НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Завідувач кафедри
_____Наталія АУШЕВА
«___»_____2022 р.

Дипломна робота

на здобуття ступеня бакалавра

спеціальності 122 «Комп'ютерні науки»

освітня програма «Комп'ютерний моніторинг та геометричне моделювання процесів і систем»

на тему: « Веб-система з формування метаданих аудіофайлів»

Виконав:
студент IV курсу, групи ТР-81
Бережний Владислав Олександрович

_____

Керівник:
доцент
Шалденко Олексій Вікторович

_____

Рецензент:

_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2022 рік

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший

Спеціальність 122 «Комп'ютерні науки»

освітня програма «Комп'ютерний моніторинг та геометричне моделювання процесів і систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Наталія АУШЕВА
(підпис)
"___"_____2022р.

ЗАВДАННЯ

на дипломну роботу студента

_____Бережному Владиславу Олександровичу _____
(прізвище, ім'я, по батькові)

1. Тема роботи Веб-система з формування метаданих аудіофайлів _____

керівник роботи____Шалденко Олексій Вікторович, доцент_____
(прізвище, ім'я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від "25" травня 2022р. № _____

2. Строк подання студентом роботи __ 10___ червня____ 202 2__ р._____

3. Вихідні дані до роботи мова програмування JavaScript , бібліотека React.js, платформа Node.js ,фреймворк Express.js, СКБД PostgreSql

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати існуючі додатки для формування метаданих для аудіофайлів, обґрунтувати обрані засоби та методи для програмної реалізації, спроектувати структуру бази даних, спроектувати архітектуру програми, розробити програмний продукт, реалізувати інтерфейс користувача системи.

5.Орієнтовний перелік ілюстративного матеріалу: діаграма прецедентів, концептуальна модель бази даних, інтерфейс користувача.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”10” вересня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
	Затвердження теми роботи	25.05.2022	
	Вивчення та аналіз задачі	05.04.22 – 20.04.22	
	Розробка архітектури та загальної структури системи	20.04.22 – 27.04.22	
	Розробка структур окремих підсистем	02.05.22 – 09.05.22	
	Програмна реалізація системи	09.05.22 – 24.05.22	
	Оформлення пояснювальної записки	23.05.22 – 27.05.22	
	Захист програмного продукту	25.05.22 – 27.05.22	
	Передзахист	06.06.22 – 09.06.22	
	Захист	20.06.22 – 30.06.22	

Студент _____ Бережний В.О.
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Шалденко О.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Метою дипломної роботи є розробка веб-системи з формування метаданих аудіофайлів. Система дозволяє формувати та редагувати метадані аудіофайлів. Додаток дає можливість користувачу редагувати такі метадані як: назва, виконавець, альбом, жанр та обирати обкладинку альбом.

Систему було розроблено за допомогою бібліотеки React.js, платформи Node.js, фреймворку Express.js, мови програмування JavaScript, розширення для Java Script – JSX та системи управління базою даних PostgreSQL.

Записка містить: 54 сторінок, 24 рисунків, 1 додаток і 22 посилань.

Ключові слова: JavaScript, метадані, веб-система, інформаційні технології

ABSTRACT

The purpose of the thesis is to develop a web system for generating metadata for audio files. The system allows you to create and edit metadata of audio files. The application allows the user to edit metadata such as title, artist, album, genre and select album art.

The system was developed using the React.js library, the Node.js platform, the Express.js framework, the JavaScript programming language, the Java Script extension - JSX and the PostgreSQL database management system.

The note contains: 54 pages, 24 figures,1 appendix and 22 links.

Keywords: JavaScript, metadata, web system, information technology.

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

БД — база даних.

ПЗ – програмне забезпечення.

ПП- програмний продукт

СУБД — система управління базами даних.

JSON (JavaScript Object Notation) — текстовий структурований формат даних.

API (Application Programming Interface) — стандартизований інтерфейс для взаємодії клієнтів з сервером за стандартним протоколом.

JWT (JSON Web Token) - це відкритий стандарт для створення токенів доступу на основі JSON.

CSS (Cascading Style Sheets) — це спеціальна мова стилю сторінок.

JS (Java Script) — мова програмування.

PostgreSQL — система керування базами даних

Email (email) — електронна пошта.

Клієнт — веб-браузер.

ORM (Object-Relational Mapping) – об'єктно-реляційне представлення.

JSX (Javascript XML) – це розширення синтаксису JavaScript.

HTML (HyperText Markup Language) — мова гіпертекстової розмітки.

UI (User Interface) – користувацький інтерфейс.

XML (Extensible Markup Language) – розширювана мова розмітки.

XML (Extensible Markup Language) – розширювана мова розмітки.

MVC (Model-view-controller) - Модель–вигляд–контролер - це архітектурний

REST (Representational state transfer) - архітектурний стиль взаємодії компонентів розподіленої програми у мережі.

ECMAScript — стандарт мови програмування, затверджений міжнародною організацією ECMA згідно зі специфікацією ECMA-262.

DOM (Document Object Model) - це об'єктна модель документа, який браузер створює в пам'яті комп'ютер на основі HTML-коду, отриманого від сервера.

Drag-and-drop (у перекладі з англ. - "тягни-і-кидай", "бери-і-кинь") - спосіб оперування елементами інтерфейсу в інтерфейсах користувача.

ЗМІСТ

ВСТУП

В сучасному світі дуже гостро стоїть питання систематизації та подальшої уніфікації інформації. Одним з самих розповсюджених типів зберігання інформації є аудіофайли, тому питання формування метаданих набуває актуальності. Виникає гостра необхідність у правильному оформлені файлів з уніфікованою структурою метаданих, наприклад, аудіо метадані дозволяють спростити пошук цифрової музики. Користувач має змогу знайти той чи інший аудіофайл використовуючи будь-який стрімінговий сервіс, наприклад, Spotify або AppleMusic, завдяки цим метаданим.

Самі собою метадані аудіофайлів — це набір інформації, яка відноситься до файлу пісні, наприклад, ім’я виконавця, продюсера, автора, назва пісні, дата випуску, жанр або тривалість треку. Мова метаданих забезпечує рівень стандартизації, який сприяє кращій сумісності та інтеграції між різними програмами та інформаційними системами.

Метадані спрощують роботу з аудіофайлами, адже вони дозволяють виконати сортування файлів за різними параметрами: автором, назвою, жанром. Більшість алгоритмів для роботи з аудиофайлами не використовує аудіодоріжку, бо вся необхідна інформація вже зберігається у середині аудіофайлу у так званих тегах (поля метаданих по принципу ключ-значення). Тому у випадку коли одна із цих відомостей відсутня чи з помилкою, це може негативно позначитися на компенсації її авторам, через те що у файла зникає можливість на ідентифікацію.

Для вирішення питання стандартизації та уніфікації метаданих в аудіофайлах було виконано цю дипломну роботу, яка представляє собою веб-сервіс для формування метаданих аудіофайлі. Розроблений програмний продукт дозволяє виконати усі необхідні дії для взаємодії з аудіофайлами та застосувати алгоритми, що дозволяють виконати задачу уніфікації великої кількості метаданих аудіофайлів одночасно.

1 ЗАДАЧА ВЕБ-СЕРВІСУ З ФОРМУВАННЯ МЕТАДАНИХ АУДІОФАЙЛІВ

Метадані – це дані, які надають інформацію про інші дані. Метадані узагальнюють основну інформацію про дані, що спрощує пошук і роботу з конкретними екземплярами даних[17]. Метадані можна створювати вручну, щоб бути точними, або автоматично та містити більш основну інформацію.

З точки зору аудіофайлів, метадані — це просто інформація про назву альбому, ім'я виконавця, назву пісні, інформацію про авторські права, ISRC.
Деякі метадані можуть відігравати важливу роль у тому, щоб автор аудіофайлу отримував гроші, коли його пісні використовуються.

1.1 Аналіз різних веб-систем з формування метаданих аудіофайлів

Особливістю задачі створення веб-системи формування аудіо файлів є зручність та функціональність продукту. Розглянемо існуючі програмні продукти, що використовуються для формування метаданих користувача користувача.

1.1.1 Mp3Tag

Використовувати Mp3tag дуже просто, можна просто завантажити інсталяційний файл, якщо необхідно відредагувати багато файлів[16]. Також можна обрати тільки каталог і файли, які з'являються у вікні програми, є можливість обрати окремі файли або вибрати все для редагування метаданих, що зображено на рисунку 1.1.

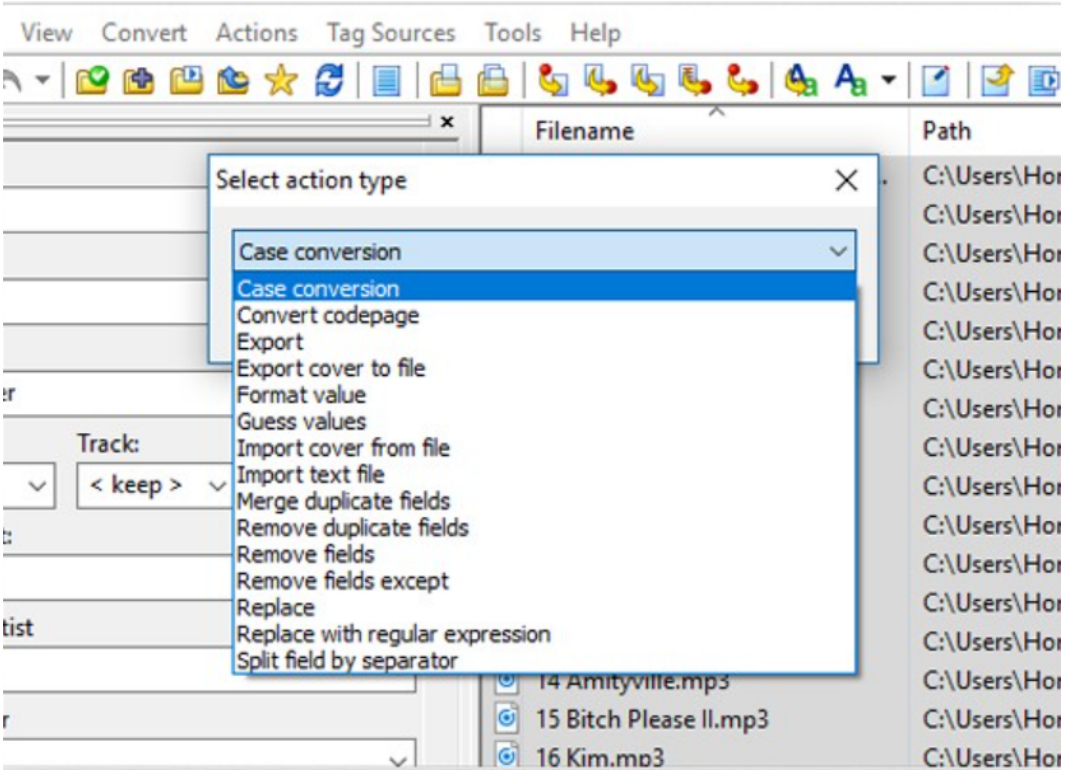

Рисунок 1.1 — Інтерфейс Mp3Tag

Особливість цієї програми - опція дій. Виконання завдань, що повторюються, може займати дуже багато часу, а також бути досить стомлюючим процесом, тож цей додаток дає можливість створювати дії, які автоматично виконують цей процес за користувача[16]. Наприклад, у користувача є кілька альбомів, і він хоче видалити всі поля з аудіофайлів, тож для того, щоб це зробити він повинен просто натиснути «Дії» і вибрати «Видалити поля, що повторюються».

