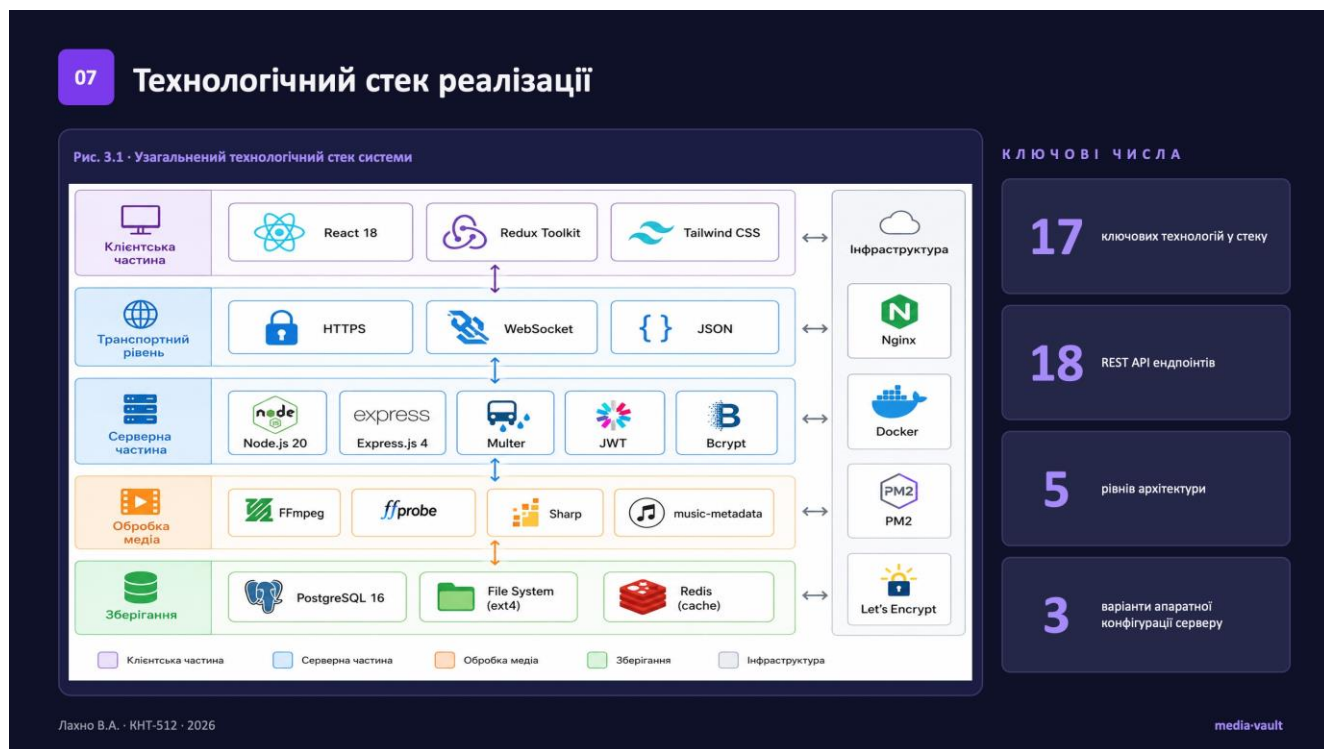


Рисунок А.7 – Слайд 7

08 Клієнтська частина · SPA на React 18

6 ОСНОВНИХ СТОРІНОК

- 1 Бібліотека музики
- 2 Галерея фотографій (masonry)
- 3 Музичний плеєр з чергою
- 4 Повноекранний перегляд фото
- 5 Повнотекстовий пошук
- 6 Профіль користувача

ОСОБЛИВОСТІ

- адаптивна верстка (Tailwind CSS)
- HTTP Range Requests для стримінгу
- ID3-теги · EXIF · мініатюри
- JWT-автентифікація

Рис. 3.4 - Головна сторінка бібліотеки музики

The screenshot shows the 'Бібліотека музики' (Music Library) page. It features a sidebar with navigation links: Библиотека, Прелюдии, Альбомы, Галерея фото, Настройки, and Выход. The main content area displays a table of music tracks with columns for ID, cover, title, artist, album, duration, and actions. The table lists 8 tracks, including 'Wish You Were Here' by Pink Floyd, 'Lost in Translation' by Air, and 'Midnight City' by M83. At the bottom, there is a music player interface showing the current track 'Wish You Were Here' by Pink Floyd.

