Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра комп'ютерної інженерії

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри,
канд. техн. наук, доцент

_____ Я. М. Крайник

« __ » _____ 2022 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Програмно-апаратний комплекс для потокової аудіопередачі

Спеціальність 123 Комп'ютерна інженерія

123 – КР.1 – 405.21810525

Студент: _____ К. Ю. Сюсько

« __ » _____ 2022 р.

Керівник: старший викладач _____ І. С. Бурлаченко

« __ » _____ 2022 р.

Миколаїв 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп’ютерних наук
Кафедра комп’ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри,
канд. техн. наук, доцент
_____Я.М. Крайник
« __ » _____
2022р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

Видано студенту групи 405 факультету комп’ютерних наук
Сюсько Кирилу Юрійовичу
(прізвище, ім’я, по батькові студента)

1. Тема кваліфікаційної роботи: Програмно-апаратний комплекс для потокової аудіопередачі**.** _____
Затверджена наказом по ЧНУ від «__» _____ 2022 р. No _____
2. Строк представлення кваліфікаційної роботи «____» _____ 2022 р.
3. Очікуваний результат роботи та початкові дані, якщо такі потрібні
Сервіс має отримувати дані від користувача, відображати іх на сторінці, на деякий сторінках повинен віддавати аудіо файли за допомогою потокової передачі, надавати права іншим користувачам, слухати, коментувати, додавати собі у альбоми або плейлісти.
_____
4. Перелік питань, що підлягають розробці

Проаналізувати існуючі high load потокові онлайн сервіси та вибрати стек технологій для реалізації програмно-апаратного комплексу, розробити апаратне забезпечення для організації серверної складової потокової аудіо-передачі, визначити ефективну архітектуру апаратного забезпечення, розробити програмне забезпечення для організації серверної та клієнтської складової потокової аудіо-передачі, налаштувати клієнт-серверну архітектуру застосунка.

5. Перелік графічних матеріалів

Скріншоти класів моделей, контролерів, валідаторів, команд в терміналі

Діаграма класів моделей проєкту

Блок-схема алгоритму роботи програмного забезпечення

Скріншот дизайну сторінки з потоковою аудіопередачею

6. Завдання до спеціальної частини

Розглянути основні державні норми України, щодо праці в умовах використання комп’ютерів та екраних пристроїв загалом, щодо вентиляції та кондиціонування, організація повітрообміну, норм шумів та вібрацій. Ознайомитись з правами та нормами праці в умовах використання комп’ютерів та екраних пристроїв загалом.

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
ст. викладач А. О. Алексєєва	кафедра екології Медичного інституту ЧНУ ім. Петра Могили	спеціальна частина з охорони праці

Керівник роботи ____ старший викладач ___І. С. Бурлаченко ___ _____

(посада, прізвище, ім’я, по батькові) (підпис)

Завдання прийнято до виконання __Сюсько Кирило Юрійович_____ _____

(прізвище, ім’я, по батькові студента) (підпис)

Дата видачі завдання «____» _____ 20_____ р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

Тема: Програмно-апаратний комплекс для потокової аудіопередачі

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КР	27.02.2022	20.03.2022	Виконано
2	Огляд літератури за темою роботи	21.03.2022	25.04.2022	Виконано
3	Складання календарного плану КР	26.04.2022	29.04.2022	Виконано
4	Аналіз предметної області	01.05.2022	06.05.2022	Виконано
5	Розробка проєктних рішень	07.05.2022	14.05.2022	Виконано
6	Моделювання та конструювання АПЗ	15.05.2022	02.06.2022	Виконано
7	Перевірка працездатності, тестування розробленого АПЗ, аналіз результатів тестування	03.06.2022	05.06.2022	Виконано
8	Відгук керівника КР	06.06.2022	07.06.2022	Виконано
9	Оформлення КР та презентації	08.06.2022	12.06.2022	Виконано
10	Попередній захист	13.06.2022	13.06.2022	Виконано
11	Рецензування	14.06.2022	16.06.2022	Виконано
12	Завершення оформлення КР та презентації	17.06.2022	25.06.2022	Виконано
13	Захист кваліфікаційної роботи	27.06.2022	28.06.2022	Виконано

Розробила студент_____Сюсько Кирило Юрійович_____ _____

(прізвище, ім’я, по батькові) (підпис)

«__» _____ 2022 р.

Керівник роботи ___ старший викладач , І. С. Бурлаченко ____ _____

(посада, прізвище, ім’я, по батькові) (підпис)

«__» _____

2022 р.

АНОТАЦІЯ

бакалаврської роботи

«Програмно-апаратний комплекс для потокової аудіопередачі»

Студент: Сюсько Кирило Юрійович

Керівник: старший викладач І. С. Бурлаченко

Бакалаврська робота присвячена розробці апаратно-програмного комплексу для потокової аудіопередачі. Розглянуто наявні в наш час сервіси з схожою спрямованістю. Практичне значення результатів дослідження та розроблення полягає в тому що, на підставі результатів можливе вдосконалення апаратно-програмного комплексу, у засвоєнні навичок процесу проектування, розробки, створення потокового онлайн сервісу, можливостях застосування розробленого програмно-апаратного комплексу.

Пояснювальна записка бакалаврської роботи складається зі вступу, трьох розділів, висновків та двох додатків. У вступі визначається актуальність теми, сформульовані мета, об’єкт, предмет та завдання дослідження та розроблення бакалаврської роботи. У першому розділі досліджуються та аналізуються різноманітні існуючі сервіси, апаратні та програмні рішення, на основі яких робляться висновки на основі яких робиться вибір стеку технологій. У другому розділі пояснюється вибір та демонструється апаратна складова проєкту. У третьому розділі наведені дані про програмну частину комплексу та результати тестувань. Четвертий розділ присвячений охороні праці, щодо норм праці на підприємствах. У висновках наведено аналіз виконаної роботи та отриманих результатів дослідження та розроблення.

У додатках наведений код з серверної та клієнтскої частинах проєкту

Ключові слова: потокова передача, норм праці на підприємствах з використанням комп’ютерів, серверна частина, клієнтська чатсина, Django, Vue.js, Postgresql.

ABSTRACT

of the Bachelor’s Thesis

"Hardware and software complex for people with respiratory disorders"

Student: Siusko Kyrylo Yuriiovych

Consultant: senior lecturer I. S. Burlachenko

The bachelor's thesis is devoted to the development of a hardware-software complex for streaming audio transmission. The services available at present with a similar orientation are considered. The practical significance of the results of research and development is that, based on the results, it is possible to improve the hardware and software complex, in mastering the skills of design, development, online streaming service, the possibility of using the developed software and hardware complex.

The explanatory note of the bachelor's thesis consists of an introduction, three sections, conclusions and two appendices. The introduction determines the relevance of the topic, formulates the purpose, object, subject and objectives of research and development of the bachelor's thesis. The first section explores and analyzes a variety of existing services, hardware and software solutions, based on which conclusions are drawn on the basis of which the choice of technology stack. The second section explains the selection and demonstrates the hardware component of the project. The third section provides data on the software part of the complex and test results. The fourth section is devoted to labor protection, on labor standards in enterprises. The conclusions provide an analysis of the work performed and the results of research and development.

The appendices contain code from the server and client parts of the project

Keywords: streaming, labor standards in enterprises using computers, server part, client chat, Django, Vue.js, Postgresql.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

MVC – Model View Controler

RSS – Rich Site Summary

URL – Uniform Resource Locator

MVVM – Model View View Model

SSR – Server Side Render

RFC – Request For Comment

API – Application Programming Interface

HTML – Hyper Text Markup Language

CSS – Cascading Style Sheets

Україномовні скорочення та абревіатури

ЧНУ – Чорноморський національний університет

ВСТУП

У вік цифрових технологій людство все більш проводить часу дома, нехтуючи розвитком соціальних навичок. Людям важливо бути потрібними та почутими у соціумі, обмінюватись своєю творчістю та розвиватись як музичний виконавець. Цю проблему вирішують потокові онлайн сервіси музики. Людина може поділитись з соціумом своєю працею, отримати відгук, після цього проаналізувати і рухатись далі. Також людина може прослухати чиюсь творчість та можливо навіть знайти щось цікаве для себе, що можливо модернізувати.

Також все більш соціальних мереж містять потокові онлайн сервіси музики, що є гарним індикатором актуальності. Людям подобається мати можливість прослухати музику, поставити відгук і це все в режимі онлайн. Музика завжди об'єднувала людей, під музику можна радіти, сумувати, танцювати або просто стрибати під крутий гітарний риф, загалом під будь-який спектр емоцій людина може знайти музику, яка буде їй до душі. За допомогою музики людина здатна передати свої почуття, кожна емоційна частина настрою людини повністю переходить на гітарний риф або біт.

Метою роботи: розроблення high-load онлайн сервісу потокової аудіо-передачі у соціальних мережах для розповсюдження музики.

Об’єкт: Веб-сервіс для організації соціальної взаємодії музикантів.

Предмет: технології побудови мікро-сервісної архітектури для організації high load потокової аудіо-передачі.

Для досягнення поставленої мети було поставлено наступні **задачі** :

- проаналізувати існуючі high load потокові онлайн сервіси та вибрати стек технологій для реалізації програмно-апаратного комплексу;

- розробити апаратне забезпечення для організації серверної складової потокової аудіо-передачі;
- визначити ефективну архітектуру апаратного забезпечення.
- розробити програмне забезпечення для організації серверної та клієнтської складової потокової аудіо-передачі;
- налаштувати клієнт-серверну архітектуру застосунка.

Практичне значення отриманих результатів полягає у тому, що розроблений застосунок буде може бути використаний для об’єднування людей навколо музики у соціальних мережах, таким чином сприяючи розвитку їхніх професійних музичних та соціальних навичок. Завдяки цьому сервісу люди зможуть перевіряти свої музичні матеріали на слухачах та отримувати оцінки та відгуки, на основі яких зможуть покращити свої здібності. Так само цей веб-застосунок буде сприяти спілкуванню людей і допомагати виходити новий рівень спілкування, так само застосунок може бути поштовхом для музичної особистості, що ще не сформувалася.

1. АНАЛІТИЧНИЙ ОГЛЯД ІСНУЮЧИХ ТЕХНОЛОГІЙ ПОТОКОВОЇ АУДІОПЕРЕДАЧІ

1.1 Мікросервісна архітектура серверного застосунка

Мікросервісна архітектура існує вже деякий час. Мікросервіс – це найпростіша одиниця, сервіс, який приймає вхідні запити для здійснення дії. Це може бути backend сервіс, який доступний цілодобово та без вихідних, або функція, яка викликається, коли відбувається подія. Простими словами, функція або набір функцій, доступних через певний API через мережу. Отже, це backend служба, розгорнута на сервері. У якомусь сенсі це монолітний застосунок. Однак він не несе в собі всю функціональність системи, а лише меншу частинку логіки. На відміну від моноліту, отриманий застосунку побудований як набір відносно невеликих незалежних служб, що називаються «мікросервісами», які комунікують через комп’ютерну мережу. Можна сказати, мікросервіси – це ті самі логічні модулі монолітного додатка, які розподілені через комп’ютерну мережу, замість того, щоб працювати в рамках одного процесу пристрою.

Проєкт планується писатись за допомогою мікросервісної архітектури, тому при виборі технологій потрібно ретельно розглянути реалізацію мікросервісної архітектури.

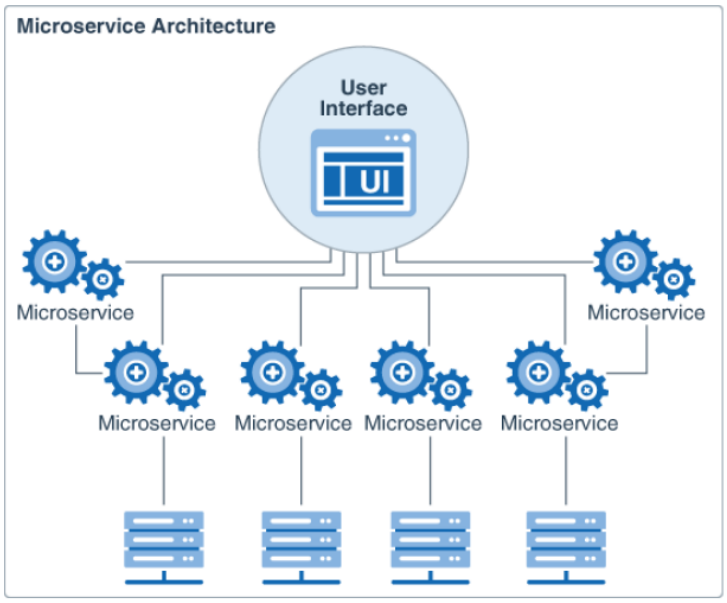

Рисунок 1.1 – Модель взаємодії мікросервісної архітектури

1.2 Клієнт-серверна архітектура проєкту

Клієнт — сервер це мережева або обчислювальна архітектура, в якій завдання або мережеве навантаження розподілені між постачальниками послуг, які називаються серверами, та замовниками послуг, які називають клієнтами. Фактично клієнт та сервер – це програмне забезпечення. Зазвичай серверна та клієнтська частини програми розташовані на різних обчислювальних машинах та взаємодіють між собою через мережу за допомогою мережевих протоколів, але інколи можуть розташовуватись також і на одній обчислювальній машині. Серверна частина проєкту завжди очікує від клієнтської частини запити, після отримання якого за логікою вирішують надавати чи не надавати їм свої ресурси у вигляді даних наприклад, як передача файлів за допомогою FTP, HTTP, HTTPS, потокова мультимедіа або робота з базами даних або у вигляді сервісних функцій наприклад таких як, робота з електронною поштою, спілкування за допомогою систем миттєвого обміну повідомленнями за

допомогою протоколу UDP або перегляду веб-сторінок у всесвітній мережі інтернет. Оскільки одна серверна частина може обробляти запити від багатьох клієнтських частин, її часто розміщують на спеціально виділеній обчислювальній машині, яка налаштовується особливим чином, як правило, спільно з іншими серверами, тому продуктивність цієї машини має бути високою. Через особливу роль цієї машини в мережі, специфіки її обладнання та програмного забезпечення її часто також називають сервером, а машини, які виконують клієнтські запроси, відповідно, клієнтами.

Взагалі кажучи, служба - це абстракція комп'ютерних ресурсів, і клієнту не потрібно турбуватися про те, як сервер працює під час виконання запиту та доставки відповіді. Клієнту потрібно тільки зрозуміти відповідь, засновану на відомому протоколі програми, тобто зміст та форматування даних для запитуваної послуги.

Клієнтська та серверна частини обмінюються повідомленнями за допомогою шаблону запит та відповідь. Клієнтська частина надсилає запит, а серверна повертає відповідь на цей запит. Цей обмін повідомленнями є прикладом між серверної взаємодії. Мова та правила спілкування визначені у протоколі зв'язку. Усі протоколи клієнт-серверної моделі працюють лише на рівні застосунків. Протокол прикладного рівня визначає основні шаблони діалогу. Щоб ще більше формалізувати обмін даними, сервер може продати інтерфейс прикладного програмування API. API – це рівень абстракції для доступу до сервісу. Обмежуючи зв'язок певним форматом контенту, він полегшує синтаксичний аналіз. Абстрагуючи доступ, він полегшує між платформний обмін даними.

Сервер може отримувати запити від багатьох клієнтів за короткий період часу. Комп'ютер може виконувати лише обмежену кількість завдань у будь-який момент і покладається на систему планування для визначення пріоритетів

вхідних запитів від клієнтів для їхнього задоволення. Щоб запобігти зловживанням та максимізувати доступність серверне програмне забезпечення може обмежувати доступність для клієнтів. Атаки типу "відмова в обслуговуванні" використовують обов'язки сервера обробляти запити, такі атаки діють шляхом навантаження сервера надмірною частотою запитів. Шифрування слід застосовувати, якщо між клієнтом та сервером має передаватися конфіденційна інформація.