Якщо поля недоречні, наприклад, якщо в іменах файлів відображаються деякі значки, а теги мають справжні імена, можна просто перемикати ці поля натисканням кнопки в параметрі перетворення в рядку меню, як на рисунку 1.2.

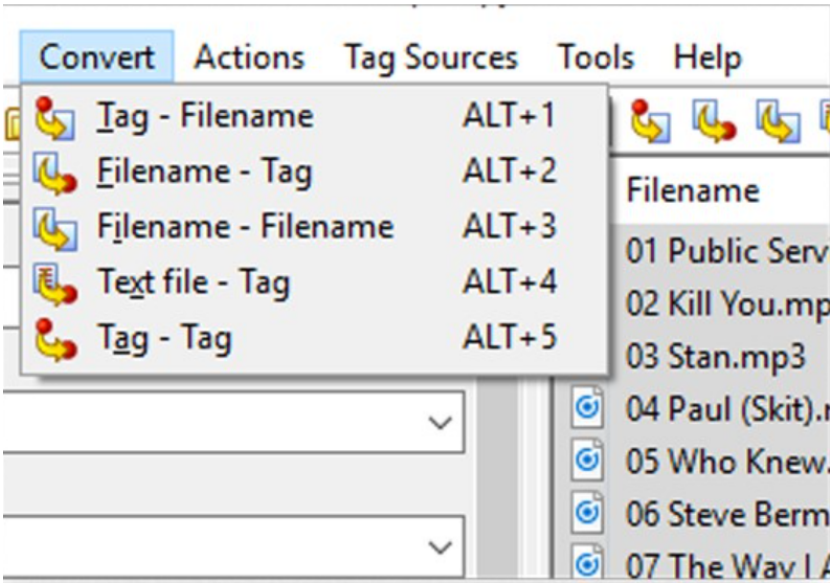

Рисунок 1.2 — Інтерфейс Mp3Tag 2

У програмі налаштовані зручні гарячі клавіші для швидкого виконання різних команд.

1.1.2 MusicBrainz Picard

Кросплатформенний менеджер тегів (доступний на: Linux, Mac OS, Windows) з підтримкою більшості відомих аудіо форматів - MP3, Ogg Vorbis, FLAC та інші.

В MusicBrainz Picard доступний пошук аудіозаписів по AcoustID (Acoustic Fingerprint) та Disc ID. Що важливо, для Picard створено значну кількість плагінів, що розширюють можливості редагування тегів. Додатково можна налаштувати автоматичне скачування обкладинок (раніше ця можливість була реалізована у вигляді плагіна). Таким чином, під час операції Lookup програма з'єднуватиметься з потрібним сервером і завантажуватиме обкладинку, додаючи її в метадані. Інтерфейс MusicBrainz зображено на рисунку 1.3.

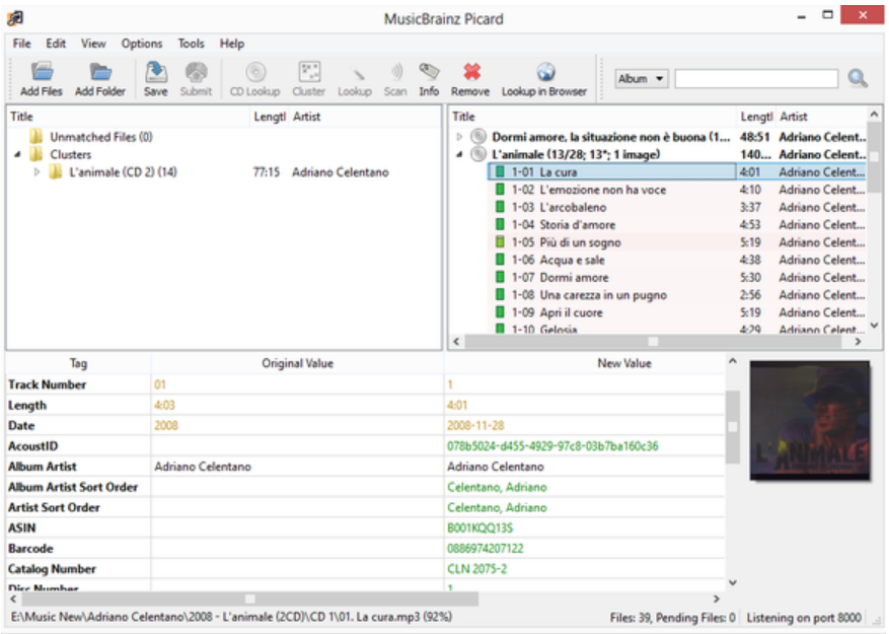

Рисунок 1.3 — Інтерфейс MusicBrainz Picard

Отже, у сервіса є цікаві засоби автоматизації, але на жаль повністю відсутні стандартні інструменти для пакетного перейменування.

1.1.3 Metatogger

Metatogger – безкоштовний редактор тегів, працює з Ogg Vorbis, FLAC, MP3, Musepack, Windows Media, WavPack, Monkey's Audio та іншими форматами[18]. Головною особливістю програми можна назвати зручні інтеграційні можливості – серед них заявлена підтримка локальної бази даних із MusicBrainz, функція acoustic footprint, пошук текстів композицій та інші.

Головне вікно досить просторе, з нього зручно керувати широким списком композицій, альбомів, виконавців. Головне вікно зображено на рисунку 1.4.

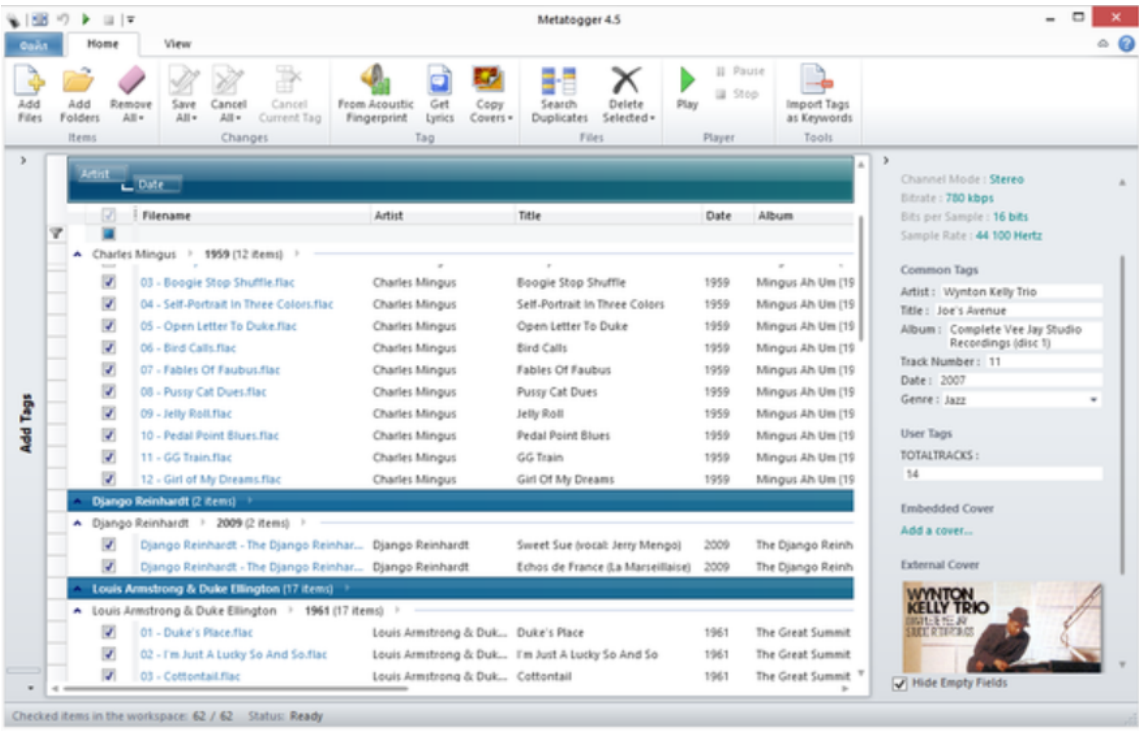

Рисунок 1.4 — Інтерфейс Metatogger

Широка панель інструментів дозволяє швидко орієнтуватися серед опцій без напруженого пошуку команд. Бічні панелі по обидва боки від центральної колонки ховаються. За допомогою меню View легко додати до робочого списку потрібну колонку або видалити зайву. При більш уважному розгляді відкриваються інші можливості колонок: при перетягуванні на панель поверх заголовків можна встановити кілька рівнів сортування списку.

Крім стандартних можливостей редагування тегів, є онлайн-інструменти, які покликані автоматизувати заповнення метаданих. Так, за допомогою функції Acoustic Fingerprint можна додати інформацію про теги, використовуючи цифровий відбиток аудіофайлу. Metatogger відправляє "знімок" аудіозапису на сервіс і звіряє його з наявною базою. Локальна база даних налічує понад 500 тис. альбомів та 700 тис. обкладинок[18].

Проаналізувавши даний сервіс можна впевнено сказати, що він відмінно інтегрований з іншими онлайн-сервісами, має підтримку локальної бази даних та

зручний інтерфейс. Але відсутність експорту та доволі важкі для освоєння скрипти не залишають можливості стати універсальним рішенням для користувача будь якого рівня.

1.1.4 Tagscanner

Tagscanner - програма для організації музичної колекції та редагування тегів. Підтримує формати аудіо MP3, OGG, FLAC та інші. Також розпізнає та читає відомі види метаданих, отримує теги напряму с аудіофайлів але дуже повільно. Широкі можливості імпорту тегів із музичних сервісів та експорту у різні формати, що можна побачити на рисунку 1.5, де зображене головне меню програми для організації музичної колекції. Можна створювати свої особисті колекції, змінювати теги власних аудіофайлів для автоматичного занесення до особистих колекцій.

Широка панель інструментів дозволяє швидко орієнтуватися серед опцій без напруженого пошуку команд. Інтерфейс хоч і інтуїтивно зрозумілий але зручним його назвати важко.