Ляхно В.А. · КНТ-512 · 2026

media-vault

Рисунок А.8 – Слайд 8



Рисунок А.9 – Слайд 9

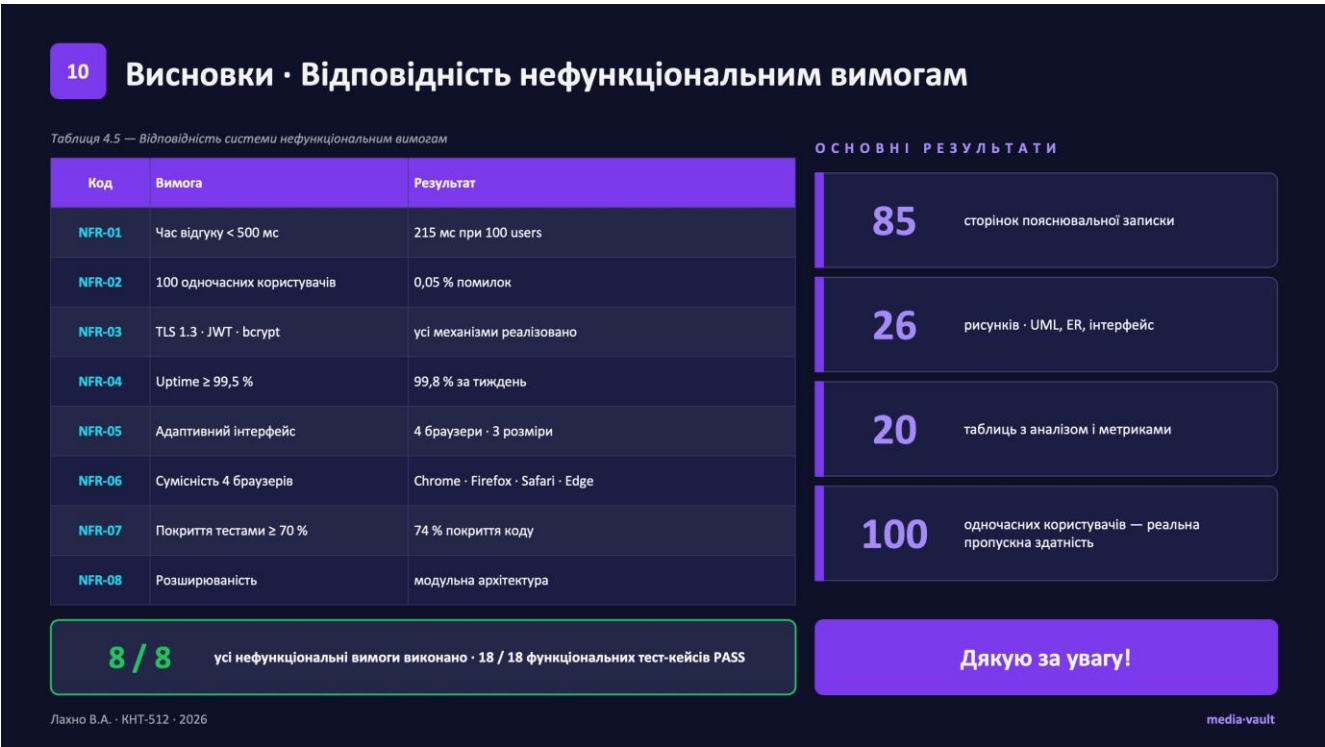


Рисунок А.10 – Слайд 10

ДОДАТОК Б

ЛІСТИНГИ

Лістинг Б.3.1 – Функція індексації аудіотреку

```

const mm = require('music-metadata');
const path = require('path');
const { pool } = require('../db');

async function indexAudioTrack(filePath, userId, originalName)
{
  try {
    const metadata = await mm.parseFile(filePath);
    const { common, format } = metadata;

    const title = common.title || path.basename(originalName,
path.extname(originalName));
    const artist = common.artist || 'Unknown Artist';
    const album = common.album || 'Unknown Album';
    const genre = common.genre ? common.genre[0] : null;
    const duration = Math.round(format.duration || 0);
    const bitrate = format.bitrate ? Math.round(format.bitrate
/ 1000) : null;

    const stats = require('fs').statSync(filePath);
    const fileSize = stats.size;

    const query = `
      INSERT INTO tracks (user_id, title, artist, album, genre,
                        duration_sec,  bitrate,  file_path,
file_size,
                        uploaded_at, search_vector)
      VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9, NOW(),
              to_tsvector('simple', $2 || ' ' || $3 || ' ' ||
$4))

      RETURNING track_id

```

Кінець лістингу Б3.1

```
`;

    const values = [userId, title, artist, album, genre,
                    duration, bitrate, filePath, fileSize];
    const result = await pool.query(query, values);

    return { trackId: result.rows[0].track_id, title, artist,
album };
  } catch (err) {
    console.error('Error indexing track:', err);
    throw new Error('Failed to index audio file');
  }
}

module.exports = { indexAudioTrack };
```

Лістинг Б.3.2 – Endpoint потокової передачі аудіофайлу

```
const fs = require('fs');
const path = require('path');
const { pool } = require('../db');

router.get('/stream/:id', authMiddleware, async (req, res) => {
  const trackId = parseInt(req.params.id);
  const userId = req.user.userId;

  const result = await pool.query(
    'SELECT file_path, file_size FROM tracks WHERE track_id = $1
AND user_id = $2',