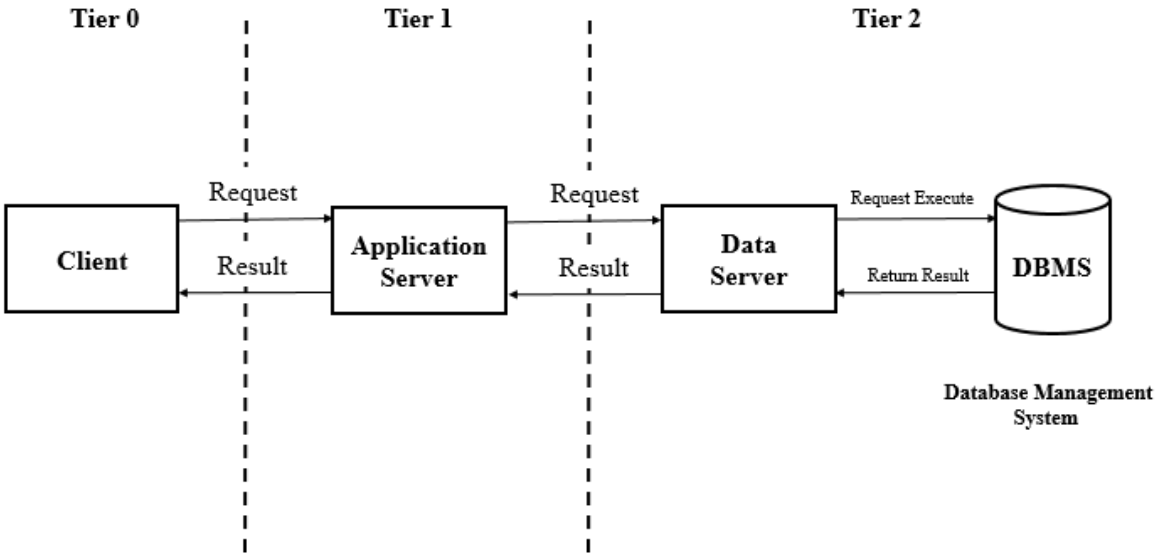

Рисунок 1.2 – Модель взаємодії клієнт-серверної архітектури

1.3 Аналіз фреймворку Laravel Django як платформи для розробки серверної частини

Laravel — безкоштовний, з відкритим кодом PHP-фреймворк, створений Тейлор Отвел і призначений для розробки вебдодатків відповідно до шаблону model–view–controller.

Серед особливостей фреймворку Laravel можна назвати такі можливості як, модульна система упакування з виділеним менеджером залежностей

Composer, різні способи для доступу до бази даних, утиліти, які допомагають в розгортанні застосунків і технічного обслуговування, а також його орієнтація на синтаксичний цукор.

Станом на березень 2015 року, Laravel вважався одним з найпопулярніших PHP фреймворком, разом з Symfony5, Nette, CodeIgniter, Yii2 й іншими.

Сирцевий код Laravel'a розміщується на GitHub і ліцензований відповідно до умов MIT License

Тейлор Отвел створив Laravel як спробу забезпечити досконалішу альтернативу фреймворку CodeIgniter, який не забезпечував певні функції, такі як вбудовану підтримку для аутентифікації і авторизації користувачів. Перша бета-версія Laravel була розміщена 9 червня 2011 року, пізніше випустили Laravel 1 в тому ж місяці. Laravel 1 містить вбудовану підтримку для аутентифікації, локалізації, моделі, уявлення, сесій, маршрутизації та інших механізмів, але відсутня підтримка контролерів, що не заважає йому бути справжнім MVC фреймворком.

Laravel 2 був випущений у вересні 2011 року, Laravel 3 - у лютому 2012 року, Laravel 4 -у травні 2013 року, Laravel 5 - у лютому 2015 року, Laravel 6 - у вересні 2019 року, Laravel 7 - у березні 2020 року, Laravel 8 - у вересні 2020 року.

У 2015 році в результаті опитування sitepoint.com з використання PHP-фреймворків серед програмістів зайняв перше місце у номінаціях:

- Фреймворк корпоративного рівня.
- Фреймворк для особистих проєктів.
- Кладезь антипаттернів.

Фреймворк Laravel написаний на високорівневій мові програмування PHP. Цей фреймворк є дуже популярним. Тому що мова на якій він написан дуже

проста для входження у програмування. Дуже багато новачків обирають саме цю мову програмування. Так як його дуже часто обирають, навколо нього сформувалось непогане коло користувачів, а це означає, що якщо виникне питання, то google можно з легкістю знайти на нього відповідь.

1.4 Аналіз та аргументація вибору Django як платформи для розробки серверної частини

Django поступово зростала та виходила на нові ступені визнання під час розробки реальних проєктів, створених командою розробників в Лоуренсі, штат Канзас, США. Він народився наприкінці 2003 року, коли розробники газети Lawrence Journal World, Едріан Холоваті і Сімон Віллісон, почали використовувати мову програмування Python для розробки своїх вебзастосунків. Команда World Online, відповідальна за розробку та підтримку декількох місцевих новинних сайтів, зростала та процвітала серед розробників, з великою терміновістю роботи журналістів. Для сайтів, включаючи Lawrence-Journal-World.com, Lawrencies.com та KIUsports.com, менеджери та журналісти компанії вимагали, щоб нові можливості та цілі проєкту були розроблені з максимальною швидкістю, часто час йшов на дні або на години. Таким чином, Симон і Едріан почали розроблювати середу розробки що заощаджує час розробки, виходячи з потреб компаній. Це був єдиний спосіб, за допомогою якого вони могли створювати керовані програмні застосунки в жорстких рамках часових термінів.

Django це високорівневий фреймворк написаний на мові програмування Python. Фреймворк Django надає програмний каркас для розробки сервісів. Django бува названий на честь джазмена Джанго Рейнхардта, тому що один з головних розробників дуже полюбляв цього музиканта.

Сайт на Django будується з однієї або декількох частин, які рекомендується робити модульними. Це одна з істотних архітектурних відмінностей цього фреймворку від деяких інших.

Django дотримується архітектури модель-вигляд-контролер тобто MVC. Але, те що в класичному розумінні архітектури MVC зветься контролером в Django називається вигляд, а те, що мало бути виглядом, називається шаблоном. Таким чином розробники Django часто використовують замість MVC, MTV – що є модель-шаблон-вигляд.

Початкова розробка на фреймворку Django дуже проста, так як всі потрібні засоби для роботи ресурсів надаються відразу після створення проєкту.

Фреймворк Django надає великий ряд засобів, які допомагають пришвидшити розробку веб-застосунків. Можно навести такий приклад, розробнику не потрібно створювати контролер та сторінку для адміністративної панелі сайту, Django зробить все сам, в нього є вбудований модуль для керування вмістом, який можна включити або виключити в кореневому файлі налаштувань, після вміст може керуватись відразу декількома сайтами з одного сервера. Адміністративна панель дозволяє створювати, змінювати і видаляти будь-які об'єкти наповнення сайту, записуючи всі дії, також адміністративна панель надає інтерфейс для управління користувачами та цілими групами користувачів з різними правами доступу до контенту.

У фреймворку Django також включені компоненти для системи коментарів, синдикації RSS і Atom, статичних сторінок якими можна управляти без потреби писати контролери та шаблони відображення, перенаправлення URL та інше.

Django підтримує парадигму ООП. Об'єкти бази даних в термінології Django називаються моделями. Фреймворк Django надає у використання розробників розвинутий, прикладний програмний інтерфейс який є

високорівневим доступом до даних, які зберігаються у базі даних. У великому кількості випадків немає потреби писати SQL-запити, але суворої заборони немає, якщо потрібно можно з легкістю використовувати SQL-запити.

```python
class Album(models.Model):
    """
        ORM Model for track's albums
    """
    user = models.ForeignKey(AuthUser, on_delete=models.CASCADE, related_name='albums')
    name = models.CharField(max_length=50)
    description = models.TextField(max_length=1000)
    private = models.BooleanField(default=False)
    cover = models.ImageField(
        upload_to=get_path_upload_cover_album,
        blank=True,
        null=True,
        validators=[FileExtensionValidator(allowed_extensions=['jpg']), validate_size_image]
    )
```

Рисунок 1.3 – Модель Альбомів

Автоматична побудова інтерфейсу для адміністрування. Django автоматично створить для вас CRUD-інтерфейс панелі адміністрування.

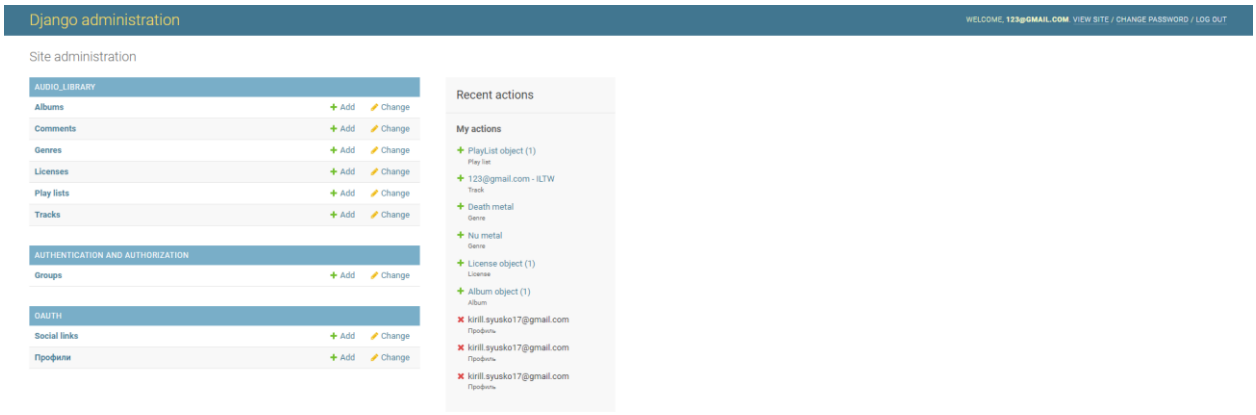

Рисунок 1.4 – Стандартна панель адміністрування Django

Елегантні URL. Парсери URL-ів побудовані на регулярних виразах. Фреймворк Django не обмежує розробників у використанні конкретної схеми посилань.

Зручна система шаблонів. В Django для роботи з html шаблонами існує окрема мова для опису шаблонів, вона називається jinja. Jinja це дуже проста і дружня мова шаблонів для непрограмістів. В цій мові присутні оператори циклу, умови та форматування даних. З практики верстальника можна освоїти та редагувати Django-шаблони за 2 заняття. Мова шаблонів виконується як функція відображення даних, якщо Django використовується як серверна та клієнтська частина одночасно. Операторами мови шаблонів jinja не можливо змінити дані в базі даних.

Гнучка підсистема кешування. У фреймворку Django проєкт може бути налаштований на роботу з memory cache або будь-яким іншим фреймворком, це робиться дуже просто, потрібно лише змінити налаштування в кореневому файлі налаштувань проєкту. Інструменти Django дозволяють кешувати SQL-вибірки, цілі шаблони або їх частини, звичайні окремі змінні та навіть самі url шляхи.

Проста інтернаціоналізація. Переклади у Django проєктах не є проблемою. Інтернаціоналізація головною концепцією є лінивий переклад. Це насамперед означає, що якщо певний рядок тексту не був знайдений у файлах перекладу, то буде використано базовий текст й не буде показано повідомлення про помилку. Проте повної заборони на використання функцій, які будуть обробляти помилки за наявність перекладу рядкових даних немає. Для перекладу тексту всередині програмного коду використовується функція gettext, яка зазвичай імпортується як нижнє підкреслення - «_».

Конфігурація серверу. Django створювався для роботи під управлінням вебсервера Apache з такими модулями як mod python, використанням

PostgreSQL як бази даних. На даний момент Django також підтримує FastCGI, mod wsgi, що є синхроною складовою фреймворку та asgi, що в свою чергу є асинхроною складовою фреймворку, наприклад для роботи з веб-сокетами. Веб сервером для Django на даний момент також можуть бути Apache, CherryPy, nginx, lighttpd. Django також підтримує такі системи баз даних як SQLite MySQL, PostgreSQL та Oracle. Фреймворк Django також має власний вебсервер для розробки та налагоджування. Сервер може автоматично відслідковувати зміни у файлах кореневого коду і перезапускається, що зручно при розробці проєкту.

Підтримка Rest API. Для того щоб написати кінцеві точки потрібно встановити додатковий модуль, який називається Dango REST Framework, скорочено DRF. Після встановлення DRF, все що потрібно зробити, це взяти дані з бази даних за допомогою ORM моделі, серіалізувати ці данні та віддати їх на кінцеву точку.

Найбільш суттєвим плюсом фреймфорку Django є його вбудовані класи. Для прикладу розглянемо найбільш часто використовуємий вбудований клас «ModelViewSet» який імпортується з модуля rest_framwork.

```
class LicenseView(viewsets.ModelViewSet):
    """
        CRUD for author's licenses
    """
    model = models.License
    serializer_class = serializer.LicenseSerializer
```

Рисунок 1.5 – Приклад використання класу «ModelViewSet»

Після створення класу нам потрібно успадкуватись від класу ModelViewSet і нам стає доступна вся логіка цього класу. Отже щоб отримати 5 працюючих кінцевих точок нам потрібно лише 3 строки коду, це дуже

прискорює написання програм, достатньо вказати клас моделі та клас Serializer. На виході ми отримуємо повний CRUD набір. Але це не все, в цей клас можливо додати безліч чого потрібного, ми можемо додати клас пагінації, фільтрації, парсеру та клас перевірки на права доступу.

Django має дуже багато плюсів, всі вони обумовлені швидкістю розробки і зручності для програміста. Але не варто забувати і про мінуси, Django досі є асинхронним фреймворком, так само він написаний мовою програмування Python, який є скриптовою, інтропретованою мовою програмування, що у свою чергу робить його повільним.

1.5 Аналіз платформи React.js як платформи для розробки клєінтської частини

React.js це бібліотека написана на мові JavaScript з відкритим кодом. Бібліотека React.js була розроблена для створення інтерфейсів користувача, яка повина вирішувати проблеми оновлення вмісту вебсторінки, з якими в основному стикаються розробники односторінкових застосунків. Ця бібліотека розробляється компанією Facebook та спільнотою індивідуальних програмістів. React.js також дозволяє створювати великі веб застосунки, які використовують дані, котрі беруться за допомогою запитів до серверу та змінюються з часом, без перезавантаження сторінки. Його головною метою є те, щоб бути достатньо швидким, масштабованим та простим для розробників будь-якого рівня. Бібліотека React.js обробляє тільки користувацькі інтерфейси у застосунках. Це відповідає шаблону модель-вигляд-контролер, тобто MVC, також вона може бути використана у поєднанні з іншими JavaScript бібліотеками або великими фреймворками MVC, таких як Vue.js та AngularJS. Вона також може бути

використана з React на основі надбудов, щоб обробляти частини коду, що не мають користувацького інтерфейсу побудови веб застосунків. Як бібліотеку інтерфейсу користувача React.js найчастіше використовують разом з іншими бібліотеками, такими як Redux для роботи з незмінними частинами коду.

В даний час React використовують такі великі компанії як Sony, Netflix, Yahoo, Khan Academy, Atlassian, Airbnb та інші.