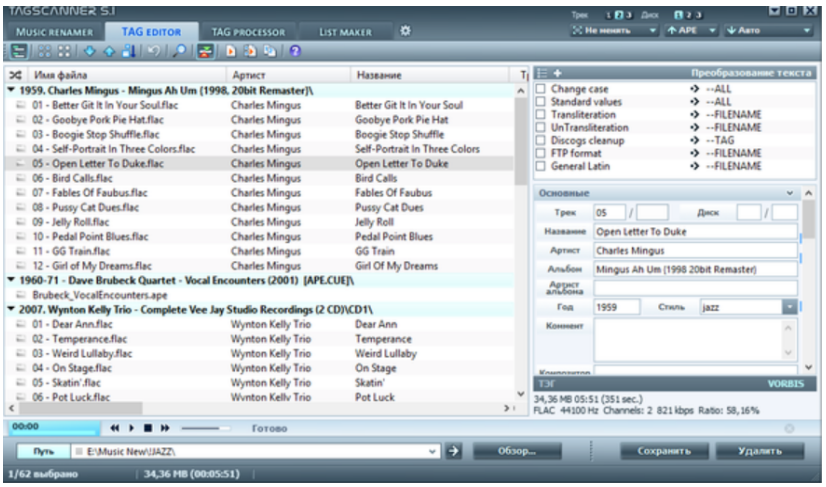

Рисунок 1.5 — Інтерфейс Tagscanner

Відповідно до назви вкладок, можна виділити такі можливості програми:

1. Перейменування файлів (Music Renamer).
2. Редагування тегів (TAG Editor).
3. Пакетна робота з тегами (TAG Processor).
4. Створення звітів та плейлистів (List Maker).

Отже, программа має зручний інтерфейс, готові шаблони, гнучкі можливості щодо перейменування тегів та вбудований аудіопрогравач.

1.2 Задача формування метаданих

Задача формування метаданих аудіофайлів полягає в створенні веб-системи зі зручним інтерфейсом, що буде націлений допомогти користувачам зручно редагувати та формувати метадані аудіофайлу.

Тому веб-сервіс має забезпечувати наступні функції:

- Авторизація.
- Завантаження аудіофайлу.
- Перегляд метаданих аудіофайлу.
- Редагування метаданих аудіофайлу.
- Скачування аудіофайлу.
- Створення каталогу збережених аудіофайлів.

Користувачі будуть створювати особисті акаунти та зберігати свої файли у системі, тому також необхідно потурбуватися про безпеку даних.

2 ЗАСОБИ РОЗРОБКИ

Проаналізувавши задачу програмного продукту, було вирішено обрати REST архітектуру. Таку архітектуру було обрано одразу з декількох причин, а саме — наша система передбачає взаємодію на рівні клієнт-серверної моделі, а основні обчислення будуть відбуватися на сервері[5].

Через те що системі будуть оброблятися файли, кількість обчислень може ставати доволі вагомою і на деяких слабких клієнтах можуть з'являтися труднощі з продуктивністю.

Також важливим є за допомогою яких саме засобів розробки буде реалізовано програмний продукт, бо саме засоби розробки та їх вдалий вибір, може значно полегшити як етап програмування так і впливати на продуктивність роботи веб-системи. Неправильний вибір засобів розробки може негативно вплинути на подальший розвиток продукту вчасності його масштабування — додавання нових модулів та переробка вже існуючих.

2.1 RESTful API

Для побудови веб-системи обрано архітектуру REST, яка необхідна для реалізації архітектурного стилю програмного інтерфейсу додатка під назвою RESTful API.

Схему RESTful API наведено на рисунку 2.1. Клієнт використовує HTTP запити для доступу і користування даними з бази даних. Самі HTTP запити розділяють на GET, POST, PUT та DELETE запити, які в свою чергу відповідають за отримання, внесення, оновлення та видалення даних на сервері[5]. А саму відповідь з серверу REST API повертає до клієнта у форматі JSON.

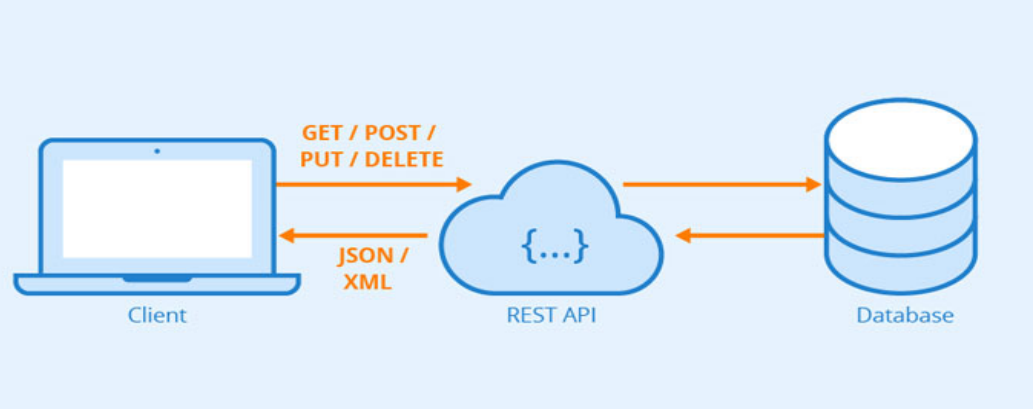

Рисунок 2.1 — Схема RESTful API застосунку

API для сайтів це модель що дозволяє двум програмним застосунком обмінюватися даними за допомогою HTTP запитів повністю розділивши клієнтську і серверну логіку, як це зображенно на рисунку 2.2.

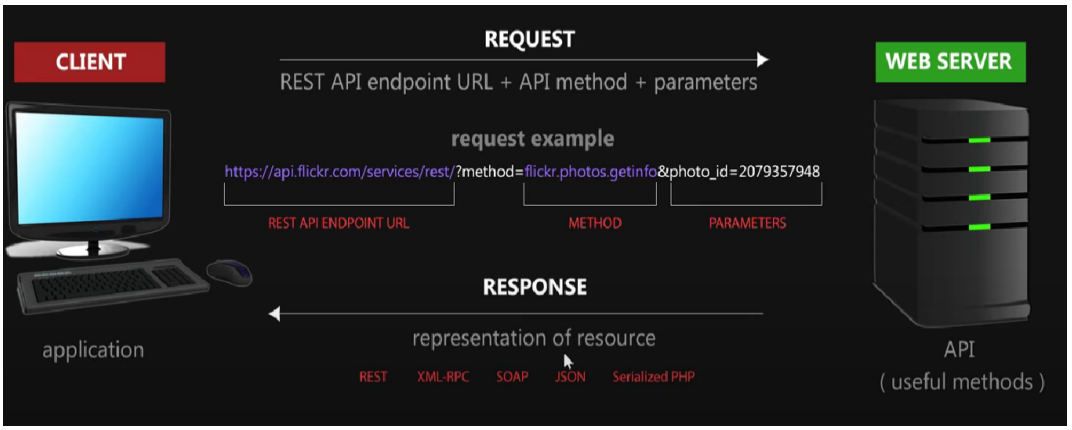

Рисунок 2.2 — Обмін даними між клієнтом і сервером за допомогою HTTP запитів

Основним компонентом системи є сервер, в ньому і буде зосереджена вся логіка системи. Сервер виконує роботу по обробці HTTP запитів, за допомогою яких користувач може отримувати необхідні сторінки сайту. Також на сервер

завантажуються аудіо файли користувача, де і відбувається їх обробка, зберігання та запис до бази даних необхідної інформації.

2.2 Node.js — серверне середовище з відкритим вихідним кодом

Ми реалізуємо RESTful API додаток, який дуже зручно створювати за допомогою саме платформи Node.js[1], на рисунку 2.3 можно побачити для чого можна застосовувати Node.js.

Рисунок. 2.3 — Перелік застосунків Node.js

Більше того Node.js має дуже важливу та унікальну перевагу, адже використання Node.js дає змогу писати як кліенську чатину так і код на стороні сервера на однаковому язиці програмування JavaScript[10]. Це значно спрощує розробку, через відсутність потреби у вивченні іншої мови програмування та використання додаткових IDE.

Сама по собі платформа Node.js – це серверна платформа з відкритим кодом для праці с JavaScript через двигун V8[9]. Схема роботи високопродуктивного двигуна V8 відображена на рисунку 2.4.

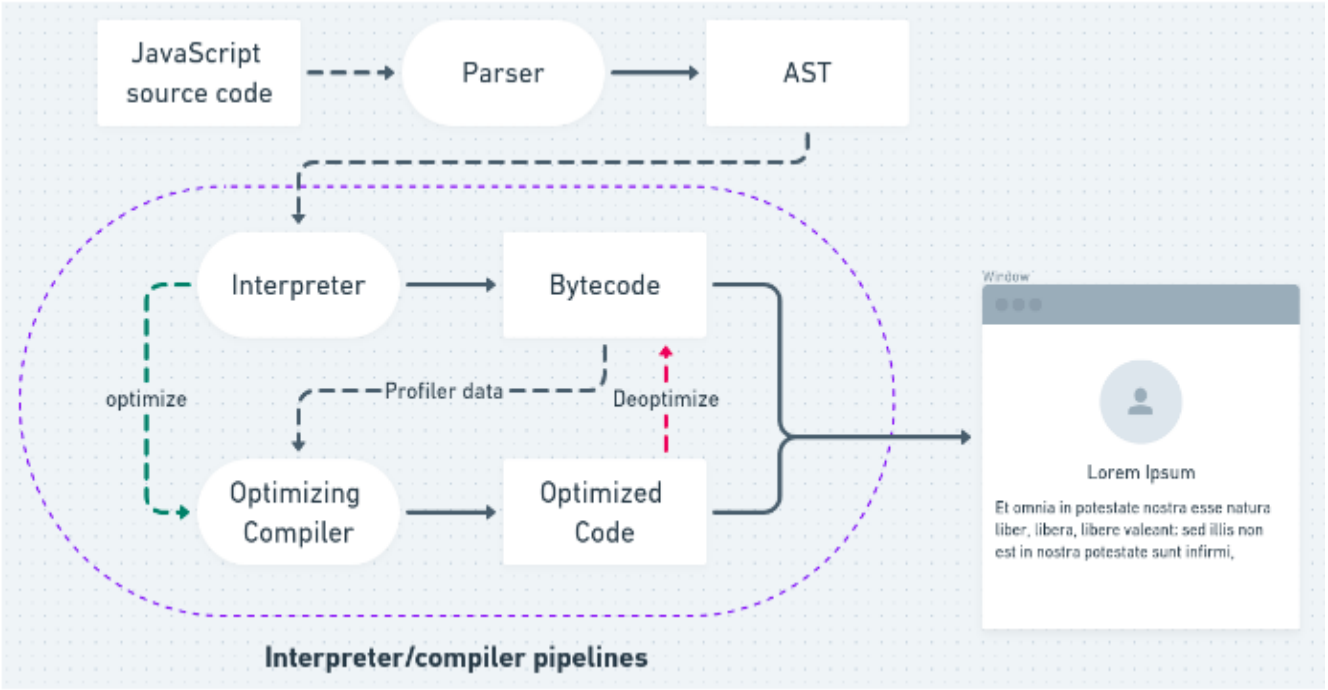

Рисунок. 2.4 — Двигун V8

Node.js має менеджер пакетів npm, що надає великий вибір пакетів, створених спільнотою розробників[1]. Деякі важливі для проекту бібліотеки на фреймворки, що представленні у інструменті Node.js, менеджері пакетів npm, ми будемо використовувати у нашій веб-системі.