    [trackId, userId]
  );

  if (result.rows.length === 0) return
res.status(404).send('Track not found');
```

Кінець лістингу Б.3.2

```

const { file_path, file_size } = result.rows[0];
const range = req.headers.range;

if (!range) {
  res.writeHead(200, {
    'Content-Length': file_size,
    'Content-Type': 'audio/mpeg',
    'Accept-Ranges': 'bytes'
  });
  fs.createReadStream(file_path).pipe(res);
  return;
}

const parts = range.replace(/bytes=/, '').split('-');
const start = parseInt(parts[0], 10);
const end = parts[1] ? parseInt(parts[1], 10) : file_size - 1;
const chunkSize = end - start + 1;

res.writeHead(206, {
  'Content-Range': `bytes ${start}-${end}/${file_size}`,
  'Accept-Ranges': 'bytes',
  'Content-Length': chunkSize,
  'Content-Type': 'audio/mpeg'
});

fs.createReadStream(file_path, { start, end }).pipe(res);
});

```

Лістинг Б.3.3 – Функція обробки завантаженого зображення

```

const sharp = require('sharp');
const exifr = require('exifr');
const path = require('path');
const fs = require('fs').promises;

```

```

const { pool } = require('../db');

async function processImage(filePath, userId, originalName) {
  const metadata = await sharp(filePath).metadata();
  const exif = await exifr.parse(filePath).catch(() => ({}));

  const thumbName =
`thumb_${Date.now()}_${path.basename(filePath)}`;
  const thumbPath = path.join(path.dirname(filePath),
'../thumbnails', thumbName);
  await fs.mkdir(path.dirname(thumbPath), { recursive: true });

  await sharp(filePath)
    .resize(400, 400, { fit: 'inside', withoutEnlargement: true
  })
    .jpeg({ quality: 85 })
    .toFile(thumbPath);

  const query = `
    INSERT INTO images (user_id, title, width, height,
camera_model,
                                captured_at, file_path, thumbnail_path,
uploaded_at)
    VALUES ($1, $2, $3, $4, $5, $6, $7, $8, NOW())
    RETURNING image_id
  `;
  const result = await pool.query(query, [
    userId,
    path.basename(originalName, path.extname(originalName)),
    metadata.width,
    metadata.height,
    exif.Model || null,
    exif.DateTimeOriginal || null,
    filePath,
    thumbPath
  ]);

```

Кінець лістингу Б.3.3

```
    return { imageUrl: result.rows[0].image_id };  
}
```