Бібліотеку було створено Джорданом Волком, програмістом який працює у компанії Facebook. Автор працював над проєктом під впливом Laravel фреймворку для PHP. У 2011 року Facebook випустили фреймворк під назвою React.js, за рік до цього вони використали посилання на нього у своєму блозі в Instagram. Також бібліотека була представлена як проєкт з відкритим початковим кодом на конференції всіх розробників JSConf US, що проходила у США у травні 2013 року. На цій конференції React.js влаштовану компанією Facebook у березні проєкт було представлено як бібліотеку з відкритим програмним забезпеченням.

Одностороння передача даних. Властивості передаються в обробник компоненту, як властивості html тегу і після цього вимальовуються на сторінках. Компонент React.js не може напряму змінювати властивості атрибутів, що йому передаються, але може їх змінювати через функції які називаються callback. Такий механізм отримав назву «властивості донизу, події нагору».

Віртуальний DOM. Бібліотека React.js повністю підтримує віртуальний DOM, а не працює виключно з DOM браузера. За допомогою цього бібліотека визначає, які частини DOM змінилися, порівнію її зі збереженою версією віртуального DOM, і таким чином можно визначити, як найефективніше оновити DOM браузера. Таким чином програміст працює зі сторінкою,

вважаючи що вона оновлюється вся, але бібліотека самостійно вирішує які компоненти сторінки треба оновити.

JSX. Компоненти React зазвичай написані на основі JSX. JSX це мова шаблонізатору для бібліотеки React.js. Код який був написаний на JSX компілюється та має тригери на виклики методів бібліотеки React. Також розробник може писати на чистій мові програмування JavaScript. JSX дуже сильно нагадує іншу мову шаблонів, яку також створили у компанії Facebook, але для розширення PHP.

Не лише відмальовка HTML в браузері. В бібліотеці React.js використовують не лише для відмальовування HTML в браузері. Наприклад, компанія Facebook має динамічні графіки які відмальовуються в HTML теги <canvas>, такі компаніх як PayPal та Netflix використовують ізоморфне завантаження для відмальовування ідентичного HTML на боці серверу та клієнту.

Методи життєвого циклу біблотеки React.js це ряд методів, які вбудовані у корінь React.js. Вони дозволяють розробникам обробляти дані в різних точках життєвого циклу застосунку написаному за допомогою React.js. Наприклад такі методи як:

- shouldComponentUpdate це метод життєвого циклу, за допомогою якого Javascript оновить компонент, використовуючи при цьому логічні змінні.
- componentWillMount це також метод життєвого циклу, за допомогою якого Javascript налаштувує певні дані перед обробленням компонентів вставленних у віртуальний DOM.
- componentDidMount це метод життєвого циклу, який є подібний до компонента WillMount, але він працює після методу render який в свою чергу відмальовує сторінку в HTML DOM, і може

використовуватися для додавання JSON-даних, також він презначений для визначення властивостей та станів.

Метод render є найважливішим методом життєвого циклу, він є необхідним для будь-якого компоненту біблоітеки React.js. Метод render це те, що з'єднує код JSX і відображає йогов же як відмальовану сторінку в DOM.

Вкладені елементи. Кілька елементів які розташовані на одному рівні повинні бути згорнутими в один елемент контейнеру, найчастіще це є елемент , або повернутий як список у вигляді масиву.

Атрибути. JSX надають ряд атрибутів та елементів які призначені для відображення їх у форматі HTML. Також атрибути користувача можуть бути передані до компоненту. Всі атрибути які було отримано компонентом будуть збережені як реквізит.

Вирази написані на мові програмування JavaScript можна використовувати в JSX за домогою фігурних дужок, сам код пишеться в середині них. Для підтримки концепції щодо одностороннього потоку даних у бібліотеці React.js, використовується архітектура Flux, яка являє собою альтернативу популярній архітектурі модель-вигляд-контролер, тобто MVC. Flux це функції дій, які надсилаються через центральний обробник до сховища, а зміни зі сховища надсилають назад до вигляду. Якщо використовувати разом з біблоітекою React.js та архітектуру Flux це здійснюється за допомогою компоненти властивостей.

Бібліотека React.js в заємодії з архітектурою Flux не повинена безпосередньо змінювати будь-які реквізити, які були передані, але повинен передати функції зворотного виклику callback, щов свою чергу створює дії, які відправляють до обробника та до модифікації сховища. Сховище також можна розглядати як модель адже воно також може змінюватися у відповідь на дії, отримані від обробника.

Цей шаблон іноді виражається як властивості стікають вниз, дії — вгору. Дуже багато реалізацій Flux створено з моменту його створення, мабуть, найбільш відомим з них є Redux, він має одне сховище, яке часто називають єдине джерело істини, це практика структурування інформаційних моделей та пов'язаних ним схем даних, так що кожен елемент даних зберігається рівно один раз.

1.6 Аналіз та аргументація вибору Vue.js як платформи для розробки клієнтської частини

Vue.js це фреймворк який написаний на мові програмування JavaScript та має відкритий доступ до вихідного коду для створення інтерфейсів користувача. Легко інтегрується у проєкти з використанням інших JavaScript модулей бібіліотек. Може використовуватись як веб-фреймворк для розробки лише односторінкових програм у реактивному стилі так і цілих проєктів.

На даний момент підтримується автором Еваном Ю та іншими активними членами основної команди з різних компаній, таких як Netlify, Netguru, Baidu, Livestorm.

Опитування, проведене в 2016 році для JavaScript, показало, що Vue має 89% задоволеності розробників. На GitHub проєкт має 188 тисяч зірок, є третім за величиною проєктом в історії GitHub

У 2013 році співробітник Google Еван Ю, працюючи над одним з проєктів, дійшов висновку, що не існує готових рішень для швидкого прототипування складних інтерфейсів користувача веб-додатків: React тоді був на ранній стадії розробки, основними інструментами були такі складні фреймворки, як AngularJS або орієнтований на MVC-архітектуру Backbone.js, які не відрізнялися простотою і орієнтовані на розробку великих додатків. Для

подолання цього пропуску Ю почав розробку Vue.js, яка, зберігаючи простоту, виявилася придатною як прототипування, але й повноцінної розробки.

У жовтні 2015 року було випущено версію 1.0 бібліотеки, у вересні 2016 року — версію 2.0.

Починаючи з версії 2.0 він підтримує також рендеринг за сервера, він же SSR Server-Side Rendering.

18 вересня 2020 року була випущена версія Vue.js 3.0.0 «One Piece», за словами розробників «забезпечує покращену продуктивність, менший розмір пакетів, кращу інтеграцію з TypeScript, нові API для вирішення великомасштабних завдань і міцну основу для майбутніх перспективі »[9].

Реліз 3.0 увібрав у себе понад 2 роки зусиль з розробки, включаючи понад 30 RFC, понад 2600 коммітів, 628 запитів від 99 розробників, плюс величезний обсяг роботи над розробкою та документацією. Весь код був переписаний більш ефективний TypeScript, що дає переваги в гнучкій розробці. Також представлений новий набір API Composition.

Остання версія Vue.js 3.2 була видана 5 серпня 2021 року під назвою The Quintessential Quintuplets.

Основною концепцією розробники називають Vue.js прогресивним та поступово адаптованим у порівнянні з іншими веб-фреймворками.

Це дозволяє розробнику настроїти структуру програми відповідно до власних вимог. Розробники вважають Vue.js більш простим у освоєнні, ніж той самий AngularJS, адже API побудований набагато простіше для освоєння. У Vue.js можна використовувати лише знання JavaScript та HTML. Можливе застосування Typescript. У Vue.js є власна офіційна досить багата документація на багатьох мовах, викладена на vuejs.org, яка може стати прикладом у поясненні проєктування та розробки в браузері. В середині фреймворка Vue.js реалізується шаблон MVVM, тобто Vue.js надає можливість прив'язки даних на

Javascript, так що виведення та введення даних сполучаються безпосередньо з джерелом даних тобто за допомогою динамічних моделей Vue.js. Таким чином, режим ручного визначення даних наприклад через jQuery з HTML та самим DOM не потрібен. При цьому немає потреби в жодних додаткових анотаціях, як у Knockout.js, оголошені у Vue-Element звичайні змінні JavaScript включаються як реактивні елементи.

Реактивність у Vue.js означає, що представлення моделі MVC змінюється динамічно, тобто в міру зміни моделі. У Vue розробник може просто прив'язати уявлення до відповідної модел Vue.js і, Vue.js автоматично спостерігає за змінами моделі і перемальовує уявлення для цілого DOM. Ці функції роблять управління станом Vue.js досить простим та інтуїтивно зрозумілим.

Фреймворк Vue.js такоє має багато складових для роботи ефектів переходу між сторінками. Vue надає різні способи застосування ефектів переходу під час вставлення, оновлення або видалення DOM. Включає такі інструменти які надають такі можливості:

- Можливість автоматично застосовувати CSS класи та атрибути при переходах та анімації
- Можливість працювати зі сторонніми модулями для анімації CSS, наприклад Animate.css
- Можливість використовувати JavaScript методи для перехоплення переходу, щоб керувати безпосередньо DOM
- Можливість використовувати фреймворк у поєднанні зі сторонніми бібліотеками анімації JavaScript, наприклад такими як Velocity.js.

Модульність. Vue.js може бути доповнений розширеннями, вони можуть містити Mixins, директиви, фільтри, компоненти та об'єкти. Як офіційні розширення пропонуються Vuex, менеджер станів, підключений до Redux, і маршрутизатор Vue, компонентний маршрутизатор. У розділі awesome-Vue

підтримується поточна колекція розширень. Наприклад, є розширення для виконання HTTP-запитів. проте ця була створена як офіційна бібліотека з Vue.js-Portfolio.

Компоненти. Щоб краще адаптуватися до цього корисно для великих проєктів, де може знадобитися модуляризація, Vue може і зберігається такі компоненти як окремий файл з розширенням .vue як визначення «компонентів одного файлу», замість використання Vue.component для зареєстрованих компонентів. Всередині окремі файлові компоненти HTML, CSS та JavaScript вставляються в блоки.

Розробники можуть використовувати інструменти складання Webpack або Browserify як для однофайлових, так і для упаковки компонентів.

Отже, чому саме Vue.js. Це відносно новий фреймворк, він дуже схожий на React.js і Angular.js, тому що всі вони виконують ту саму функцію. Обробку клієнтських запитів. Він має дуже зручний і дружній синтаксис, хорошу та головне швидку реактивність, яка обходить за швидкістю той самий React.js.

За час його існування він об'єднав дуже велике коло розробників і створив велике співтовариство. У нього найкраща документація з усіх фронтенд фреймворків, яка так само підтримує різні мови. Так само в новому релізі з'явилося нове Composition API, яке має ще кращий синтаксис від того, що було раніше.

1.7 Порівняльний аналіз вже існуючих проєктів

Аналіз робочих аналогів

Spotify AB – компанія розробник рушія Spotify, має на рахунку дуже вдалий багатокористувацький проєкт. Так як планується багатокористувацький онлайн проєкт то буде розглянуто сервіс Spotify веб версію.

Spotify — стримінговий сервіс потокового аудіо, що дозволяє прослуховувати музичні композиції та підкастів. Надає послуги легального онлайнового стримінгу аудіозаписів основних світових та незалежних лейблів, у тому числі Sony, EMI, Warner Music Group, Universal та BBC. Був запущений у жовтні 2008 року шведською компанією Spotify AB, який на той час був стартапом.

Spotify є першим стримінговим сервісом, що дає можливість слухати музыку онлайн, не завантажуючи її на пристрій. Сервіс є доступним у всіх країнах Америки, Європи та Океанії, а також у більшості країн Азії та Африки.

Серверна частина якого була написана на мові програмування Python за допомогою фреймворку Django.

В цьому сервісі можно створити свій профіль, додати йому картинку, створити свої плейлисти, додати вже існуючі, так само ділитися своїми плейлистами з усіма, якщо профіль відкритий, якщо профіль закритого типу, то всі ваші плейлисти залишаться видимими тільки для вас.

Можно прослуховувати музику, але додати, щось своє, поділитися своєю творчістю простий користувач, не зможе, це можуть робити лише офіційно зареєстровані компанії.

Apple Music це сервіс який надає можливість прослуховувати музику онлайн. Сервіс Apple Music був розроблений компанією Apple. У користувачів є можливість самим обирати музику, яку вони бажають послухати на своєму пристрої. Apple Music включає в себе модеровану радіостанцію Beats 1 та платформу Connect. Сервіс пропонує підказки та рекомендації, які базуються на вподобаних пісень користувача, сервіс також має інтеграцію з голосовими командами Apple помічника Siri.

Ще до того як був створений свервіс Apple Music так продукти компанії Apple як iPod та iTunes були дуже відомими революційними продуктами в сфері цифрової музики.

Восени 2015 року з'явилась підтримка смартфонів на базі операційної системи Android та Apple TV. Онлайн радіо можна слухати, додавши радіостанцію в iTunes до закладки «Радіо». Користувачі Windows Vista та Windows XP мають доступ лише до iTunes Radio, але не мають доступу до стримінгових радіо, тому що вони не підтримують кодування DRM. Щоб мати доступу до онлайн трансяцій Apple Music необхідно мати обліковий запис Apple ID в країні, де цей сервіс працює.

Apple Music – це дуже гарно працюючий сервіс, в ньому є можливість додати музику у свій playlist та прослухати її, але цей сервіс має точно такі ж недоліки як і Spotify, в ньому немає можливості додати свою музику, тобто звичайний користувач не має змоги додати свою музику, це можуть зробити лише компанії у яких є на це ліцензії. Також тут немає можливості отримувати відгук на музику та багато інших недоліків, всі вони схожі з недоліками Spotify.

Для вирішення цих недоліків була поставлена задача розробити свій сервіс потокової аудіо-передачі.

1.8 Аналіз інструментів для реалізації проєкту

Для реалізації проєкту буде потрібне програмне забезпечення двох видів- для розробки серверної складової та клієнтської.

Так як мова розробки серверної частини це Python то середовище повинно буди адаптовано до цієї мови. Найкращім вибором, згідно за документацією, це буде PyCharm 2016 року та пізніше.

PyCharm це інтегроване середовище для розробки яке заточене під мову програмування Python. Це середовище для розробки надає засоби для аналізу

коду, графічний відладник, інструмент для запуску юніт-тестів та підтримує веброзробку на фреймворку Django. PyCharm розроблена компанією JetBrains на основі IntelliJ IDEA на мові програмування Java.

PyCharm працює під керуванням операційних системам Windows, Linux та Mac OS X.

Налаштування PyCharm для роботи з Django може допомогти підвищити ефективність і загальний досвід роботи. Для легкого доступу до звичайних дій під час розробки у PyCharm за допомогою рушія буде використано розширення Django, DjangoRest, Git, Docker, PostgersqlBD.

Щодо реалізації візуальної складової, то на кожен етап розробки існує призначене специфічне забезпечення. Розглянемо програмне забезпечення яке пристосоване до сумісної роботи з рушієм.

Так як мова розробки клієнтської частини це JavaScript то середовище повинно буди адаптовано до цієї мови. Найкращім вибором, згідно за документацією, це буде Visual Studio 2016 року та пізніше.

Microsoft Visual Studio це серія продуктів компанії Microsoft, вона включає в себе інтегроване середовище розробки для програмного забезпечення та інші інструменти. Дані продукти дозволяють програмісту розробляти як консольні програми, так і ігри, програми з графічним інтерфейсом, у тому числі з підтримкою технології Windows Forms, UWP а також веб-сайти, веб-програми, веб-служби як в рідному, так і в керованому кодах всіх платформ, підтримуваних Windows, Windows CE, .NET Framework, Windows Mobile, .NET Core, .NET, Xbox, MAUI, Windows Phone, .NET Compact Framework та Silverlight. Після придбання компанії Xamarin корпорацією Microsoft з'явилася можливість розробки IOS та Android програм.

Visual Studio включає редактор вихідного коду з підтримкою таких технології як IntelliSense та можливістю найпростішого рефакторингу коду.