2.3 Модуль music-metadata-browser парсер для роботи с метаданими аудіофайлів

Модуль music-metadata-browser – це один із пакетів, менеджера пакетів npm платформи Node.js, що використовується як парсер для метаданих аудіофайлів. Модуль music-metadata-browser розроблений спеціально для браузерних додатків, та підтримує майже всі існуючі аудіо формати і їх теги.

2.4 PostgreSQL — об'єктно-реляційна система управління базами даних.

PostgreSQL[2] — це одна з найсучасніших систем баз даних, яка має відкритий вихідний код. PostgreSQL підтримує запити SQL і JSON.

PostgreSQL — це надійна база даних, яка підтримується більш ніж 20 роками безкоштовного розвитку спільноти. PostgreSQL є основною базою даних для більшості веб-додатків, а також мобільного та аналітичного програмного забезпечення.

На початку проект називався POSTGRES, посилаючись на стару базу даних Ingres, яка також була розроблена в Берклі. Метою проекту POSTGRES було додати мінімальні функції, необхідні для підтримки кількох типів даних. У 1996 році проєкт POSTGRES був перейменований на PostgreSQL, щоб чітко проілюструвати його підтримку SQL[14]. Сьогодні PostgreSQL зазвичай скорочується як Postgres.

Спочатку PostgreSQL був розроблений для роботи на UNIX-подібних платформах. Потім PostgreSQL був розроблений і працював на різних платформах, таких як Windows, macOS і Solaris.

Нижче наведено поширені випадки використання PostgreSQL[7].

1) Надійна база даних у стеку LAPP

LAPP означає Linux, Apache, PostgreSQL і PHP (або Python і Perl). PostgreSQL в основному використовується як надійна внутрішня база даних, яка забезпечує роботу багатьох динамічних веб-сайтів і веб-додатків.

2) База даних транзакцій загального призначення

Великі корпорації та стартапи використовують PostgreSQL як основні бази даних для підтримки своїх програм і продуктів.

3) Геопросторова база даних

PostgreSQL з розширенням PostGIS підтримує геопросторові бази даних для геоінформаційних систем (ГІС). PostgreSQL підтримує більшість популярних мов програмування: Python, Java, C#, C/C+, Ruby, JavaScript (Node.js), Perl.

PostgreSQL має багато розширених функцій, які пропонують інші системи керування базами даних корпоративного класу, наприклад:

- Типи, визначені користувачем.
- Спадкування таблиці.
- Складний механізм блокування
- Посилальна цілісність зовнішнього ключа.
- Вкладені транзакції (точки збереження)
- Багатоверсійний контроль паралельності (MVCC).
- Асинхронна реплікація.

Останні версії PostgreSQL підтримують такі функції:

- Власна версія Microsoft Windows Server.
- Таблиці.
- Відновлення в момент часу.

Хто використовує PostgreSQL

Багато компаній створили продукти та рішення на основі PostgreSQL. Деякі представлені компанії: Apple, Fujitsu, Red Hat, Cisco, Instagram тощо [2].

2.5 React.js – бібліотека JavaScript

React - це JavaScript-бібліотека для створення інтерфейсів користувача. Важливо звернути увагу, що це саме бібліотека, а не фреймворк.

По-перше, його використання ні до чого не зобов'язує розробника, адже React не формує "фрейм" проекту.

По-друге, React[4] виконує єдине завдання: генерує на сторінці компонент інтерфейсу, синхронізуючі його з даними програми, і лише цієї бібліотеки в загальному випадку недостатньо для того, щоб повністю реалізувати проект.

Незабаром після появи React, з’явилися подібні до нього рішення (Vue.js, Svelte), тому що вони допомагають вирішувати проблеми, ґрунтуючись на ідеї декларативного програмування, а не на імперативному підході. Декларативний підхід полягає у описі кінцевого результату. При імперативному підході описуються конкретні кроки задля досягнення кінцевого результату. Виявилося, що декларативний підхід чудово підходить для створення інтерфейсів, і він прижився у суспільстві.

React спрощує розробку, бо за допомогою цієї бібліотеки можна створити інтерфейс використовуючи окремі компоненти[5]. Зникає необхідність працювати безпосередньо з DOM деревом, тому що ця бібліотека додає додатковтй шар абстракції.

Зазвичай у проектах налагоджений процес збирання за допомогою таких комплектувальників модулів як: webpack, parcel, rollup або інші, але треба зазначити, що це необов’язково при використанні React. Використовуючи при розробці цю бібліотеку, програміст пише код на чистому JavaScript і не має необхідності вивчати нову мову або використовувати розширення, але з React зазвичай використовують JSX[8].

React – це проект із відкритим вихідним кодом. Завдяки цьому його можна безпечно використовувати навіть у комерційних програмах.

Однак при використанні React є особливості, які важливо враховувати

- Розмір програми значно збільшиться через використання React (~40 kB для пакетів React та React-dom).
- Код виконується в браузері, що робить запуск програми повільніше. Із віртуальним DOM деревом виникають свої непередбачені витрати: по-перше, витрати часу через виконання (порівняння віртуальних дерев відбувається не одразу); по-друге, віртуальні дерева необхідно. десь зберігати, то ж це займає досить значну кількість пам'яті.
- Чим більше елементів на сторінці, тим більші витрати і це стає реальною проблемою на мобільних девайсах.

Враховуючи всі особливості, можна визначити яким проектам підійде React:

- Проєкт, який буде надалі розширюватись і над ним буде працювати команда розробників. Використовуючи вже відому технологію, зробить планування розробки та підтримування коду значно простішим.
- Завдяки компонентному підходу, який є основою бібліотеки React, її можна використовувати для середніх і великих проектів. Даний підхід спрощує процес структурування та дає виграш у довгостроковій перспективі.
- Legacy-проектам, які мають пройти через рефакторинг. Бо React можна додавати до вже існуючого проекту і оновлювати код поступово.

Ще React буде не кращим вибором якщо необхідно реалізувати елементи проекту, які є чутливими до споживаних ресурсів. В даній ситуації можливе гібридне використання: коли сама програма здебільшого написана на React, а вимогливі до продуктивності місця з ним не взаємодіють – бібліотека ніяк не заважає робити такі інтеграції [4].

2.6 JSX — розширення скриптової мови JavaScript

React охоплює той факт, що логіка візуалізації по суті поєднана з іншою логікою інтерфейсу користувача: як обробляються події, як змінюється стан з часом і як дані готуються до відображення.

Замість того, щоб штучно розділяти технології шляхом розміщення розмітки та логіки в окремих файлах, React розділяє проблеми за допомогою компонентів. React не вимагає використання JSX, але це може бути дуже зручно. Це також дозволяє React показувати більше корисних повідомлень про помилки та попередження.

JSX[3] має подібний до XML/HTML синтаксис, який використовується React, для розширення ECMAScript, щоб текст, подібний до XML/HTML, міг співіснувати разом із кодом JavaScript/React в одному документі. Знайдений HTML-подібний текст, знайдений у файлах з розширенням .js, перетворюється препроцессорами (тобто транспіляторами, такими як Babel) у об’єкти JavaScript.

За допомогою JSX можна писати короткі HTML/XML-подібні структури (DOM-подібні деревоподібні структури) в той самий файл, де пишеться код JavaScript, тоді Babel перетворить ці вирази на фактичний код JavaScript [3].

2.7 Postman для тестування API

Postman — програма для тестування API. Це популярний API клієнт, який дозволяє розробляти, тестувати та документувати API.

Тестувальники, за допомогою Postman можуть надсилати HTTP/s запити до серверів та отримувати від них відповіді. За допомогою такого підходу можна протестувати бекенд сервери та переконатися, що вони коректно працюють.
Основне призначення програми – створення колекцій із запитами до API.

Будь-який розробник або тестувальник, відкривши колекцію, зможе легко розібратися в роботі серверу.

Postman – популярний інструмент. Він має такі переваги:

- Безкоштовний. Postman – безкоштовний інструмент.
- Простий у використанні. Дуже просто почати користуватися – Postman інтуїтивно зрозумілий.
- Підтримує різні API. За допомогою Postman можна виконувати різні типи запитів до будь-яких API (REST, SOAP, GraphQL).
- Розширюваний. Postman можна настроїти під ваші конкретні потреби за допомогою Postman API.
- Інтегрований. Можна легко інтегрувати набори тестів у CI/CD інструмент

за допомогою Newman (CLI collection runner – дозволяє запускати Postman-колекції у командному рядку).

- Велике ком'юніті. Postman дуже популярний і, як наслідок, має велике ком'юніті, яке підкаже відповіді на більшість запитань.

Також Postman дає можливість створювати та проектувати на основі Mock-серверу проектувати дизайн API. Реалізацію сервера та клієнта можна запустити одночасно. Прямо з Postman можна писати тести та проводити автоматизоване тестування. Для адміністраторів передбачена ще можливість створення колекцій для моніторингу сервісів.

2.8 Фреймворк Express.js

Програмний продукт реалізується як RESTful API веб-система на Node.js тому вибір Express.js, як програмний каркас для розробки серверної частини — це послідовне рішення.

Фреймворк Express.js, реалізує набір функцій, що необхідні для створення ефективних API, це значно спрощує роботу по написанню програмного коду, тому пропорційно зменшує витрачений на розробку час.

За допомогою Express.js[6] можна структурувати веб-систему так, щоб система мала змогу обробляти одразу декілька HTTP запитів по заданій URL адресі.

Схема праці фреймворка Express.js зображена на рисунку 2.6.

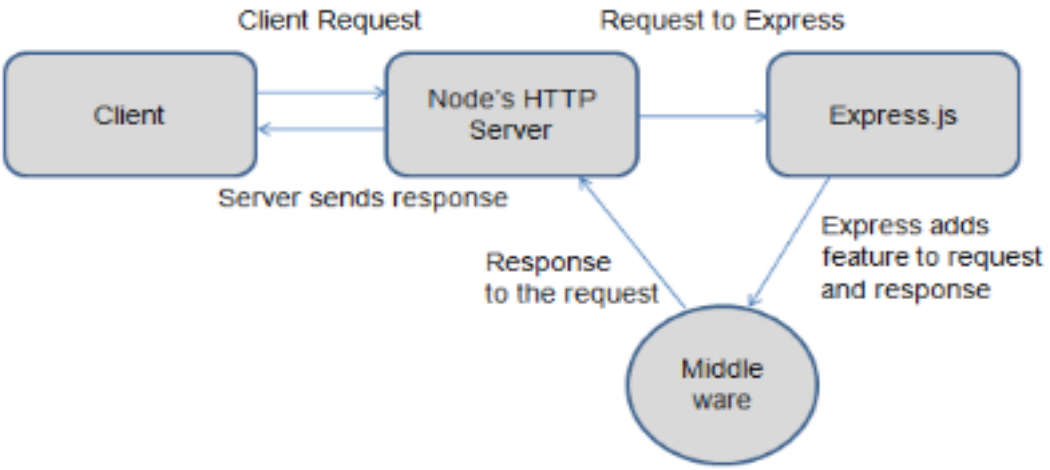

Рисунок 2.6 — Схема праці фреймворка Express.js

Node.js Express надає можливість користуватися проміжними обробниками Middleware та вже готовими функціями для обробки HTTP запитів[11— 12], використання яких і підрозуміває обрана REST архітектура. Цей фреймворк дає можливість зручно визначати маршрути програми на основі методів HTTP.

Даний фреймворк також неабияк допомагає з керуванням серверів та з налаштуванням маршрутів.

2.9 Бібліотека Jsonwebtoken

Бібліотека jsonwebtoken надає можливість створювати JSON Web Token[16], для безпечної передачі даних на клієнт. На рисунку 2.7 изображена структура JSON Web Token.

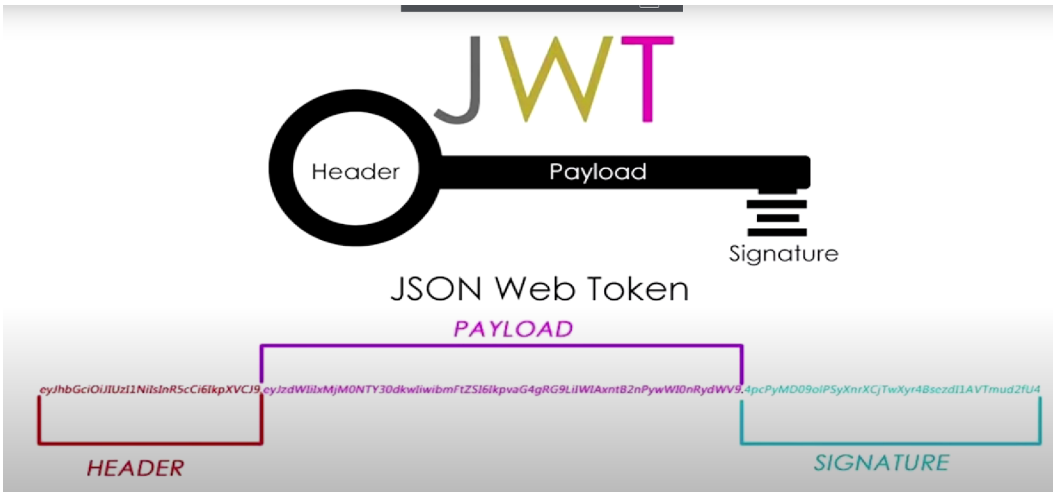

Рисунок 2.7 — Структура JSON Web Token

Веб-система підрозумовує використання функції авторизації, для доступу до даних користувача. За допомогою JSON Web Token ми можемо безпечно зберігати та передавати інформацію у зашифрованому вигляді на стороні користувача за допомогою localstorage або файлів cookie. Та при кожному відвідуванні користувачем сторінки веб-системи перевіряти збережений JSON Web Token користувача та надавати доступ до його особистого кабінета без авторизації.

Схема використання JSON Web Token для авторизації зображена на рисунку 2.8.

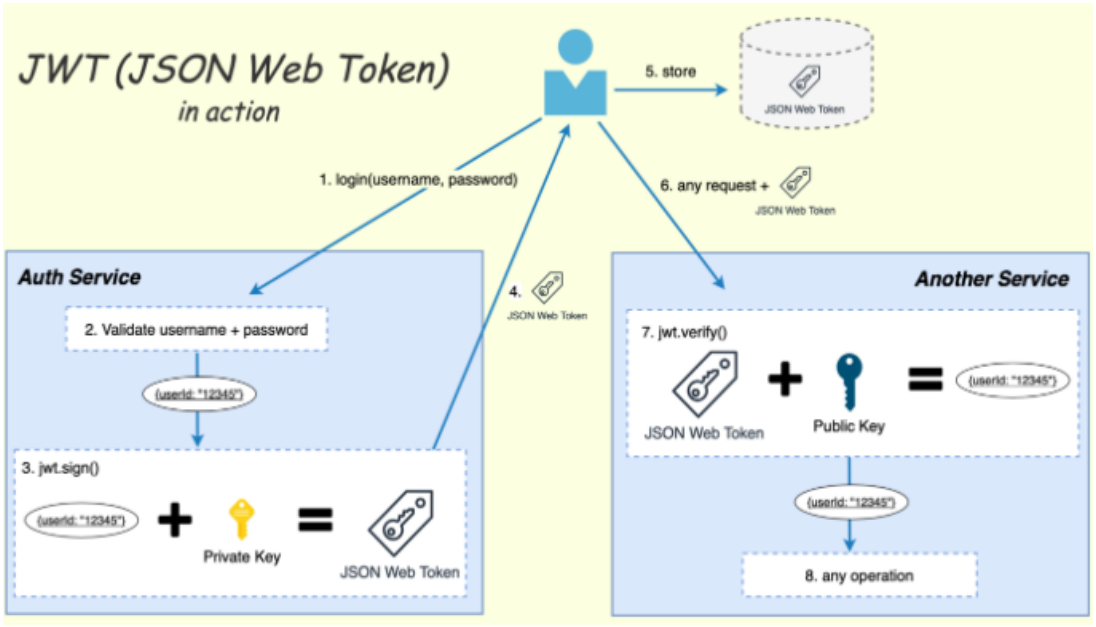

Рисунок 2.8 — Схема використання JSON Web Token

JSON Web Token можна задати тривалість життя токену, час впродовж котрого користувач може входити у системи без ручної авторизації, що значно підвищить безпеку.

2.10 Visual Studio Code

Visual Studio Code – візуальний редактор коду компанії Майкрософт. Це потужний програмний продукт, дивлячись на свою легкість, розробникам вдалося зробити його досить функціональним і корисним. Редактор чудово розуміє javascript, TypeScript або Node.js, але при необхідності можливості програми можуть бути істотно розширені за рахунок швидкої установки розширень. Цей варіант програми призначено для роботи в операційній системі Windows. Для прискорення роботи реалізована система гарячих клавіш, призначення команд яких може бути змінено будь-якої миті на зручній карті швидких клавіш. Також

добре працює функція експорту проектів у текстових форматах. Visual Studio Code чудово підходить розробникам веб-застосунків, хмарного програмного забезпечення, при перегляді, редагуванні код.

У даному продукті інтегровано безліч корисних функцій, а також вбудований інструмент компіляції коду. Це справді чудовий помічник для роботи кодом виправлення, розробки, тестування та створення ПЗ.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Аналіз предметної області та проектування є першими етапами у життєвому циклі створення програмного рішення. Одним із результатів цього етапу є діаграма варіантів використання (Use Case), що описує основні групи користувачів системи та варіанти її використання.

Веб-сервіс має виконувати наступні задачі:

- Авторизація.
- Завантаження аудіофайлу.
- Перегляд метаданих аудіофайлу.
- Редагування метаданих аудіофайлу.
- Скачування аудіофайлу.
- Створення каталогу збережених аудіофайлів.

На рисунку 3.1 зображено діаграма прецедентів по поставленим задачам.

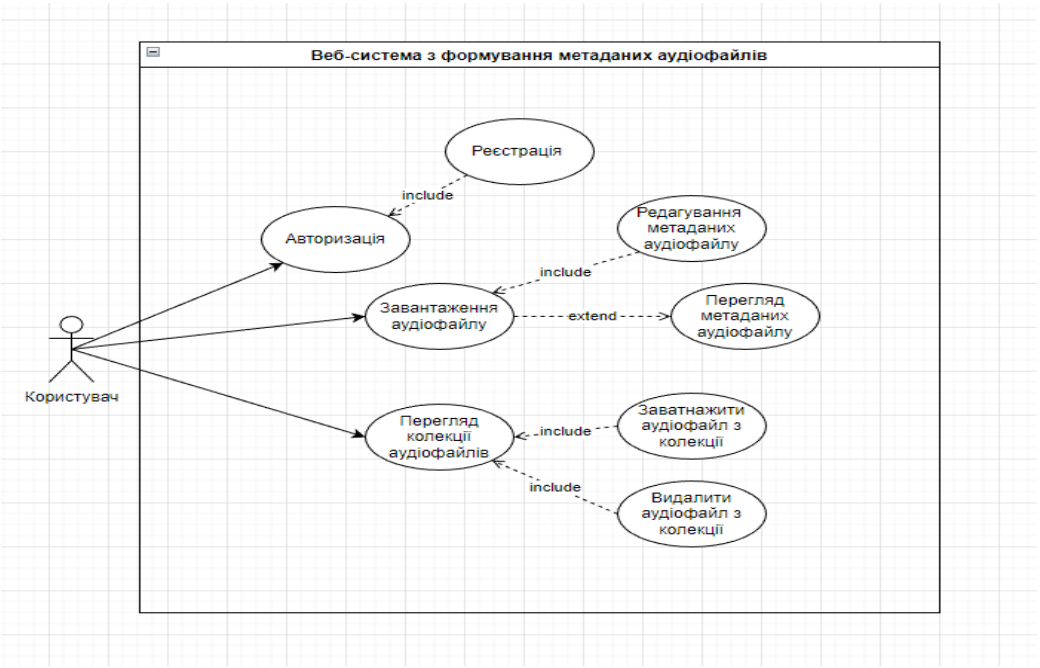

Рисунок 3.1 — Діаграма прецедентів

Для авторизації у системі користувачу необхідно зареєструватися. Для редагування та перегляду метаданих треба завантажити аудіофайл, список завантажених файлів користувач може переглянути у своїй колекції, звідки можна аудіофайли завантажувати на свій пристрій або видаляти с колекції.

Далі необхідно спроектувати концептуальну модель бази даних. Концептуальне проектування починається з аналізу предметної галузі, включає аналіз концептуальних вимог та інформаційних потреб, виявлення інформаційних об'єктів та зв’язків між ними, побудова концептуальної моделі (схеми) даних. Схема зображена на рисунку 3.2.