Вбудований налагоджувач може працювати як налагоджувач рівня вихідного коду, так і налагоджувач машинного рівня. Інші вбудовані інструменти включають редактор Windows Form для спрощення створення графічного інтерфейсу програми, веб-редактор, як дизайнер класів та дизайнер схеми бази даних. Visual Studio дозволяє створювати та підключати сторонні бібліотеки та плагіни для розширення функціональності практично на кожному рівні, включаючи додавання та підтримки систем контролю версій вихідного коду як, наприклад, Visual SourceSafe та Subversion, додавання нових наборів інструментів наприклад, для редагування та візуального проєктування коду предметно-орієнтованими мовами програмування або інструментами для інших аспектів процесу розробки програмного забезпечення наприклад, клієнт Team Explorer для роботи з Team Foundation Server.

Налаштування Visual Studio для роботи з Vue.js може допомогти підвищити ефективність і загальний досвід роботи. Для легкого доступу до звичайних дій під час розробки у Visual Studio за допомогою рушія буде використано розширення JavaScript, Vue.js.

1.8.1 Додаткове ПЗ до Серверної частини

Docker це інструментарій для керування ізольованими контейнерами на базі операційної системи Linux. Docker доповнює інструментарій LXC більш високорівневим API, що дозволяє керувати контейнерами на рівні ізоляції окремих процесів. Зокрема, Docker дозволяє не переймаючись вмістом контейнера запускати довільні процеси в режимі ізоляції і потім переносити і клонувати сформовані для даних процесів контейнери на інші сервери, беручи на себе всю роботу з створення, обслуговування та підтримки контейнерів.

Початковий код Docker був написаний мовою Go і поширюється під ліцензією Apache 2.0. Інструментарій базується на застосуванні вбудованих в

ядро операційної системи Linux штатних механізмів ізоляції на основі просторів імен namespaces та груп управління cgroups. Для створення контейнерів використовуються скрипти lxc. Для формування контейнера досить завантажити базовий образ оточення, команда docker pull base, після чого можна запускати в ізольованих середовищах довільні програми наприклад, для запуску з bash потрібно виконати команду docker run -i -t base/bin/bash.

Redis це розподілене сховище пар ключів та значень, що зберігаються в оперативній пам'яті, з можливістю забезпечувати довговічність зберігання на бажаннях користувача, але не довше цикла роботи комп’ютеру, після оновлення оперативної пам'яті все щ обуло записано зникне. Redis є програмним забезпечення з відкритим сирцевим кодом написане на мові програмування C. Фінансується озробка Redis компанією VMware. Серцеві тексти поширюються в рамках ліцензії BSD.

Redis надає схожі на memory cache функції для зберігання даних у форматі ключів та значень, розширені за допомогою підтримки структурованих даних, таких як хеші, списки та множини. На відміну від memory cache, Redis забезпечує постійне зберігання даних у пам’яті та гарантує збереження БД у разі аварійного завершення роботи. Клієнтські бібліотеки доступні для більшості популярних мов програмування, включаючи такі Python, Perl, Java, PHP, Tcl та Ruby.

Також у Redis є підтримка транзакцій, що дозволяють виконати за один прохід цілу групу команд, гарантуючи несуперечність і послідовність, команди від інших запитів не можуть вклинитися, виконання заданого набору команд, а у разі виникнення проблем дозволяє відкотити всі зміни. Всі дані у повному обсязі зберігаються в оперативній пам'яті. Збереження всіх даних в оперативній пам'яті дозволяє досягти значної продуктивності. При тестуванні Redis на

сервері з CPU Xeon X332 2.4 ГГц вдалося забезпечити 110000 операцій запису і 81000 операцій читання за секунду.

Для управління даними підтримуються такі команди, як інкремент та декремент, стандартні операції над множинами та списками перетин, об'єднання, та перейменування ключів, множинні вибірки та функції сортування. Також підтримується два режими зберігання, це періодична синхронізація даних на диск та ведення на диску логінізації змін. В іншому випадку гарантується повне збереження всіх виконаних змін. Можлива організація master-slave реплікації даних на кілька серверів, що здійснюється в неблокуючому режимі. Доступний також режим обміну повідомленнями публікація та підписка, при якому створюється канал, повідомлення з якого поширюються клієнтам, що підписані на канал.

PostgreSQL — об'єктно-реляційна система управління базами даних СКБД. Є альтернативою як комерційним СУБД Microsoft SQL Server, Oracle Database, IBM DB2 та інші, так і СУБД з відкритим кодом Firebird, SQLite та MySQL.

Порівняно з іншими проєктами з відкритим кодом, такими як MySQL, FreeBSD або Apache, PostgreSQL не контролюється якоюсь однією компанією, її розробка можлива завдяки співпраці багатьох людей та компаній, які прагнуть використовувати цю СУБД та впроваджувати в неї найновіші досягнення в сфері розробки баз даних.

Ядро серверу PostgreSQL було написано на мові C. Зазвичай розповсюджується у вигляді набору текстових файлів із серцевим кодом. Для того, щоб встановити сервер бази даних Postgresql необхідно відкомпілювати файли на комп'ютері та скопіювати в окремий каталог. Весь процес детально описаний у документації.

PostgreSQL - вільна об'єктно-реляційна система управління базами даних СУБД.

PostgreSQL створена на основі некомерційної СУБД Postgres, розробленої як open-source проєкт у Каліфорнійському університеті в Берклі. До розробки Postgres, що почалася в 1986 році, мав безпосереднє відношення Майкл Стоунбрейкер, керівник більш раннього проєкту Ingres, на той момент вже придбаного компанією Computer Associates. Назва розшифровувалося як Post Ingres, і при створенні Postgres були застосовані багато ранніх напрацювань.

Стоунбрейкер та його студенти розробляли нову СУБД протягом восьми років із 1986 по 1994 роки. За цей період в синтаксис були введені процедури, правила, типи користувача та інші компоненти. У 1995 році розробка знову розділилася. Стоунбрейкер використав отриманий досвід у створенні комерційної СУБД Illustra, що просувається його власною однойменною компанією яка згодом була придбана компанією Informix, а його студенти розробили нову версію Postgres - Postgres95, в якій мова запитів POSTQUEL це спадщина компанії Ingres на SQL.

PostgreSQL як сервер для баз даних існує в реалізації для багатьох UNIX-подібних платформ, включаючи AIX, HP-UX, macOS, IRIX, різні BSD-системи, Linux, Solaris та OpenSolaris, QNX, Tru64, а також для Microsoft Windows.

PostgreSQL базується на мові SQL і підтримує багато можливостей SQL стандарту 2011 року.

У PostgreSQL версії 14 є такі обмеження

Таблиця 1.1 – Обмеження бази даних Postgresql 14 версії

Характеристика	Зачення
Максимальний розмір бази даних	Немає обмежень
Максимальний розмір таблиці	32 Тбайт

Характеристика	Зачення
Максимальний розмір поля	1 Гбайт
Максимум записів у таблиці	Обмежено розмірами таблиці
Максимум полів у записі	250-1600, залежно від типів полів
Максимум індексів у таблиці	Немає обмежень

Сильними сторонами PostgreSQL вважаються:

- високоефективні та надійні механізми транзакцій та реплікації;
- система вбудованих мов програмування, що розширюється: у стандартній поставці підтримуються PL-pgSQL, PL-Python, PL-Perl і PL-Tcl. Додатково можна використовувати PL-Py, PL-Java, PL-PHP, PL-Scheme, PL-R, PL-Ruby, PL-V8 та PL-sh, а також є підтримка завантаження модулів розширення мовою C;
- успадкування;
- можливість індексування геометричних зокрема, географічних об'єктів та наявність базованого на ній розширення PostGIS;
- вбудована підтримка слабоструктурованих даних у форматі JSON з можливістю їхньої індексації;
- розширюваність, є можливість створювати нові типи даних, нові типи індексів, мови програмування, модулі розширення, також підключати будь-які зовнішні джерела даних.

Висновок до розділу 1

У першому розділі було проведено аналіз предметної області необхідної для реалізації проєкту. Визначено переваги та недоліки вибраних технічних засобів розробки, проведено аналіз комп'ютерного забезпечення, архітектуру серверного додатка, аналіз робочого проєкту та можливих інструментів для роботи з фреймоворками.

Під час дослідження було зроблено висновки щодо необхідного комп’ютерного забезпечення та очікувань від роботи з проєктом який буде написаний за допомогою Python фреймворку Django та JavaScript фреймворку Vue.js :

- Переважним плюсом Django є швидкість виконання поставленої задачі, вбудовані класи, вбудована панель адміністрування, робота з кешуванням, веб-сокети та ще дуже багато чого вбудованого всередині.
- Перевагами Vue.js це його новизна на ринку, максимально добра документація та його синтаксис.
- Основним апаратним забезпеченням для роботи з серверами Django та Vue.js є процесор (CPU) та оперативна пам’ять (RAM).
- У реалізації проєкту важливо звернути увагу що серверна продуктивність повністю залежить від апаратного забезпечення, яке встановлено на хостінгу, а клієнтська залежить від апаратного забезпечення, яке встановлено у користувача.

Виходячи з цього аналізу програмного забезпечення описаному у даному розділі було обрано наступний стек тех3нологій: мову програмування Python, фреймворк Django для серверної частини, мову JavaScript, фреймворк Vue.js для клієнтської частини.

2. АПАРАТНЕ ЗАБЕЗПЕЧЕННЯ ТА АЛГОРИТМИ ПОТОКОВОЇ АУДІОПЕРЕДАЧІ

2.1 Апаратна частина хостінг Heroku

Heroku це хмарна PaaS-платформа, яка підтримує великий ряд мов програмування. Компанія яка володіє правами на Heroku називається Salesforce.com та це є офіційний сайт компаніх. Платформа Heroku, одна з перших хмарних платформ, вона з'явилась в червні 2007 року і спочатку мала підтримку лише однієї мови програмування і це була мова програмування Ruby, але зараз список підтримуваних мов також містить в собі Python, Java, Golang, Node.js, Clojure, Scala та PHP. На серверах хмарної платформи Heroku використовуються операційні системи Debian або Ubuntu яка в свою чергу також заснована на основі Debian.

Джеймс Лінденбаум, Оріон Генрі та Адам Віґґінс заснували проєкт Heroku ще в 2007 року як підтримку для проєктів, які були засновані на інтерфейсі Rack web server. Компанія Salesforce.com викупила Heroku 8 грудня 2010 року, зробивши її своєю дочірньою компанією. Мацумото Юкіхіро, який був творцем мови програмування Ruby, прийшов в компанію 12 липня 2011 року одразу на посаду головного інженера. У тому ж місяці Heroku розробила та впровадила підтримку Node.js та Clojure.

Платформа Heroku та компанія Facebook представили нову опцію Heroku для Facebook 15 вересня 2011 року. Heroku також надає підтримку управління базами даних, наприклад таких систем як MongoDB, Membase, CouchDB та Redis, крім основної системи PostgreSQL.

Програми, що працюють на Heroku, використовують також DNS сервери Heroku зазвичай додатки мають доменне ім'я наприклад такого виду

ім'я_додатку.herokuapp.com. Для кожного проєкту на платформі Heroku виділяється кілька незалежних віртуальних процесів, які називаються «dynos». Вони розподілені по спеціальній віртуальній сітці «dynos grid», яка складається з декількох серверів. Heroku також має систему контролю версій Git, але наприклад на даний момент вона не працює, так як зараз йде розгляд справи злиттям даних користувачів Heroku які використовували github.

У червневні 2012 року через сильний шторм в Північній Америці майже всі додатки, що працювали на Heroku, відключилися, але доступ був відновлений менш ніж через 1 добу.

Зазвичай, програми працюють на виділеному сервері, а для сайтів використовують хостинги. Але можливості хостингів обмежені. А виділені сервери, такі як VPS, потрібно налаштовувати: самостійно визначати архітектуру, збирати програму, піклуватися про безпеку. Витрачати на це ресурси не завжди можливо.

У разі використовується Heroku. Платформа дозволяє завантажувати будь-яку програму та не займатися настроюванням серверної частини.

Heroku - Platform as a Service. Це означає, що платформа працює як сервіс: надає користувачеві певні функції та можливості, доступ до систем та ПЗ. При цьому її інфраструктура повністю прихована.

За користувача все роблять співробітники сервісу - ця робота залишається "під капотом", а багато процесів автоматизовані. За безпеку, архітектуру та налаштування сервера відповідають спеціалісти платформи.

Heroku потрібен:

- для розміщення застосунків та веб-сервісів;
- спрощення та прискорення циклу розробки;
- зниження потреби у складній роботі із сервером;
- роботи з навантаженими програмами;

- швидкого масштабування проектів.

Є нюанси. На ціну впливає кількість ресурсів, які використовує клієнт. Тому Heroku не завжди підходить для хайлоад-проектів: забезпечувати роботу сервера може бути дешевшим, ніж використовувати платформу.

Платформа Heroku працює за допомогою диносів. Диноси це додатки, що працюють в Heroku, виконуються ізольовано від інших — вони укладені в спеціальні контейнери, які називаються диносами або дино (dyno, dynos). Диноси дозволяють створити легковажне незалежне середовище і розгорнути у ньому додаток те щоб налаштування його середовища конфліктували коїться з іншими. Одна програма може використовуватися декількома диносами, і проект легко масштабується під завдання розробника.

Типи процесів. Диноси мають шаблони — прототипи, на основі яких створюється контейнер, як деталь по кресленню. Саме завдяки їм програми в Heroku легко масштабувати.

- Кожен тип процесу відповідає за свою частину роботи і не торкається інших модулів. Це допомагає паралелізму: процеси поділяються і завдання не поєднуються. Так можна уникнути конфліктів.
- Дінос легко масштабувати. Якщо програма потребує більше ресурсів, можна збільшити робочі потужності в кілька кліків. Для цього потрібно додати необхідну кількість нових диносів з такими ж типами процесів, як у тих, що використовуються до того.

Особливості Heroku. Мультимовність. Heroku підтримує Ruby, Python, PHP, Node.js, Java, Go, Scala та Clojure. Спочатку платформа створювалася для роботи з Ruby on Rails, тому в старій документації часто зустрічається згадка

цієї зв'язки. Сама платформа Heroku працює на Debian та Ubuntu, дистрибутивах Linux.

Швидке розгортання та легке масштабування. Щоб додати, розгорнути та запустити програму, достатньо ввести кілька команд у консолі. Тривала підготовка та попереднє налаштування не потрібні. Працювати з сервісом може фахівець-початківець. Також використання Heroku заощаджує час розробника при запуску та масштабуванні нового проекту. Збільшити кількість диносів можна за допомогою однієї команди консолі.

Додаткові можливості. Серед проектів Heroku — власна СУБД SQL database as a service, програмне забезпечення для зв'язку команди розробників між собою, сервіси автоматизації для програм різними мовами та багато іншого. Платформа працює і з noSQL-рішеннями. Інструменти можна користуватися разом з основним хмарним сервісом.

Інтеграція із сервісами. Heroku "з коробки" підтримує Docker та Git. Вони доступні навіть у базових тарифах. Якщо програмісту не вистачає вбудованих можливостей та власних проектів Heroku, він може скористатися додатковими модулями, які відкривають доступ до стороннього ПЗ.

Безкоштовний доступ до невеликих проектів. Heroku є початковий тариф Free. Він безкоштовно дає користувачеві 550-1000 годин роботи диносів на місяць. У тарифі доступні два типи процесів і можливість додавати власні домени. Через 30 хвилин без активності сервіс «засинає»: цього можна уникнути при виборі іншого базового тарифу.

Детальна документація. На сайті Heroku Dev Center доступні покрокові посібники та туторіали. Вони детально описані особливості роботи з Heroku для застосунків на всіх підтримуваних мовах. Офіційна документація доступна англійською та японською.