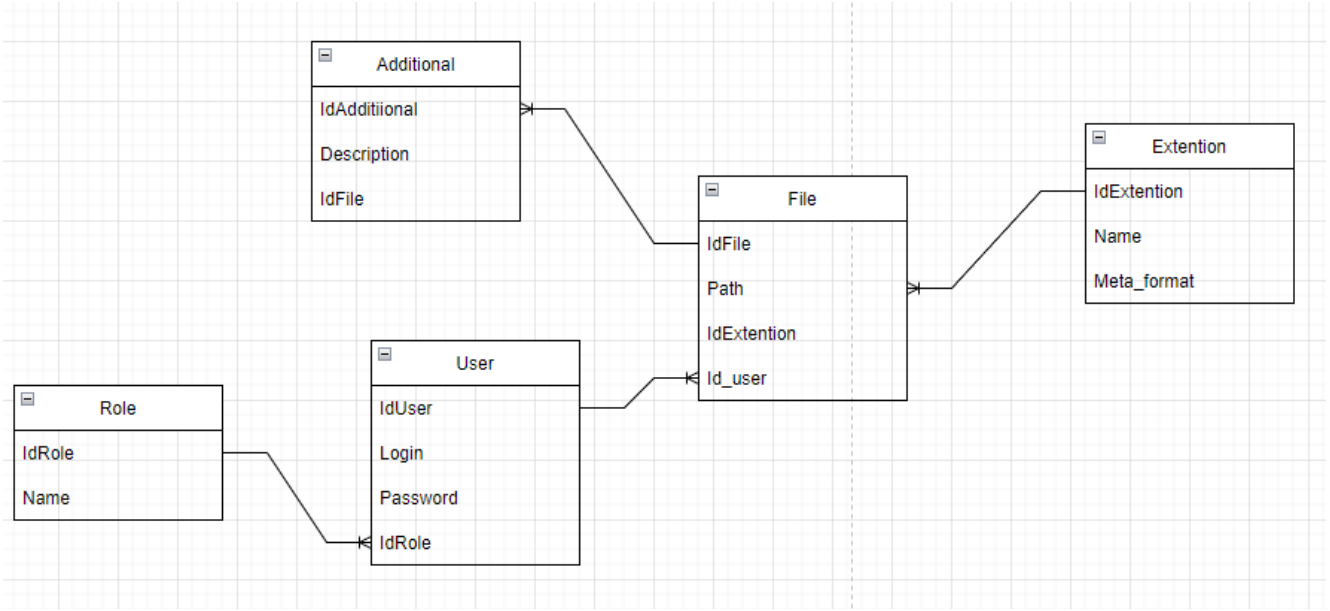

Рисунок 3.2 — Концептуальна модель бази даних

Для створення бази даних на PostgreSQL скористаємося объектно-реляційним-відображенням Sequelize з фреймворка Node.Js, для побудови бази даних використовуючи мову JavaScript, застосунок можна побачити на рисунку 3.3.

Створення бази даних за допомогою Sequelize надасть нам змогу працювати з таблицями субд, як зі звичайними JavaScript объектами.

```
const sequelize = require('./../db/db');
const {DataTypes} = require('sequelize');

const User = sequelize.define('user', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
    email: {type: DataTypes.STRING, unique: true},
    password: {type: DataTypes.STRING},
});

const Additional = sequelize.define('additional', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
    description: {type: DataTypes.STRING, allowNull: false},
});

const File = sequelize.define('file', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
    path: {type: DataTypes.STRING, unique: true, allowNull: false},
    extention: {type: DataTypes.STRING, allowNull: false},
});
User.hasMany(File);
File.belongsTo(User);

File.hasOne(Additional);
Additional.belongsTo(File);
```

Рисунок 3.3 — Реалізація бази даних

Перший вхід до системи буде складатися з 2 пунктів:

- Реєстрація.
- Авторизація.

Під час реєстрації дані користувача заносяться у бд і для безпеки даних користувача його пароль зберігається у зашифрованому вигляді. Шифрування відбувається за допомогою пакета каталогу npm, бібліотеки bcrypt, що можна побачити на рисунку 3.4.

```
async registration(req, res, next) {
    const {email, password, role} = req.body;
    if (!email || !password) {
        return next(ApiError.badRequest('Not valid email or password'));
    }

    const candidate = await User.findOne({where: {email}});
    if(candidate) {
        return next(ApiError.badRequest('User with this email already exists'));
    }

    const hashPassword = await bcrypt.hash(password, 5);
    const user = await User.create({email, role, password: hashPassword})
    const token = generateJwt(user.id, user.email, user.role);
    return res.json({token});
}
```

Рисунок 3.4 — Реєстрація та шифрування паролю

Також на рисунку 3.4 можно побачити що, при вході у систему буде генеруватися спеціальний jwt токен за допомогою бібліотеки jwtwebtoken який буде зберігати логін та пароль користувача у зашифрованому вигляді. Цей веб токен буде зберігатися у користувача в localstorage користувача та надасть можливість залишатися у системі навіть після перезавантаження сторінки, за допомогою спеціального middleware протокола, що будуть зчитувати токен користувача при всіх запитах користувача до серверу і перевіряти наявність його авторизації у системі, якщо токен дійсний то авторизація буде відбуватися автоматично, а токен перезаписуватемится, оновлюючи свій термін життя.

На рисунку 3.5 зображена структура jwt токену.

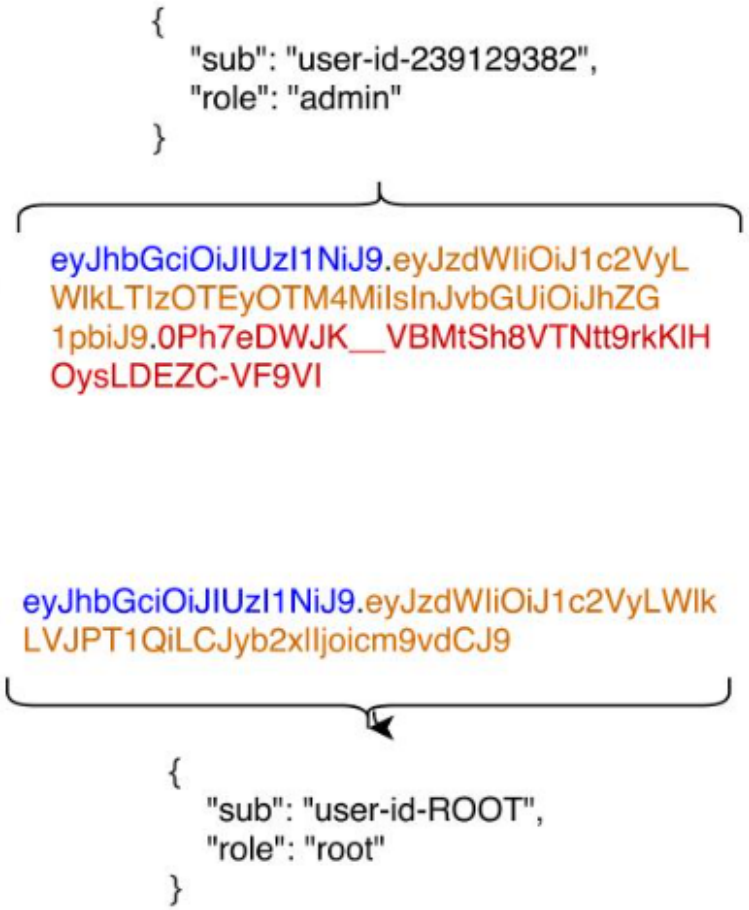

Рисунок 3.5 — Структура збереженя данних у jwt токені

Після входу до системи користувача автоматично переносить до вкладинки завантаження файлу. Завантаження файлу на сервер відбувається за допомогою HTML інтерфейсу Drag and Drop. Після чого відбувається збереження файлу на сервері і внесення його до бази даних, за допомогою бібліотеки uuid ми генеруємо унікальне значення, як на рисунку 3.6, яке буде слугувати файлу – назвою, під якою він зберігатиметься у спеціальній папці для статики, а у базу даних вже буде записуватися саме шлях за яким ми зберегли файл на сервері.

Рисунок 3.6 — Алгоритм генерації унікального uuid

Після завантаження аудіо файлу в систему за допомогою бібліотеки music-metadata-browser розпарсимо файл і виймаємо його теги(метадані) і відправляємо на кліент.

Бібліотека ffmetadata підтримує майже всі аудіо формати, та являю собою пакет, менеджера пакетів, npm фреймворку Node.js. Яка дозволяє зчитувати метадані відеофайлів та редагувати їх.

Користувач може переглянути всі метадані файла на кліенті та редагувати їх за допомогою панелі редагування. Далі за допомогою тієї ж бібліотеки Node.js ffmetadata записуємо отриманні від користувача дані у теги файлу, та зберігаємо його на сервері. Для подальшого використання цього аудіо файлу користувачем веб-системи.

Отже, в результаті аналізу програмного продукту та вимог, які були висунуті при проектуванні, було обрано та обґрунтовано засоби форматування метаданих аудіофайлів користувача. Визначено набір задач і методів їх рішення для побудови веб-системи.

4 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Програмний застосунок використовує web-технології та потребує від користувача стабільне під'єднання до мережі інтернет та будь-який браузер що використовує ES-6 методи, наприклад: Google Chrome, Mozilla Firefox, тощо.

При відвідуванні сайту користувач бачить перед собою вікно авторизації, користувачу необхідно авторизуватися у системі щоб почати роботу. Вікно авторизації зображено на рисунку 4.1.

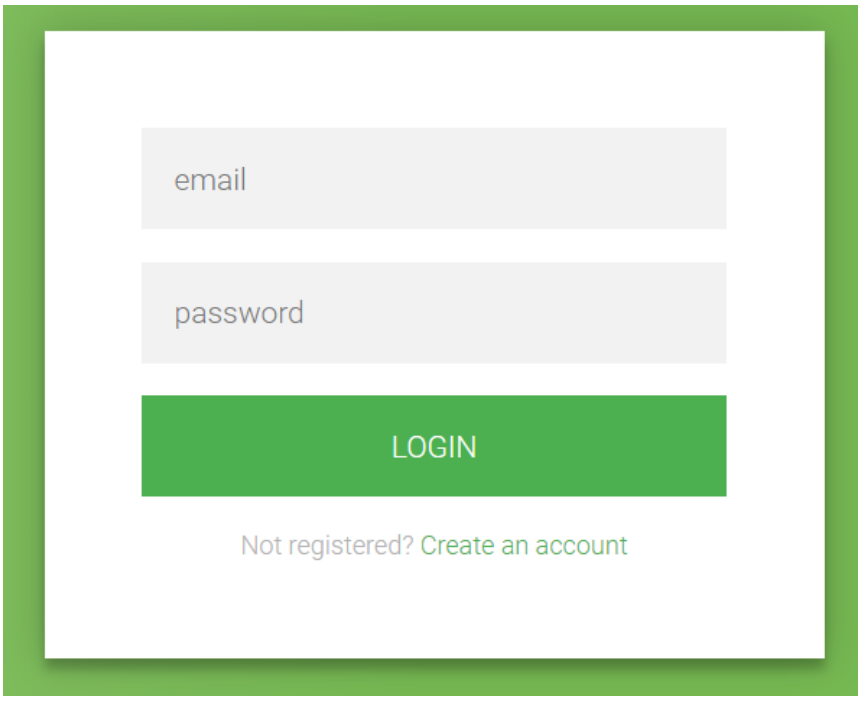

Рисунок 4.1— Вхід в систему

Якщо користувач не має акаунту то він має нажати на кнопку «Create an account» і перейти до вікна авторизації та зареєструватися, на рисунку 4.2 показана форма реєстрації.