На власному досвіді можу сказати, що хмарна PaaS-платформа Heroku працює на дуже високому рівні, але інколи мабудь раз в пів року стаються якісь не заплановані технічні роботи. Якщо ж брати за весь час роботи з цією платформою, то з упевненостю можно сказати, що Heroku – це найкращий вибір для розробки та тестування роботи застосунків не на локальній машині. Ще одним значним плюсом платформи Heroku є те, що вона має декілько планів підписки, для цього проєкту було обрано бехкоштовну підписку для того щоб продемонструвати демо версію. В безкоштовнії версії надається можливість деплоїти з проєкт з github, операційну система Ubuntu 20 версії, запис усіх логів, які допогають у розробці, завдяки ним можно швидко визначити де ховається помилка, є дозвіл на встановлення власного домену, також 512 мегабайтів оперативної пам’яті та 2-х ядерний процесор. В майбутньому якщо буде прийнятно рішення продовжувати використовувати цю платформу потрібно будет вибрати план за 25 або 100 доларів, який надає 16+ гігабайтів оперативної пам’яті та процесор який буде мати 10 або більше потоків, щоб було можливо запускати Django сервер за допомогою gunicorn або використовувати docker, kubernetes, щоб запускати 10 або більше клонів проєкту серверної частини, що дасть приріст запитів на секунду, яких Django зможе обробити.

Також платформа Heroku була обрана тому що в ній можно створити не лише проєкт та запустити його, так само можно створити примірник бази даних, в цьому проєкті буде використовуватись база даних Postgresql 14 версії. Її можно створити у вкладці ресурси в проєкті Heroku. В безкоштовній версії база даних витримує 10 тисяч записів, для демо версії цього проєкту цього повино вистачити. Також у майбутньому планується використовувати Redis, його також можно додати у вкладці ресурси, але перед цим вже Heroku попросить прив’язати банківську карту.

Отже щоб перенести 2 проєкта на платформу Heroku потрібно створити 2 проєкта та налагодити. Перший проєкт створюється для серверної частини, потрібно вказати яка мова програмування буде використовуватись було вказано, що Python, також потрібно вибрати локацію хостінгу, було обрано Європу для того щоб уникнути проблем із затримкою відповіді між клієнтською та серверною частинами. Для клієнтської частини проєкту потрібно точно також створити проєк та також вказати локацію хостінгу Європу. Процес перенесення проєктів на платформу Heroku буде описано в 3 розділі дипломної роботи.

2.2 Апаратна частина Google Cloud Platform

Google Cloud Platform це набір хмарних служб які надаються компанією Google, вони виконуються на тій ж самій інфраструктурі, яка використовується в Google для їхніх продуктів, які призначені для кінцевих споживачів, серед них є такі продукти як YouTube та Google Search. Крім інструментів для управління, так само надається великий модульний ряд служб хмар, серед них є такі як зберігання даних, аналіз даних, хмарні обчислення та машинне навчання. Для того щоб зареєструватись необхідно прив’язати банківську картку чи банківський рахунок.

Google Cloud Platform надає такі послуги, платформа як послуга, безсерверні обчислення та як інфраструктура як послуга.

У 2008 року у квітні Google анонсувала App Engine. App Engine це платформа для розробки та хостингу веб-додатків які зберігаються безпосередньо у дата-центрах компанії Google. Це справді був перший хмарний сервіс, який представлений компанією. Для суспільства цей сервіс став доступним у грудні 2011 року. З того моменту як було анонсовано App Engine компанія Google встигла додати багато хмарних служб до цієї платформи.

Google Cloud Platform це сервіс який є частиною Google Cloud, який так само включає в себе корпоративні версії Android та Chrome OS, G Suite, Google Maps та API для машинного навчання.

У січні 2019 року Google випустила чотири нові програми сертифікації для хмарних інженерів та розробників, так програми як: Professional Cloud Network Engineer, Professional Cloud Security Engineer, Professional Cloud Developer, та G Suite. Пройти навчання можна на платформі Coursera та інших партнерів компанії

Для цього проєкту також знадобиться один із сервісів які надаються Google Cloud Platform, а саме буде потрібно Google bucket який є одним з сервісів Google Cloud Storage.

Google Cloud Storage — це веб-сервіс для зберігання файлів у режимі REST, який забезпечує зберігання та доступ до даних в інфраструктурі Google Cloud Platform. Послуга поєднує продуктивність і масштабованість хмари Google з розширеними можливостями безпеки та спільного доступу. Це інфраструктура як послуга IaaS, порівнянна з Amazon S3. На відміну від Google Drive і відповідно до різних специфікацій сервісу, Google Cloud Storage більше підходить для підприємств.

Активація користувача здійснюється через консоль розробника API. Власники облікових записів Google повинні спочатку отримати доступ до служби, увійшовши в систему, а потім погоджуючись з Умовами надання послуг, а потім увімкнувши структуру виставлення рахунків. Google Cloud Storage зберігає об’єкти (спочатку обмежені 100 ГіБ, наразі – до 5 ТіБ) у проектах, які організовані в сегменті. Усі запити авторизуються за допомогою політик керування ідентифікацією та доступом або списків контролю доступу, пов’язаних з обліковим записом користувача чи служби. Назви сегментів і

ключі вибираються так, щоб об’єкти можна було адресувати за допомогою URL-адрес HTTP.

Google Cloud Storage пропонує чотири класи сховища, ідентичні за пропускною здатністю, затримкою та довговічністю. Чотири класи: Multi-Regional Storage, Regional Storage, Nearline Storage і Coldline Storage, відрізняються своєю ціною, мінімальною тривалістю зберігання та доступністю.

- Сумісність – Google Cloud Storage взаємодіє з іншими інструментами та бібліотеками хмарного сховища, які працюють із такими службами, як Amazon S3 та Eucalyptus Systems.
- Узгодженість – Операції завантаження в Google Cloud Storage є атомарними, забезпечуючи міцну узгодженість читання після запису для всіх операцій завантаження.
- Контроль доступу – Google Cloud Storage використовує списки контролю доступу ACL для керування доступом до об’єктів і сегментів. ACL складається з одного або кількох записів, кожен з яких надає певний дозвіл на область. Дозволи визначають, що хтось може робити з об’єктом або сегментом наприклад, читати або записувати. Області дії визначають, на кого поширюється дозвіл. Наприклад, певний користувач або група користувачів наприклад, електронні адреси облікового запису Google, домен Google Apps, публічний доступ тощо
- Поновлюване завантаження – Google Cloud Storage надає функцію відновлюваної передачі даних, яка дозволяє користувачам відновлювати операції завантаження після того, як збій зв’язку перервав потік даних.

Google bucket в цьому проєткі буде використовуватись для зберігання файлів, а саме картинок, в проєкті буде декілько типів картинок, це аватар користувача, постер плейлісту або альбому, також в можно додавати картинку до будь-якого треку. Також Google bucket буде використовуватись для зберігання самих аудіо файлів, а в базі даних буде зберігатись лише посилання на Google bucket.

2.3 Алгоритм потокової аудіопередачі

Роботи потокової аудіопередачі в цьому проєкті буде працювати по такому алгоритму. З клієнтської частини коли користувач буде тиснути на кнопу відтворити трек, буде надходити запит до серверної частини, в якому буде передаватись id ідентифікатор по якому серверна чатсина буде шукати його в базі даних, серверна частина буде обрабляти цей запит, шукаючі потрібний запис у базі даних по ідентифікатору, після того як серверна частина знайде потрібний запит, вона віддасть повний примірник з усіма даними, включаючи і посилання на пісню яку потрібно відтворити. Після того як клієнтська частина отримує посилання на аудіо файл, за допомогою інструментів які надає фреймворк Vue.js за допомогою об’єкту Blob буде завантажвуватись файл та дінамічно передаватись до html тегу audio, який в свою чергу буде поступово поступово відавати аудіо файл користувачеві. Все це можливо за допомогою Web Audio API.

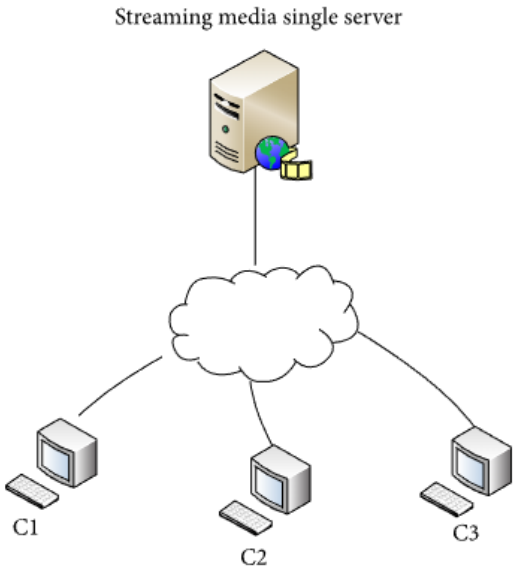

Рисунок 2.3 – модель потокової передачі даних

Веб-аудіо API – потужний та багатогранний інструмент для керування звуковою установкою на веб-сторінці, що дає можливість розробникам вибирати джерела, додавати до них спеціальні звукові ефекти (такі як панорамування), візуалізувати їх та багато іншого.

API веб-аудіо дозволяє використовувати операції над аудіо за допомогою спеціального аудіоконтексту, а також пропонує можливості використання модульної маршрутизації. Базові операції з використанням аудіовузлів, які об'єднуються разом, формуючи аудіомаршрутизатор таблицю (граф маршрутизації аудіо). Декілька джерел - з виявленням потокових схем - виявляються навіть усередині контексту. Ця модульна конструкція реалізує гнучкість у побудові складних функцій для відчуттів відчуттів.

Аудіовузли об'єднуються в ланцюги та прості мережі з входами та висновками. Вони зазвичай використовуються з використанням або більше джерелами. Джерела використовують набори семплів на одиницю часу. Наприклад, при частоті дискретизації 44100 Гц, у кожному секунді кожного

каналу розташовано 22050 семплів. Вони можуть бути або оброблені математично (дивіться OscillatorNode), або зчитуються зі звуко/відеофайлами дивіться AudioBufferSourceNode і MediaElementAudioSourceNode або з аудіо потоками дивіться MediaStreamAudioSourceNode. По факту, звукові файли - просто запис звукових коливань, що походять від мікрофона та музичних інструментів, виявляючись в одній складній хвилі. Вивідні дані цього вузла можуть бути введені іншими, які позначають або модифікують потоки звукових зразків інших потоках. Популярна модифікація - множення вихідного семпла за значенням, щоб зробити звук меншим або більш гучним (дивіться GainMode). Коли звук був успішно оброблений призначеним йому ефектом, він може бути прив'язаний до вихідного потоку (дивіться AudioContext.destination), який спрямовує звук у динаміки або мікрофон. Цей крок потрібен лише в тому випадку, якщо ви передбачаєте можливість дізнатися про ваші шедеври.

Простий, типовий порядок дій з управління аудіосистемою:

- Створимо аудіоконтекст.
- Всередині нашого контексту визначимо джерела - такі як аудіо, генератор осцилятор, потік.
- Визначимо ефекти ефектів, такі як реверберація, фільтр біквадратний, панорамування (паннер), стиснення (компресор).
- Виберемо кінцеву точку аудіосигналу, наприклад, ваші системні звукові пристрої.
- Прив'яжемо наші джерела до ефекту та ефекти до природного сигналу.

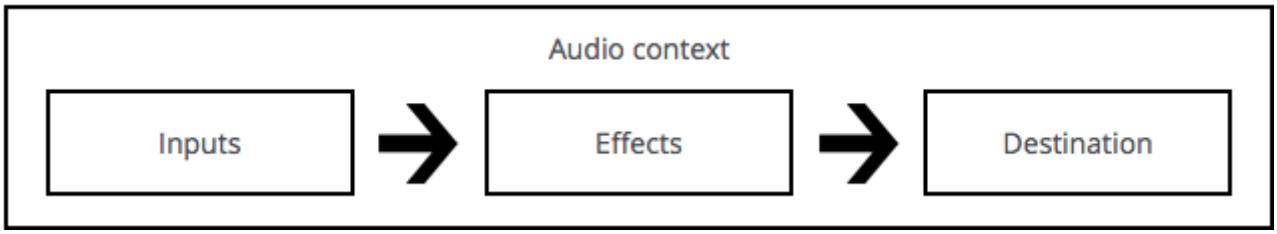

Рисунок 2.2 – Порядок дій управління потоковою аудіосистемою

Розподіл часу зустрічається з високою ймовірністю та низькою затримкою, зустрічається розробникам написання коду, що точно реагує на події та може обробити зразок навіть на високій оцінці зразків (частота дискретизації). Так що такі програми, як ритм-комп'ютер і програмний автомат завжди під рукою.

Веб-аудіо API також дає нам можливість контролювати те, яке аудіо є у приміщенні. особливу модель, що базується
на джерелі-слухачі, він дозволяє контролювати модель панорамування та обходитися без дистанційно-викликаного послаблення дистанційно-індукованого згасання або подвійного зсуву, спричиненого причиною виникнення джерела або повернення слухача.

Методи Web Audio API – які будуть використані. Насправді їх дуже багато, але розглядати всі ми не будемо, розглянемо тільки ті, які будуть задіяні в даному проєкті.

З головних контейнерів та визначень, які створюють аудіо об'єкт у Web Audio API будуть використані такі:

- AudioContext - є аудіообробним об'єктом, спорудженим з аудіомодулів, що поставляються разом, де кожен є екземпляром класу AudioNode. AudioContext дозволяє створювати вузли, що містяться у функціях обробки або декодування аудіоданих.

- AudioNode - Інтерфейс AudioNode являє собою якийсь модуль обробки, такий як джерело аудіо (тобто елемент HTML <audio> або <video>), кінцевий аудіооб'єкт, модуль захоплення обробки (наприклад, фільтр BiquadFilterNode або звуковий контролер, такий як GainNode).
- AudioParam - Інтерфейс AudioParam представляє аудіо-параметри, пов'язані з AudioNode. Він може пред'являти як точне значення параметра, і параметри, змінюються у часі.

З джерела звуку, інтерфейси, що визначають джерела звуку для використання у Web Audio API такі:

- AudioBufferSourceNode Інтерфейс AudioBufferSourceNode є джерелом звуку, що складається з аудіо даних, що зберігаються в AudioBuffer. Це AudioNode, який виступає як джерело звуку.
- MediaElementAudioSourceNode Інтерфейс MediaElementAudioSourceNode є джерелом звуку, який з'являється в елементі HTML5 <audio>або <video>. Це AudioNode, який виступає як джерело звуку.

Поділ та об'єднання аудіоканалів. Щоб розділити та об'єднати аудіоканали, буде використовуватись такий інтерфейс:

ChannelMergerNode - інтерфейс ChannelMergerNode поєднує різні монофонічні входи в один вихід. Кожен вхід використовуватиметься для заповнення каналу виведення.

Висновок до розділу 2

У другому розділі було проведено аналіз апаратного забезпечення необхідного для реалізації проєкту. Визначено переваги та недоліки вибраних апаратних платформ. Було обрано хмарну платформу Heroku як апаратну частину проєкту. Перевагами якої є зручне та швидке налаштування середовища проєкту, як для серверної так і для клієнтської частини проєкту, зручна робота з github репозиторіями, платформа Heroku має також дуже зручний термінал, який зветься dynos, також він видає одразу базовий домен, але також є можливість встановити свій, також базо версія Heroku безкоштовна, не потрібно прив’язувати ні яких банківських карток, все що потрібно зробити це зареєструватися, та створити проєкт.

Також з апаратної частини було використано хмарну платформу від команії Google під назвою Google Cloud Platform, а саме платформу Google bucket для зберігання файлів, наприклда таких як, файли картинок та музичні файли.

Було розглянуто та описано алгоритм роботи проєкту, а самє алгоритм роботи потокової аудіопередачі. Описано як і за домогою яких інструментів можливо це реалізувати.

3. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Серверна частина

3.1.1 Створення Django проєкту

Щоб почати працювати з вибраним фреймворком Django потрібно бути впевненим що інтерпретатор мови Python встановлений на комп’ютері.

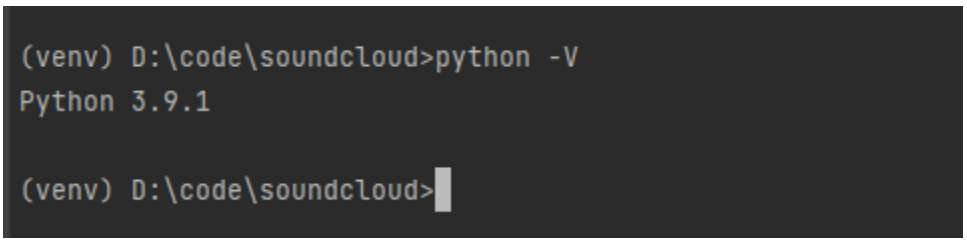

Рисунок 3.1 – Перевірка наявності інтерпретатора

Якщо інтерпертатору не має, то потрібно встановити. Для цього потрібно перейти на офіційний сайт мови програмування Python та вибрати потрібну вам версію інсталятору.

Після того як ми впевнились, що маємо встановлений Python ми можемо створювати проєкт на основі фреймворку Django. Для цього потрібно встановити фреймоворк Django, це робиться за допомогою команди pip install Django, якщо ви хочете встановити найновішу версію, якщо вам потрібна якась конкретна версія то потрібно вести команду pip install Django==3.2.1 та після слова Django вписати 2 дорівнює та вказати потрібну версію. Для мого проєкту буде використовуватись Django останьої версіі, тобто Django 4.0.5.

```
(venv) D:\code\pythonProject3>pip install Django
Collecting Django
  Downloading Django-4.0.5-py3-none-any.whl (8.0 MB)
     ---------------------------------------- 8.0/8.0 MB 5.3 MB/s eta 0:00:00
Collecting asgiref<4,>=3.4.1
  Downloading asgiref-3.5.2-py3-none-any.whl (22 kB)
Collecting tzdata
  Downloading tzdata-2022.1-py2.py3-none-any.whl (339 kB)
     ---------------------------------------- 339.5/339.5 kB 7.0 MB/s eta 0:00:00
Collecting sqlparse>=0.2.2
  Using cached sqlparse-0.4.2-py3-none-any.whl (42 kB)
Installing collected packages: tzdata, sqlparse, asgiref, Django
```

Рисунок 3.2 – Встановлення фреймворку Django

Для того щоб створити новий проек на Django потрібно вписати таку команду у терміналі – django-admin startproject <name> де name це назва проєкту. Після того як ми створили проєкт нам потрібно

```
(venv) D:\code\pythonProject3>django-admin startproject snuff
```

Рисунок 3.3– Створення проєкту

Після виконання цієї команди директорія має такий вигляд:

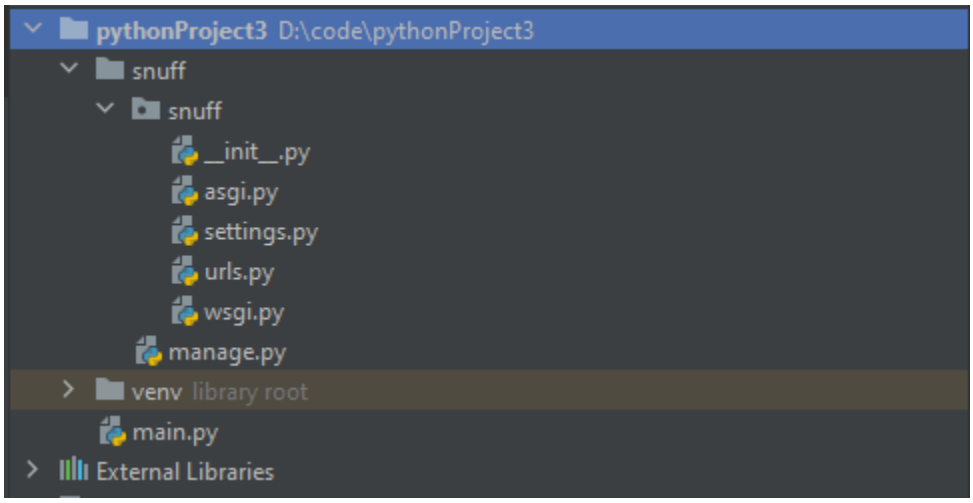

Рисунок 3.4 – Вигляд директорії після створення проєкту

Після створення проєкту потрібно створити міграції за допомогою команду python manage.py migrate. За допомогою цієї команди ми створимо таблиці в базі даних SqLite3 які потрібні Django за замовченням.

```
(venv) D:\code\pythonProject3\snuff>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
```

Рисунок 3.5 – Створення таблиць у базі даних SqLite3

Також створемо профіль адміністратора щоб була можливість користуватись вже готовою панеллю адміністратора Django. Використання цієї адміністративної паналі дуже допомагає у розробці.

Тому що тобі не потрібно йти і шукати щось в базі даних, за допомогою адміністративної панелі можно створити, видалити, та модернізувати будь який запис, що знаходиться в базі даних. Для того щоб створити профіль адміністратора, в Django це називають superuser, нам потрібно вести до терміналу таку команду – python manage.py createsuperuser. Далі з’явиться інтерактивне меню в терміналі, де потрібно буде вести свій логін, та пароль, почта не обов’язкова.

```
(venv) D:\code\pythonProject3\snuff>python manage.py createsuperuser
Username (leave blank to use 'hather'):
Email address:
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

Рисунок 3.6 – Створення профіля адміністратора

Після цього ми можемо запустити наш сервер на основі Django. Це робиться командою python manage.py runserver, що за замовченням запустить сервер на 8000 порті, але якщо є бажання запустити сервер на іншому, то це можно зроботи дописавши порт після runserver – python manage.py runserver 5000

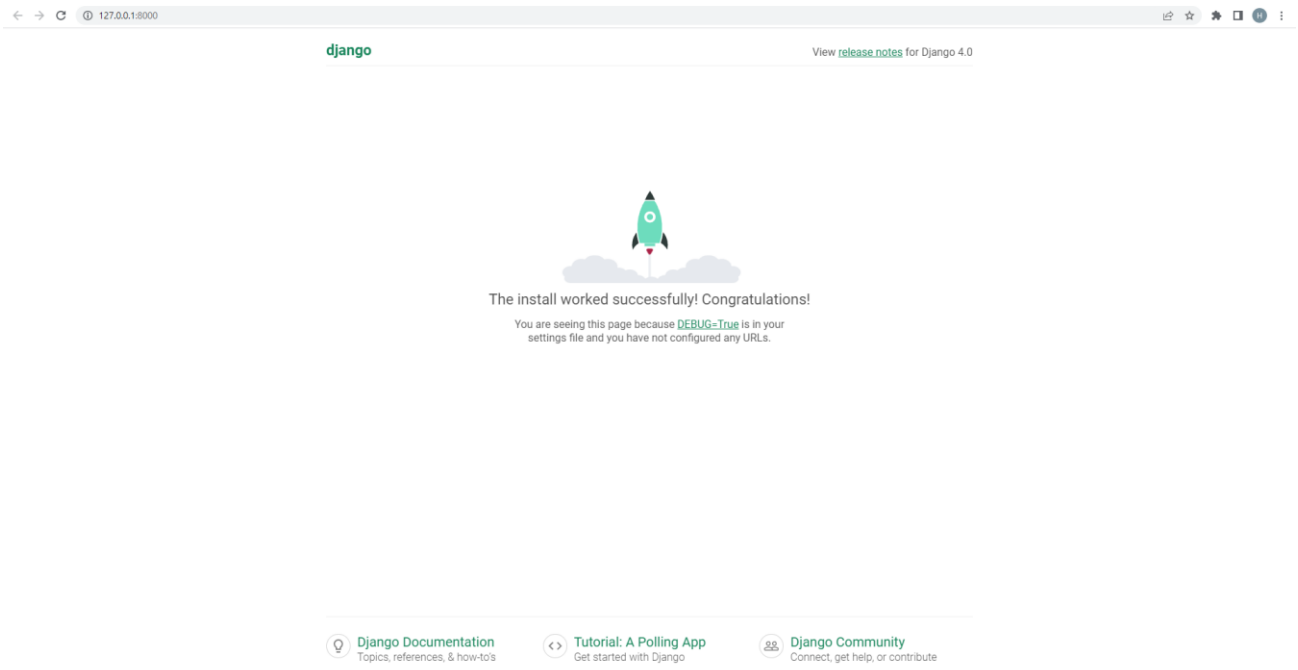

Рисунок 3.7 – Перший запуск серверу на основі Django

3.1.2 Налаштування Django проєкту

Далі потрібно підготувати проєкт для подальшої розробки. Потрібно розділити логіку на окремі мікро-сервіси. Дуже добре що фреймворк Django підтримує мікро-сервісну архітектуру. Для того щоб розділити наші сервіси потрібно створити окремі підпрограми, це робиться за допомогою команди python manage.py startapp <name> - де name це назва підпрограми. Для цього проєкту буде потрібно 2 мікросервіси це підпрограма для користувача та підпрограма для музичної бібліотеки.

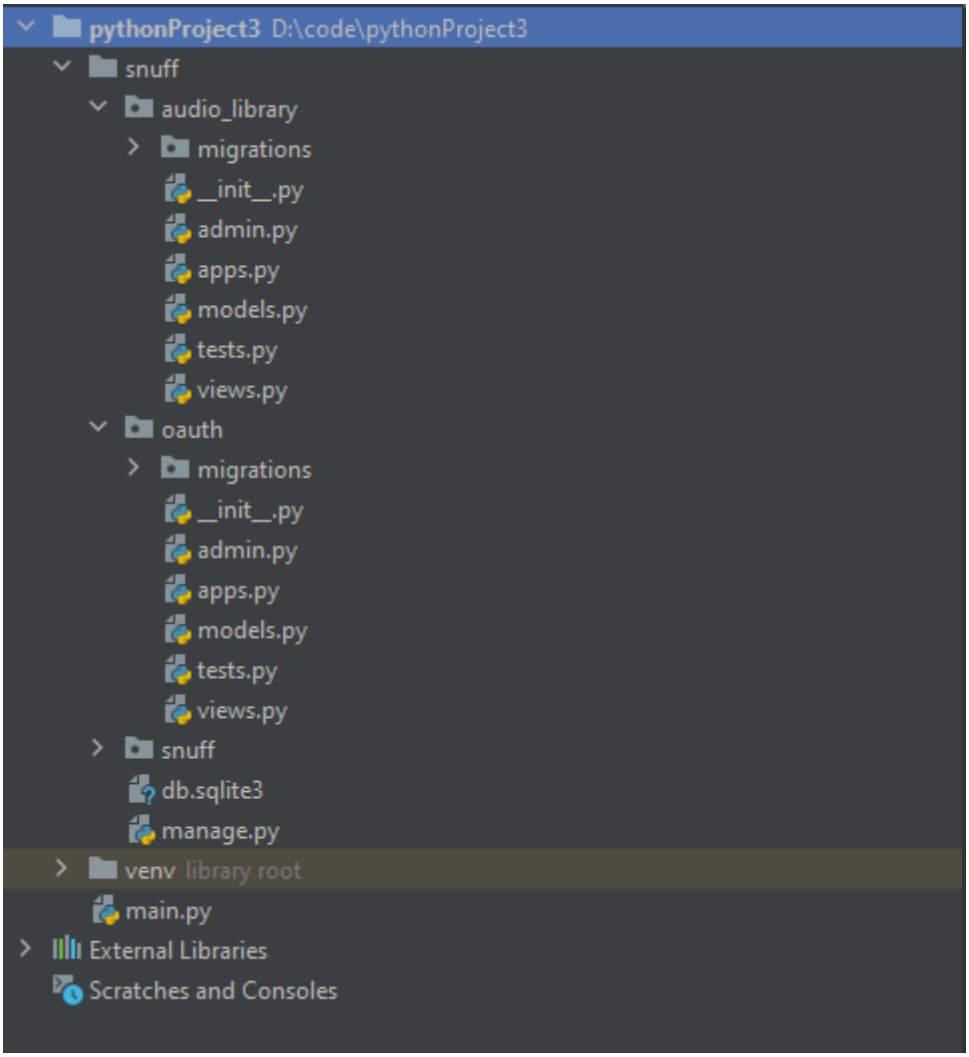

Рисунок 3.8 – Вигляд дерикторії після розділення на мікро-сервіси

Після того як ми розподілили проєкт на мікро-сервіси нам потрібно повідомити про це Django. Для цього потрібно перейти до файлу settings.py у кореневої папки проєкту. Та додати створені підпрограми у склад всіх підпрограм. Їх потрібно додати до списку INSTALLED_APPS.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'oauth',
    'audio_library'
]
```

Рисунок 3.9 – Додавання створених мікро-сервісів до списку всіх програм

Далі у цьому ж файлі потрібно змінити параметри бази даних. За замовченням Django використовує базу даних SqLite3, але це не підходить, в цьому проєкті буде використовуватись база даних Postgresql 14 версії.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': getenv('DB_NAME'),
        'HOST': getenv('DB_HOST'),
        'PORT': getenv('DB_PORT'),
        'USER': getenv('DB_USER'),
        'PASSWORD': getenv('DB_PASSWORD'),
        # 'OPTIONS': {'sslmode': 'require'},
        'TEST': {
            # 'NAME': getenv('DB_NAME'),
            'NAME': 'tests',
        },
    },
}
```

Рисунок 3.10 – Параметри для бази даних Postgresql

Для того щоб безпечно зберігати дані від бази даних, пошти потрібно створити файл .env. В цьому файлі будуть зберігатись всі потрібні дані для проєкту в середовищі нашого проєкту.

Далі потрібно встановити всі додаткові бібліотеки які знадобляться для проєкту. Такі як:

- djangorestframework – для того щоб писати Rest API які будуть взаємодіяти з клієнтською частиною.
- drf-yasg – щоб отримати автоматичне документування кінцевих точок Rest API.
- Pillow – для роботи з зображеннями.
- PyJWT – для стоверення jwt токену, який буде використовуватись для авторизації користувача.
- djoser – для полегшення роботи з jwt токенами.
- psycopg2 – для підлкючення та взаємодії з базою даних Postgresql, якщо використовується операційна система яка побудована на ядрі Linux, потрібно додати приставку –binary.

В цей список потрапили лише основні бібліотеки, ще більше 25-30 бібліотек які не потрапили, але всі вони будуть представлені у додатку А.

Для того щоб Django почав взаємодіяти з усіма цими бібліотеками їх потрібно налаштувати у кореневому файлі проєкту settings.py. Так як там дуже багато налаштувань, весь пакет налаштувань буде представлений у додатку А.

Після налаштувань у файлі settings.py потрібно ще раз зробити міграцію, так як була замінена база даних з SqLite3 на Postgresql.

На останок потрібно в кожному з мікро-сервісів створити файли urls.py для роботи з url та файл serializers.py для серіалізування даних, тобто для валідації та перетворення формату даних з пітонівського до формату json.

3.1.3 Розробка проєкту

Розробка проєкту почалась з написання моделей для кожної з таблиць. В Django робити це дуже просто за допомогою Django orm. Для цього створюється клас таблиці і описується кожне поле в цій таблиці.

Для моделі користувача було потрібно перевизначити існуючу модель користувача Django який працює за замовченням. Це робиться для того щоб користувач міг реєструватись та проходити автентифікацію за допомогою пошти як логіну.