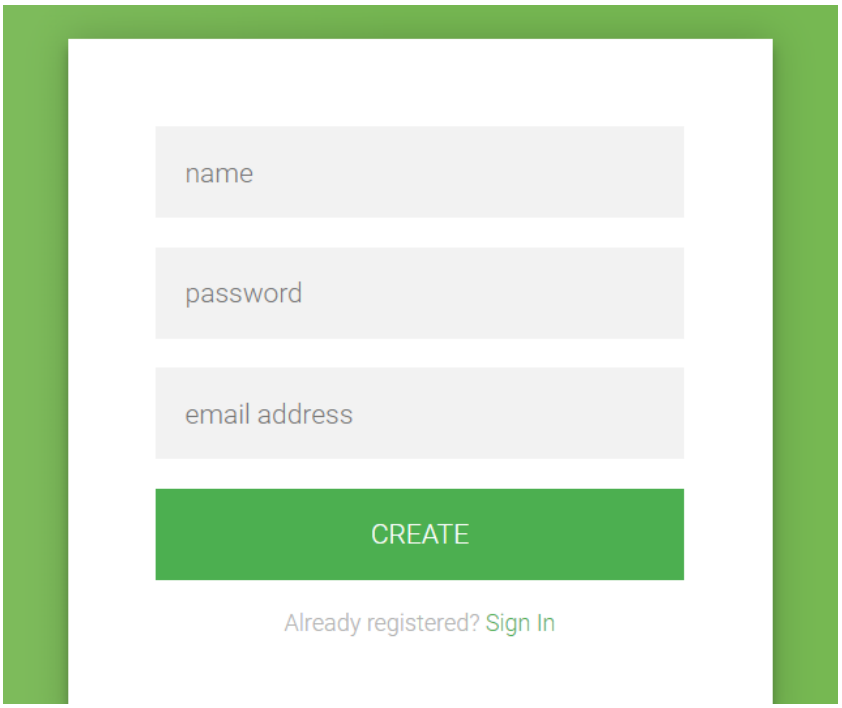

Рисунок 4.2 — Реєстрація

Далі користувач побаче вікно веб-сервісу з компонентом drag and drop, куди він зможе зручно перетягувати аудіо файли, які він хоче завантажити до системи для перегляду метаданих і подальшого їх редагування. На рисунку 4.3 зображено вікно з компонентом drag and drop.

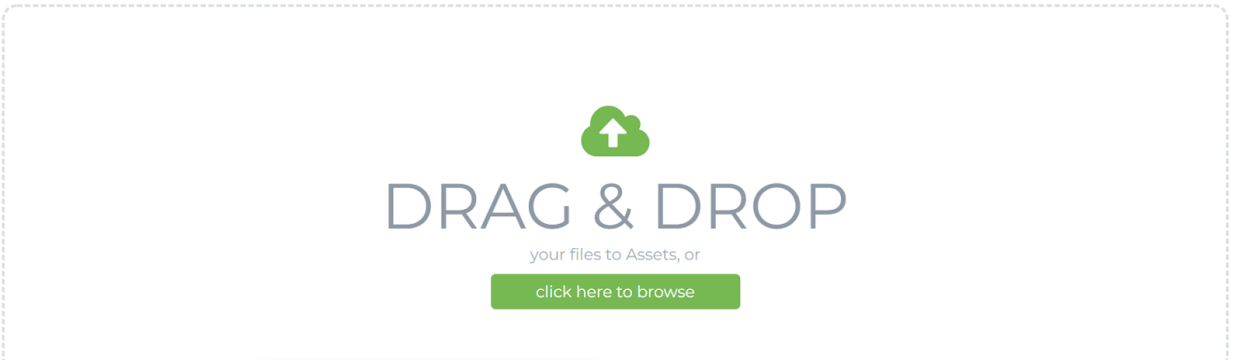

Рисунок 4.3 — Вікно drag and drop

Після того як користувач завантажив бажаний файл у нього відкривається вікно, що зображено на рисунку 4.4. Де користувач може побачити всі теги аудіо файлу розділеному на поля у дві колонки по принципу ключ - значення. У колонці зліва знаходиться назва тегу аудіо файла, а у колонці справа знаходиться поля форми у якій за замовчуванням відображаються значення що зберігаються у цих тегах.

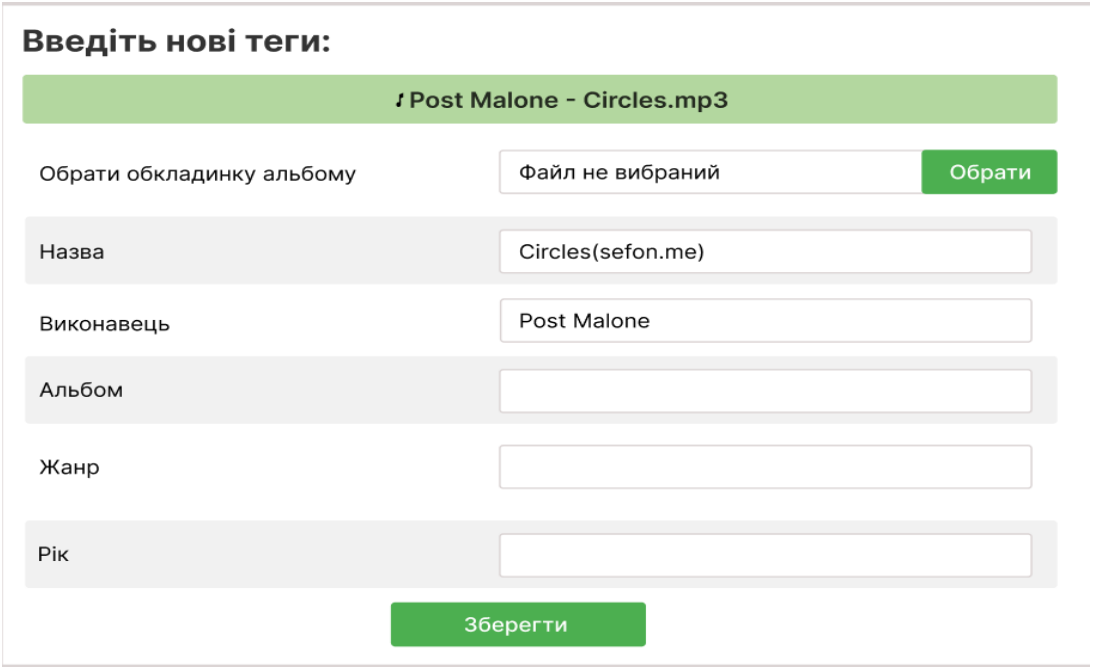

Рисунок 4.4 — Вікно для редагування тегів

Далі користувач може змінити значення у формах на свої та зберегти аудіофайл, після завершення операції з’являється вікно з підтвердженням, що редагування пройшло успішно, вікно зображенно на рисунку 4.5.

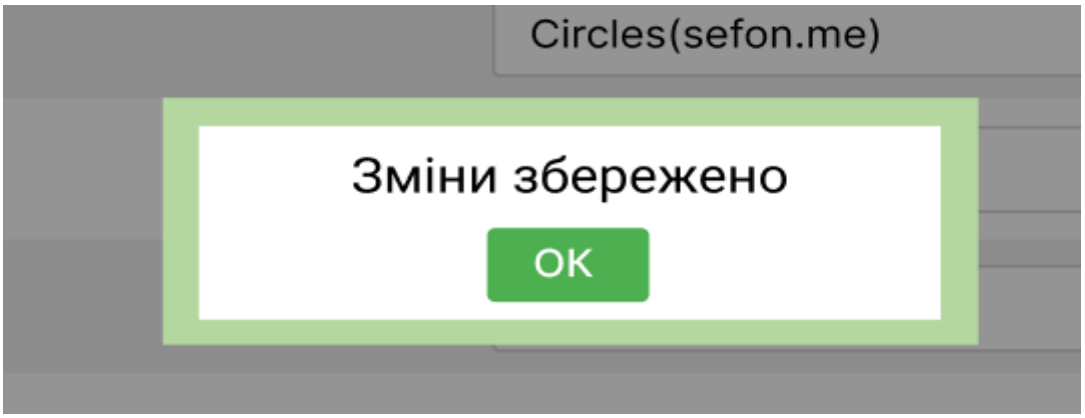

Рисунок 4.5 — Повідомлення, про збереження змін

Після натискання на кнопку “ОК” користувач переносится на сторінку каталогу своїх збережених файлів де може переглядати свої аудіофайли, які він завантажив, а також, за потреби редагувати їх повторно або видаляти. Доступ на сторінку каталогу мають тільки авторизовані користувачі. Якщо користувач не завантажував ніяких аудіо файлів, то каталог буде пустим.

Як зображено на рисунку 4.6 в каталозі відображаються назви музикальних композицій, псевдоніми авторів та альбомні обкладинки. Саме те що зберігається у тегах цих аудіофайлів. Вже звідси користувач побачить як змінилися теги аудіо файлів.