```python
class CustomUserManager(BaseUserManager):
    """  Custom user model manager where email is the unique identifiers  for authentication instead of usernames.  """
    def create_user(self, email, password, **extra_fields):
        """  Create and save a User with the given email and password.  """
        if not email:
            raise ValueError('The Email must be set')
        email = self.normalize_email(email)
        user = self.model(email=email, **extra_fields)
        user.set_password(password)
        user.save()
        return user

    def create_superuser(self, email, password, **extra_fields):
        """  Create and save a SuperUser with the given email and password.  """
        extra_fields.setdefault('is_staff', True)
        extra_fields.setdefault('is_superuser', True)
        extra_fields.setdefault('is_active', True)
        if extra_fields.get('is_staff') is not True:
            raise ValueError('Superuser must have is_staff=True.')
        if extra_fields.get('is_superuser') is not True:
            raise ValueError('Superuser must have is_superuser=True.')
        return self.create_user(email, password, **extra_fields)
```

Рисунок 3.11 – Перевизначити існуючого класу користувача

Далі створюється клас для таблиці користувача, потрібно виконати успадкування від існуючого базового класу користувача, вказується що для логіну буде використовуватись пошта. Далі описується кожне з полів таблиці користувача. Для поля avatar потрібно додати валідатори, які обмежують дозволений тип розширення картинки, та валідатор який обмежує додавати картинку більше ніж 2 мегабайти.

```python
class AuthUser(AbstractBaseUser, PermissionsMixin):
    """ Model for user in platform """

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = []

    objects = CustomUserManager()

    email = models.EmailField(max_length=150, unique=True)
    password = models.CharField(max_length=150)
    join_date = models.DateTimeField(auto_now_add=True)
    country = models.CharField(max_length=30, blank=True, null=True)
    city = models.CharField(max_length=30, blank=True, null=True)
    bio = models.TextField(max_length=2000, blank=True, null=True)
    display_name = models.CharField(max_length=30, blank=True, null=True)
    avatar = models.ImageField(
        upload_to=get_path_upload_avatar,
        blank=True,
        null=True,
        validators=[FileExtensionValidator(allowed_extensions=['jpg']), validate_size_image]
    )

    is_active = models.BooleanField(default=False)
    is_superuser = models.BooleanField(default=False)
    is_staff = models.BooleanField(default=False)

    class Meta:
        verbose_name = 'Профіль'
        verbose_name_plural = 'Профілі'

    @property
    def is_authenticated(self):
        """ It always return True. It's a way to know that user is authenticated """
        return True
```

Рисунок 3.12 – ORM модель для таблиці користувача

```python
def validate_size_image(file_obj):
    """
        Перевірка розміру фала
    """
    megabyte_limit = 2
    if file_obj.size > megabyte_limit * 1024 * 1024:
        raise ValidationError(f"Максимальный размер файла {megabyte_limit}MB")
```

Рисунок 3.14 – індивідуальний валідатор, який перевіряє розмір файлу.

Модель таблиці треків тобто пісень. Тут також є поля з файлами де також використовуються валідатори розширення та розміру файлів. З цікавого тут є прив’язка до моделі користувача цілих 2 рази. Перший раз це прив’вязка до поля user один до багатьох через ForeignKey, який відображає до якого конкретного користувача відноситься ця пісня та у другому випадку це прив’язка до користувача від кого було отримано лайк тобто кому сподобалась

пісня, ця прив’язка реалізована за допомогою ManyToMany тобто прив’язка багато до багато. Також тут представлені прив’язки до інших таблиць.

```python
class Track(models.Model):
    """ Модель треков
    """
    user = models.ForeignKey(AuthUser, on_delete=models.CASCADE, related_name='tracks')
    title = models.CharField(max_length=100)
    license = models.ForeignKey(License, on_delete=models.PROTECT, related_name='license_tracks', null=True, blank=True)
    genre = models.ManyToManyField(Genre, related_name='track_genres')
    album = models.ForeignKey(Album, on_delete=models.SET_NULL, blank=True, null=True)
    link_of_author = models.CharField(max_length=500, blank=True, null=True)
    file = models.FileField(
        upload_to=get_path_upload_track,
        validators=[FileExtensionValidator(allowed_extensions=['mp3', 'wav'])]
    )
    create_at = models.DateTimeField(auto_now_add=True)
    plays_count = models.PositiveIntegerField(default=0)
    download = models.PositiveIntegerField(default=0)
    likes_count = models.PositiveIntegerField(default=0)
    user_of_likes = models.ManyToManyField(AuthUser, related_name='likes_of_tracks', blank=True)
    private = models.BooleanField(default=False)
    cover = models.ImageField(
        upload_to=get_path_upload_cover_track,
        blank=True,
        null=True,
        validators=[FileExtensionValidator(allowed_extensions=['jpg']), validate_size_image]
    )

    is_cover = models.BooleanField(default=False)
    is_remix = models.BooleanField(default=False)

    cover_on_music = models.CharField(max_length=255, null=True, blank=True)
    remix_on_music = models.CharField(max_length=255, null=True, blank=True)

    def __str__(self):
        return f'{self.user} - {self.title}'
```

Рисунок 3.15 – ORM модель таблиці Track

В третьому розділі було показано лише 2 головні моделі. Також в проєкті існують такі моделі як:

- Follower – це модель таблиці в якій зберігаються дані про підписників.
- SocialLink - – це модель таблиці в якій зберігаються посилання на соціальні аккаунти певного користувача.
- Genre – це модель таблиці в якій зберігаються дані з назвами жанрів.
- Album – це модель таблиці в якій зберігаються дані про створені користувачем альбоми
- Comment – це модель таблиці в якій зберігаються дані відгуки користувачів на певну пісню
- PlayList – це модель таблиці в якій зберігаються дані про список пісень користувача.

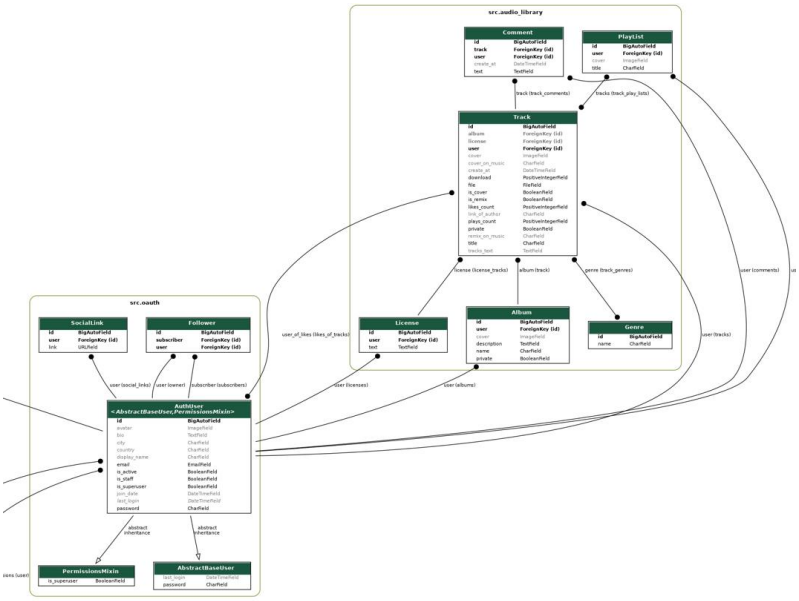

Рисунок 3.16 – часткова діаграма класів моделі

Всі моделі які не були представлені у 3 розділі будуть представлені у додатку А.

Після того як були створені моделі для таблиць даних, можно починати будувати логіку проєкту. Вся логіка буде зберігатись у файлах views.py. Це добре відомі контролери, але в джанго вони називаються views, тобто вигляд.

За допомогою бібліотеки djoser ми вже маємо кінцеві точки які працють з користувачем. За домогою цих кінцевих точок користувач може створити аккаунт, переглянути його, змінити, видалити. Також djoser надає право кастомізувати свою логіку, але в цьому проєкті вистачить логіки яка працює за замовчуванням.

Контролер для моделі Track зроблений за допомогою базового класу Django Rest Framework який називається ViewSet. За замовчуванням він надає нам усі потрібні кінцеві точки, якщо буде потрібно їх можно переробити.

```python
class TrackView(MixedSerializer, viewsets.ModelViewSet):
    """ CRUD треков
    """
    parser_classes = (parsers.MultiPartParser,)
    permission_classes = [IsAuthor]
    pagination_class = Pagination
    filter_backends = [DjangoFilterBackend]
    filterset_fields = ['name', 'album', 'genre']
    serializer_class = serializer.CreateAuthorTrackSerializer
    serializer_classes_by_action = {
        'list': serializer.AuthorTrackSerializer
    }

    def get_queryset(self):
        return models.Track.objects.filter(user=self.request.user)

    def perform_create(self, serializer):
        serializer.save(user=self.request.user)

    def perform_destroy(self, instance):
        delete_old_file(instance.cover.path)
        delete_old_file(instance.file.path)
        instance.delete()
```

Рисунок 3.17 – Контролер моделі Track

Щоб реалізувати контролер для моделі Track потрібно буде вказати модель для якої пишеться цей контролер, в цьому випадку було перероблено метод get_queryset для того щоб отримувати відфільтровану інформацію для певного користувача, id якого ми отримуємо за допомогою переданого jwt токену. Так само було перероблено метод perform_create, щоб за замовчуванням зберігати інформацію для користувача якого отримуємо за допомогою jwt токену. Так як в цих кінцевих точках будуть присутні файли, нам потрібно вказати клас парсеру для файлів в даному випадку був використаний MultiPartParser. Також потрібно обмежити дозівл для редактування профилю, для цього було дадано permission клас IsAuthor, який був кастомно створений і перевіряє чи це дійсно автор цього треку, якщо так, то дає дозвіл на редактування та видалення, якщо ні, то видасть текст помилки у відповіді. Також було використоно клас пагінації який виводить по 10 треків за одну пагінацію, щоб відповідь не була перевантажена. Також тут було використано клас фильтрації, який фильтрує список треків за назвою, альбомом або жанром.

Наприкінці потрібно вказати клас serializer. Serializer в Django це такий валідатор, який перевіряє дані які отримує і відає на ідентичність з поля вказаними в базі даних. В цьому випадку було використано 2 serializer класа, для списку, в якому не потрібно відавати повну інформацію та наватажувати клієнтську частину, та для сторінки на який буде відображатись лише один трек, на якій вже потрібна повна інформація по цьому треку.

```python
class CreateAuthorTrackSerializer(BaseSerializer):
    plays_count = serializers.IntegerField(read_only=True)
    download = serializers.IntegerField(read_only=True)
    user = serializers.IntegerField(read_only=True)

    class Meta:
        model = models.Track
        fields = (
            'id',
            'title',
            'license',
            'genre',
            'album',
            'link_of_author',
            'file',
            'create_at',
            'plays_count',
            'download',
            'private',
            'cover',
            'user',
            'is_cover',
            'is_remix',
            'cover_on_music',
            'remix_on_music',
        )
```

Рисунок 3.18 – Serializer для створення треків

Для того щоб створити serializer потрібно вказати модель для якої створюється, також потрібно вказати serializer вкладених моделей. Вкладені моделі це ті, які були прив'язані за допомогою ForeignKey або ManyToMany. Далі потрібно вказати які поля слід перевіряти.

В кінці потрібно створити шлях по якому клієнтська частина буде звертатись. Це робиться у файлі urls.py.

```python
path('track/', views.TrackView.as_view({'get': 'list', 'post': 'create'})),
path('track/<int:pk>/', views.TrackView.as_view({'put': 'update', 'delete': 'destroy'})),
```

Рисунок 3.19 – Кінцева точка для контролеру треків

Для цього вказується шлях, контролер який буде обробляти запит по цьому шляху та дії які він буде обробляти в цьому випадку шлях track/ буде відпрацьовувати на запит get list та на запит post тобто створення треку. Шлях track/<int:pk>/ буде обробляти зміну та видалення. Конструкція <int:pk> вказує що потрібно передати id треку тобто цей шлях буде шукати лише конкретний запис в базі даних, int вказує на те, що параметр pk буде цілим числом.

```python
class StreamingFileView(views.APIView):
    """
       Відтворення треку
    """
    def set_play(self):
        self.track.plays_count += 1
        self.track.save()

    def get(self, request, pk):
        self.track = get_object_or_404(models.Track, id=pk, private=False)
        if os.path.exists(self.track.file.path):
            self.set_play()
            response = HttpResponse('', content_type="audio/mpeg", status=206)
            response['X-Accel-Redirect'] = f"/mp3/{self.track.file.name}"
            return response
        else:
            return Http404
```

Рисунок 3.20 – Контролер відтворення треку

Цей контролер відповідає за передачу файлу треку на клієнтську частину. Він шукає конкретний файл в базі даних і відає, якщо не знаходить, то відає 404 помилку у відповіді. Також якщо він знаходить файл, він додає до кількості прослуховувань одне прослуховування.

```python
class DownloadTrackView(views.APIView):
    """
       Зберігання треку
    """
    def set_download(self):
        self.track.download += 1
        self.track.save()