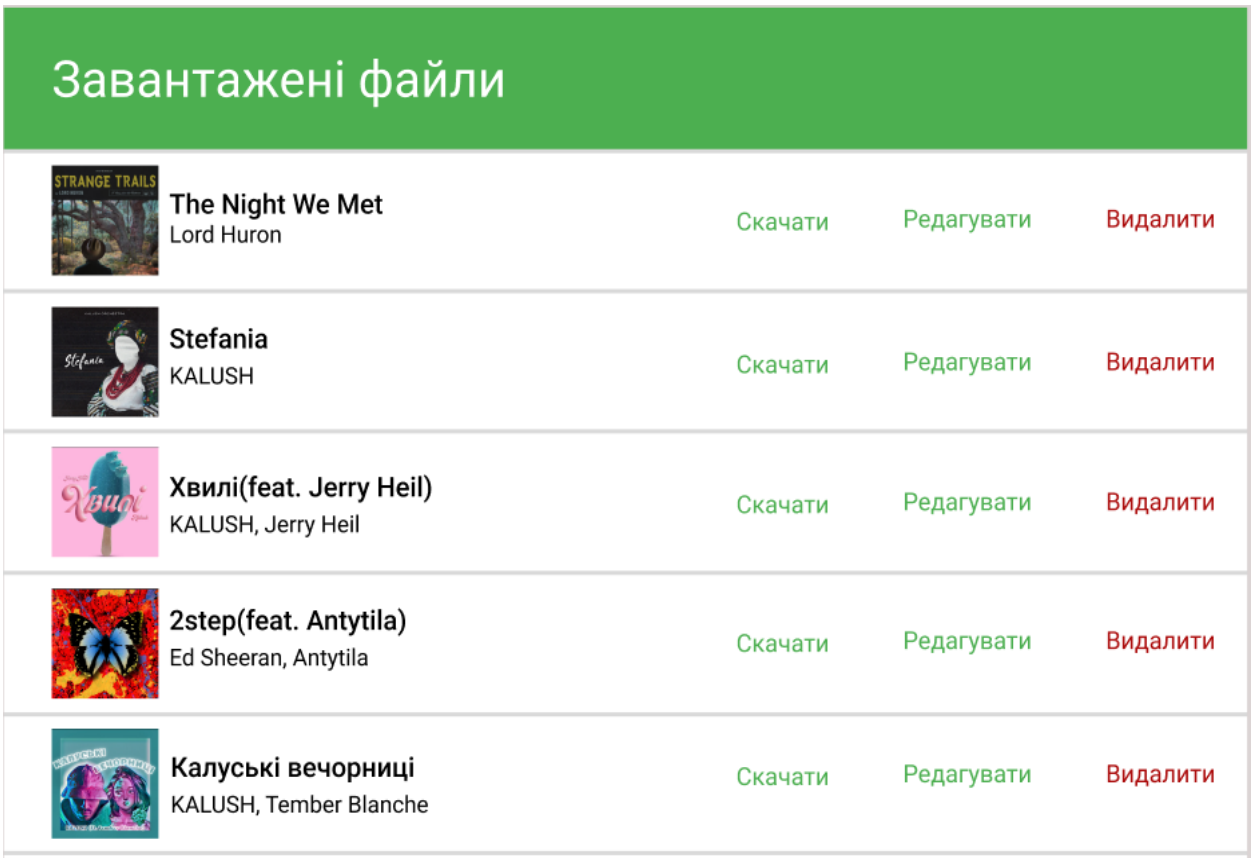

Рисунок 4.6 — Вікно перегляду завантажених файлів

В результаті користувач може зручно завантажувати аудіо файли, переглядати та змінювати їх метадані, зберігати аудіо файли у системі та скачувати відформатовані аудіо файли.

ВИСНОВКИ

В ході дипломної роботи було проаналізовано існуючі веб-системи для формування метаданих аудіофайлів. Було виявлено, що існуючі додатки мають недоліки, такі як: відсутність інструментів для пакетного перейменування, відсутність експорту та складність використання скриптів.

Були опрацьовані методи і засоби розробки веб-системи. Обґрунтовано вибір створення веб-системи, побудованої за REST архітектурою.

Було проведено комплексне тестування системи за допомогою HTTP-клієнта Postman, отриманні результати підтвердили коректне функціонування веб-сервісу. Отже система відповідає поставленим вимогам завдання.

Програмний застосунок може використовуватися будь-якою операційною системою, де встановлено браузер, що підтримує ES-6 методи та має стабільне під'єднання до мережі інтернет.

Отриманий в результаті розробки програмний продукт забезпечує користувачів зручним, інтуїтивно зрозумілим інтерфейсом з наступними функціями: авторизація, завантаження аудіофайлу, редагування метаданих аудіофайлу, функція перетягування Drag&Drop, перегляд метаданих аудіофайлу, скачування відредагованого аудіофайлу. Також користувач може зберігати свої аудіофайли для подальшого редагування та вільного доступу прямо у веб-системі з будь якого пристрою, що задовольняє мінімальним вимогам веб-сервісу обгрунтованих у роботі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Casciaro M., Mammino L. Node. js design patterns: design and implement production-grade node. js applications using proven patterns and techniques, 3rd edition. Packt Publishing, Limited, 2020. 660 p.
2. What is PostgreSQL. *PostgreSQL Tutorial - Learn PostgreSQL from Scratch*. URL: https://www.postgresqltutorial.com/postgresql-getting-started/what-is-postgresql/ (date of access: 04.04.2022).
3. Introducing JSX – React. *React – A JavaScript library for building user interfaces*. URL: https://reactjs.org/docs/introducing-jsx.html (date of access: 04.04.2022).
4. React Native - що це і як його освоїти | ApeironDB. URL: https://apeirondb.com/ua/blog/it-tehnologii-react-native (дата звернення: 04.04.2022).
5. Derks R. React projects: build 12 real-world applications from scratch using react, react native, and react 360. Packt Publishing, 2019. 474 p.
6. Express - Node.js web application framework. *Express - Node.js web application framework*. URL: https://expressjs.com/ (date of access: 08.05.2022).
7. Ferrari L., Pirozzi E. Learn PostgreSQL. Birmingham : Packt Publishing, 2020. 650 p.
8. Getting started with React - Learn web development | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started (date of access: 12.05.2022).
9. Herron D. Node.js web development. 5th ed. Birmingham : Packt Publishing, 2020. 760 p.
10. Introduction to node.js. *Introduction to Node.js*. URL: https://nodejs.dev/learn (date of access: 14.04.2022).

11. Kadwill T. Using sequelize ORM with node.js and express. *Stack Abuse*. URL: https://stackabuse.com/using-sequelize-orm-with-nodejs-and-express/ (date of access: 10.04.2022).
12. Node.js | конвейер обработки запроса и middleware. *METANIT.COM - Сайт о программировании*. URL: https://metanit.com/web/nodejs/4.2.php (дата звернення: 07.04.2022).
13. PostgreSQL: about. *PostgreSQL: The world's most advanced open source database*. URL: https://www.postgresql.org/about/ (date of access: 07.04.2022).
14. Schönig H.-J. Mastering PostgreSQL 13. 4th ed. Birmingham : Packt Publishing, 2020. 760 p.
15. What is JWT (JSON Web Token)? How does JWT Authentication work? - Blog - miniOrange. *Blog - miniOrange*. URL: https://blog.miniorange.com/what-is-jwt-json-web-token-how-does-jwt-authentication-work/ (date of access: 01.04.2022).
16. MP3TAG The universal tag editor. *Main features*. URL: https://www.mp3tag.de/en/ (date of access: 06.04.2022).
17. CambridgeAudio.-Blog- *Metadata in Digital Audio Files – What it is, where it is.* URL:https://www.cambridgeaudio.com/usa/en/blog/metadata-digital-audio-files-%E2%80%93-what-it-where-it-and-how-tidy-it (date of access: 06.04.2022).
18. Lumisoft - Luminescence Software: *Metatogger.* URL: https://www.luminescence-software.org/en/metatogger (date of access: 17.03.2022).

ДОДАТОК А

Веб-система з формування метаданих аудіофайлів

Текст програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського». ТР_22Б 12-1

Аркушів 7

Київ – 2022

1)Отримання метаданих аудіофайлу

```
const parseFile = (file)=>{
    let blob = file.getBlob();
    let tags = [];
    musicMetadata.parseBlob(blob).then(metadata => {
    if(metadata){
        const tagsArr = Object.keys(metadata)
        for (let i = 0; i < tagsArr.length; i++) {
            tags.push(metadata[tagsArr[i]]);
        }
    }
    })
    return tagsArr;
}
```

2)Функція перевірки токена користувача

```
const jwt = require('jsonwebtoken');

module.exports = function (req, res, next) {
    if(req.method === "OPTIONS") {
        next()
    }
    try {
        const token = req.headers.authorization.split(' ')[1];
        if(!token) {
            return res.status(401).json({message: "Not authorization"});
```

```
        }
        req.user = jwt.verify(token, process.env.SECRET_KEY);
        next();
    } catch (e) {
        res.status(401).json({message: "Not authorization"});
    }
};
```

3)Запити користувача для реєстрації, авторизації та перевірки токену

```
import {$authHost, $host} from "./index";
import jwt_decode from "jwt-decode";

export const registration = async (email, password) => {
        const {data} = await $host.post('api/user/registration', {email, password, role: 'ADMIN'});
    localStorage.setItem('token', data.token);
    return jwt_decode(data.token);
}

export const login = async (email, password) => {
    const {data} = await $host.post('api/user/login', {email, password});
    localStorage.setItem('token', data.token);
    return jwt_decode(data.token);
}

export const check = async () => {
    const {data} = await $authHost.get('api/user/auth');
    localStorage.setItem('token', data.token);
```

```
    return jwt_decode(data.token);
}
```

4)Пагінація сторінки каталогу

```
import React, {useContext} from 'react';
import {observer} from "mobx-react-lite";
import {Context} from "../index";
import {Pagination} from "react-bootstrap";

const Pages = observer(() => {
    const {audio} = useContext(Context);
    const pageCount = Math.ceil(device.totalCount / device.limit);
    const pages = [];

    for (let i = 0; i < pageCount; i++) {
        pages.push(i + 1);
    }

    return (
        <Pagination className="mt-5">
            {pages.map(page =>
                <Pagination.Item
                    key={page}
                    active={audio.page === page}
                    onClick={() => audio.setPage(page)}
                >
                    {page}
```

```
                </Pagination.Item>
            )}
        </Pagination>
    );
});

export default Pages;
```

5)Класс помилки

```
class ApiError extends Error {
    constructor(status, message) {
        super();
        this.status = status;
        this.message = message;
    }

    static badRequest(message) {
        return new ApiError(404, message);
    }

    static internal(message) {
        return new ApiError(500, message);
    }

    static forbidden(message) {
        return new ApiError(403, message);
    }
```

```
}

module.exports = ApiError;
```

6) Middleware для відлову помилок при запитах до серверу

```
const ApiError = require('./../error/apiError');

module.exports = function (err, req, res, next) {
    if(err instanceof ApiError) {
        return res.status(err.status).json({message: err.message})
    }

    return res.status(500).json({message: "Unexpected error"})
}
const ApiError = require('./../error/apiError');
const bcrypt = require('bcrypt');
const {User} = require('./../models/models');
const jwt = require('jsonwebtoken');
```

7)Генерація токену

```
const generateJwt = (id, email, role) => {
    return jwt.sign(
        {id, email, role},
        process.env.SECRET_KEY,
        {expiresIn: '24h'}
    );
}
```

8)Класс користувача, занесення даних до таблиць

```
class UserController {
    async registration(req, res, next) {
        const {email, password, role} = req.body;
        if (!email || !password) {
            return next(ApiError.badRequest('Not valid email or password'));
        }

        const candidate = await User.findOne({where: {email}});
        if(candidate) {
            return next(ApiError.badRequest('User with this email already exists'));
        }

        const hashPassword = await bcrypt.hash(password, 5);
        const user = await User.create({email, role, password: hashPassword})
        const token = generateJwt(user.id, user.email, user.role);
        return res.json({token});
    }


    async login(req, res, next) {
        const {email, password} = req.body;
        const user = await User.findOne({where: {email}});
        if(!user) {
            return next(ApiError.internal('User with this name didn\'t find'));
        }
        let comparePassword = bcrypt.compareSync(password, user.password);
```

```
        if(!comparePassword) {
            return next(ApiError.internal('Not valid password'));
        }
        const token = generateJwt(user.id, user.email, user.role);
        return res.json({token});
    }


    async check(req, res) {
        const token = generateJwt(req.user.id, req.user.email, req.user.role)
        return res.json({token})
    }}


module.exports = new UserController();
```