    def get(self, request, pk):
        self.track = get_object_or_404(models.Track, id=pk, private=False)
        if os.path.exists(self.track.file.path):
            self.set_download()
            response = HttpResponse('', content_type="audio/mpeg", status=206)
            response["Content-Disposition"] = f"attachment; filename={self.track.file.name}"
            response['X-Accel-Redirect'] = f"/media/{self.track.file.name}"
            return response
        else:
            return Http404
```

Рисунок 3.21 – Контролер зберігання треку

Цей контролер також шукає певний файл, якщо файл був знайдений, то у користувача почнеться загрузка цього файлу, якщо ні, то віддасть помилку 404 у відповіді. Якщо файл буде знайдений, то до кількості зберігань буде додане ще одне зберігання.

В цьому розділі не було показано всіх контролерів, serializer та кінцевих точок, всі вони були побудовані за допомогою базового класу бібліотеки Django Rest Framework та будуть наведені у додатку А.

Після закінчення розробки серверної частини було проведено тест навантаження на саму важку кінцеву точку яка містить багато вкладених моделей. Кінцева точка яка повертає трек та всю інформацію пов’язану з ним.

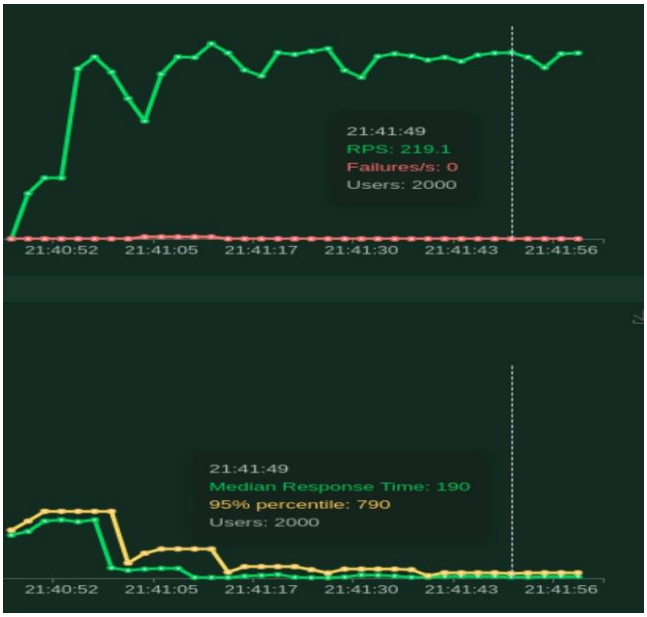

Рисунок 3.22 – тест кінцевої точки треків

З цього тесту ми бачимо, що Django може витримати 2000 одночасних запросів, час затримки всього 190 мілісекунди. Цей тест проводився одній запущеній копії серверу. Щоб отримати кращі результати требі підключати gunicorn та запускати декілька копій серверу.

3.2 Клієнтська частина

3.2.1 Створення Vue.js проєкту

На початку щоб створити проєкт Vue.js потрібно встановити Node.js. Node.js це платформа яка написана на мовах програмування C, C++ та JavaScript, яка має відкритий код. Платформа Node.js використовується для виконання високопродуктивних мережевих застосунків, які написані за допомогою мови програмування JavaScript. Засновува платформу Node.js Раян Дал. Раніше мова програмування JavaScript застосовувалась лише для обробки даних в браузері користувача, зараз за допомогою платформи Node.js який

надає можливість виконувати скрипти написані на мові JavaScript прямо на сервері та відавати результат їхнього виконання до користувача. Node.js це платформа яка перетворила JavaScript на мову загального використання, яка стала дуже популярною, вона отримала визнання та велику спільноту розробників.

Node.js має такі властивості як:

- асинхронна одно-гілкова модель виконання запитів;
- неблокуючий ввід та вивід;
- система бібліотек CommonJS;
- рушій JavaScript V8 написаний команією Google;

Для керування модулями проєкту використовується дуже зручний пакетний менеджер npm - node package manager.

Щоб встановити Node.js потрібно завантажити установник з офіційного сайту Node.js та встановити, на ланукс це можно зробити за допомогою команди sudo apt install nodejs.

Після вдалого встановлення Node.js можно встановлювати та створювати проєкт Vue.js. Для цього потрібно вписати таку команду в терміналі - vue create <name> - де name це назва проєкту. Після вводу команди з’явиться інтерактивне меню в якому можно вибрати додаткові налаштування. До додаткових налаштувань входять такі параметри:

- Вибір версії
- Встановлювати Babel
- Встановлювати TypeScript
- Встановлювати Router

- Встановлювати Vuex
- Встановлювати Pre-proccesors
- Вибір форматування
- Створювати папку для unit-тестів

Але якщо відключити всі ці налаштування, то все що не було встановлено можно буде встановити пізніше, але доведеться створювати директорії в ручну.

3.2.2 Налаштування Vue.js проєкту

Після встановлення, якщо було обрано всі налаштування для встановлення всіх компонентів, то одразу можно приступати до налаштувань, але якщо ні, то потрібно довстановити деякі бібліотеки, проблем з цим бути не може, тому що інтерпритатор завжди допоможе і підскаже які бібліотеки потрібно втсановити та навіть підскаже команди для цього, які просто потрібно буде скопіювати та вставити у термінал.

Після того як було перевіряно всі потрібні пакети, можно приступати до налаштувань проєкту. Першим ділом потрібно перейти у директорію src та знайти файл main.js це файл з головними налаштуваннями проєкту, в ньому потрібно додати деякі бібліотеки до налаштувань щоб вони почали працювати. Так виглядають налаштування мого проєкту - createApp(App).use(store).use(router).mount('#app'), де createApp – вказує назву проєкту, метод use вказує Vue.js що він буде працювати з цією бібліотекою. В цьому проєку знадобилсь лише додатковий модуль store де зберігаються всі стани незалежно в якому файлі йде робота та модуль router завдяки якому ми вказується по якому шляху який контролер буде спрацьовувати та оброблювати

цей шлях, метод mount працює з кореневим файлом html, самє цей кореневий файл буде змінюватись залежно від шляху вказаному в url користувачем.

В директорії store зберігається файл index.js в який використовується лише для того, щоб зберігати шлях до backend домену, там прописана параметр getters який і буде відавати потрібний шлях до backend домену, щоб було зручно користуватись кінцевими точками. Також store допомагає слідкувати за патерном DRY – що в переводі не повторюйся. Завдяки цьому код в цілому виглядає більш читательно.

Також в проєкті використовується модуль router він знаходиться в директорії router. В цьому файлі зберігаються всі налаштування для url шляхів. Для кожного шляху описується назва та назва компоненту тобто контролеру який буде оброблювати запити по цьому шляху. Також в цьому файлі потрібно змінити метод який відповідає за шляхи. Потрібно поставити createWebHistory замість createHashWebHistory, щоб позбавитись непотрібного знака # на початку кожного шляху після домену.

Запити до серверної частини проєкту будь відправлятись за допомогою бібліотеки axios.

3.2.3 Розробка Vue.js проєкту

Для цього проєкту знадобилось написати багато компонентів та багато самих контролерів. Компоненти в Vue.js це такі файли в яких зберігається код який можно використати декілька разів. Компоненти можно встановлювати в середину контролерів, також можно передавати файли за допомогою параметру який називається – props.

Для цього проєкту було розроблено такі контролери:

- Album.vue – контролер який обробляє шлях /albums. Він рендеріть та відображає список альбомів користувача. Також оброблює можливість зміна даних та видалення.
- AlbumCreate.vue – контролер який обробляє шлях /album/create. Він рендеріть та оброблює логіку створення альбому.
- Main.vue – контролер який обробляє шлях /. Він рендеріть та відображає список всіх треків на сайті.
- Login.vue – контролер який обробляє шлях /login. Він рендеріть та оброблює логіку реєстрації та автентифікації користувача.
- Player.vue – контролер який обробляє шлях /player. Він рендеріть та відображає список альбомів користувача та відповідає за прослуховування музики.
- Playlist.vue – контролер який обробляє шлях /playlist. Він рендеріть та відображає список плейлистів користувача. Також оброблює можливість зміна даних та видалення.
- PlaylistCreate.vue – контролер який обробляє шлях /playlist/create. Він рендеріть та оброблює логіку створення плейлістів.
- Profile.vue – контролер який обробляє шлях /me. Він рендеріть та відображає інформацію про користувача. Також оброблює можливість видалення.
- ProfileUpdate.vue – контролер який обробляє шлях /me/update. Він рендеріть та оброблює логіку зміну даних користувача.
- Tracks.vue – контролер який обробляє шлях /tracks. Він рендеріть та відображає список треків користувача. Також оброблює можливість зміна даних та видалення.
- TrackCreate.vue – контролер який обробляє шлях /track/create він рендеріть та обробляє логіку створення треків.

- Users.vue – контролер який обробляє шлях /users він рендеріть та відображає список всіх користувачів сайту та їх відкриті альбоми.

В основному всі контролери щось відображають за допомогою запитів на серверну частину, де отримують дані та щось зберігають за допомогою тих ж самих запитів на кінцеві точоки серверної частини проєкту. Код проєкту клієнтської частини буде представлений у додатку Б.

В даному проєкті було використано всі можливі запити так як:

- GET – запит на зібрання інформації з серверної частини.
- POST – запит на зберігання даних, які були отримані від користувача.
- PUT – запит на оновлення даних всіх полей в таблиці бази даних, які були отримані від користувача
- PATCH – запит на оновлення окремих полей в таблиці бази даних, які були отримані від користувача
- DELETE – запит на видалення певного запису у базі даних

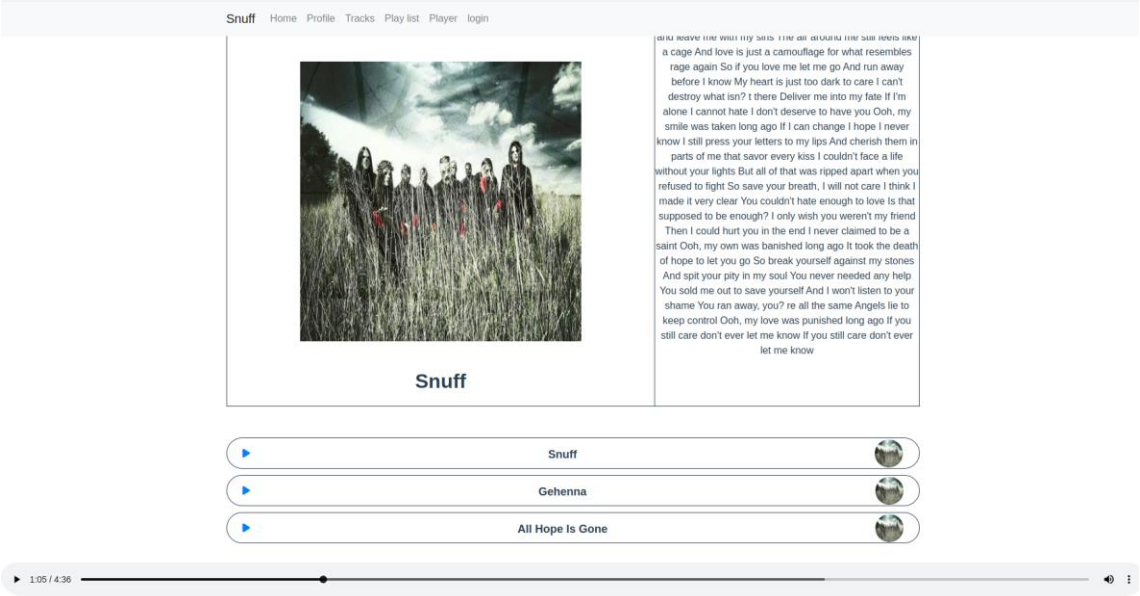

Рисунок 3.23 – вигляд сторінки потокової аудіопередачі

3.3 Налаштування для деплоя на Heroku

Для того щоб задеплоїти Python проєкт на платформу Heroku потрібно зберегти всі бібліотеки які використовуються в Django проєкті у файлі під назваю requirements.txt. Щоб це зробити потрібно ввести до терміналу таку команду pip freeze > requirements.txt. Після цього потрібно створоити у корні Django проєкту файл ProcFile без розширення та вказата в середині цього файлу налаштування, як Heroku повинен запускати Django сервер, також в цьому файлі потрібно вказати що Heroku має встановити всі бібліотеки проєкту та зробити міграції. В моєму випадку сервер буде запускатись за замовчуванням який предоставляє сам фреймворк Django. Для цього було використано такі команди - release: pip install -r requirements.txt --no-dependencies, release: python manage.py migrate, web: python manage.py runserver.

Для того щоб задеплоїти Vue.js проєкт потрібно лише прописати команду npm build. Додавати бібліотеки які були використані в проєкті не потрібно тому що в проєктах Node.js всі вони мають файл з розширеням json в якому одразу

після встановлення бібліотеки записиється назва та версія встановленої бібліотеки.

Висновки до розділу 3

У третьому розділі було повністю описано та продемонстровано всі єтапи розробки програмного забезпечення проєкту. Було показано розробку серверної та клієнтської частини проєкту від самого початку, тобто від створення проєкту до налаштувань для перенесення проєкту з локального на хмарну платформу Heroku.

В цьому розділі було представлено програмне забезпечення для проєкту. Для серверної частини проєкту було обрано фреймворку Django для мови програмування Python. Було продемонстрованно виконані класи моделей для взаємодії з базою даних за допомогою Django ORM, представленно та аргументовано логіку власноруч розроблених класів serializer, контролерів, url шляхів проєкту, які є кінцевими точками для взаємодії з клієнтською частиною. Також для цього проєкту потрібно було написати permission класи, filter класи, валідатори розмірів та розширення файлів для картинок і музичних файлів. Потрібно було налаштвувати smtp протокол у кореневому файлі з налаштуваннями для того щоб була можливість відправляти листи на пошту, вони використовуються для підтверження реєстрації на сайті, яка в свою чергу теж була реалізована. Продемонтсровано діаграму класів моделей та тестове навантаження на сервер 2000 тисячами користувачів.

Також було представлено програмне забезпечення для клінтської частини проєкту. Дя клієнтської частини було обрано фреймворк Vue.js для мови програмування JavaScript. Було продемонстровано усі єтапи розробки від створення проєкту до самого кінця, усі налаштування для проєкту, представленно та аргументовано логіку контролерів, компонентів та маршрутизаторів.

В кінці цього розділу було описано програмне налаштування проєкту для взаємодії з хмарною платформою Heroku.

ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ

В результаті виконання бакалаврської дипломної роботи було проаналізовано предметну область, а саме сервісів потокової аудіопередачі, були виявлені плюси та мінусі різних сервісів для того, щоб створити на основі цих аналізів створити власний сервіс в якому будуть враховані плюси та мінуси. Головним плюсом даного сервісу це повна взаємодія користувача з сервісом майже без обмежень. Також було додано систему маленької соціальної мережі, користувачи зможуть оцінювати та ставити свої відгуки на основі яких музичні виконавці будуть аналізувати актуальність на ринку музики.

Стосовно апаратної частини, для серверної, клієнтської частини та базу даних було розгорнуто на популярній хмарній платформі Heroku. Для цього було створено 2 проєкти на хмарній платформі Heroku, для серверної та клієнтської частини проєкту. Базу даних було створено в середині серверної частини у розділі resources. Також було створено проєкт на Google Cloud Platform, а саме bucket для зберігання самих файлів картинок та музичних файлів посилання на які будуть зберігатись у базі даних.

Що до програмного забезпечення сервісу було обрано Python Django для серверної чатсини, JavaScript Vue.js для клієнтської. Для бази даних було обрано Postgresql, також під час розробки аткивно використовувався Docker, Redis та Postman. Все що було задумано, було реалізовано.

Що до рекомендацій, сервіс вийшов працездатний, але все ж таки його потрібно модефікувати. Надалі планується додати кешування на деякі сторінки за допомогою Redis, додати фонові задачі за допомогою Celery, розробити інший алгоритм потокової аудіопередачі за допомогою веб сокетів, налаштувати запити до бази даних за допомогою select_related та

prefetch_related, які будуть кешувати вложені таблиці, також потрібно налаштувати запуск декількох копій серверу за допомогою gunicorn, також потрібно модернізувати клієнтську частину проєкту, а саме розробити кращій дизайн та на останок створити іменний домен.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Основи UML – діаграми використання (use-case) – Блог програміста [Електронний ресурс] – Режим доступу: https://pro-prof.com/archives/2594.

2. Крилов І. В. «Інформаційні технології: теорія і практика», М.: «Центр», 1996 – 167 с.

3. Кондрашова С.С. Учбовий посібник "Інформаційні технології в управлінні", К.: МАУП. 1998 – 378 с.

4. SQL – Енциклопедія мова програмування [Електронний ресурс] – Режим доступу: : http://progopedia.ru/language/sql/

5. Топ фреймворки для веб-розробки у 2018 році - JetRuby Agency [Електронний ресурс] – Режим доступу: https://jetruby.com/ru/blog/top-freimworki-2018/

6. Django Overview | Django [Електронний ресурс] – Режим доступу: https://www.djangoproject.com/start/overview/

7. Performing raw SQL queries [Електронний ресурс] – Режим доступу: https://docs.djangoproject.com/en/2.2/topics/db/sql/#django.db.models.Manager.raw

8. Django, частина 2 MTV або MVC. Моделі | VR-online – безкоштовний електронний журнал для всіх [Електронний ресурс] – Режим доступу: http://www.vr-online.ru/blog/django-chast-2-mvt-ili-mvc-modeli-8967

9. Офіційна документація до фреймворку «Django» 4.0 [Електронний ресурс] – Режим доступу: https://www.djangoproject.com/

10. Офіційна документація до фреймворку «Django REST Framework» [Електронний ресурс] – Режим доступу: https://www.django-rest-framework.org/

11. Офіційна документація до фреймворку «Vue.js» [Електронний ресурс] – Режим доступу: https://vuejs.org/guide/introduction.html

12. Офіційна документація до «Docker» [Електронний ресурс] – Режим доступу: https://docs.docker.com/

13. Офіційна документація до «Redis» [Електронний ресурс] – Режим доступу: https://redis.io/docs/

14. Офіційна документація до «Postgresql» [Електронний ресурс] – Режим доступу: https://www.postgresql.org/developer/

15. A Byte of Python. Swaroop C H, 2013р. 171 с. [Електронний ресурс] – Режим доступу: https://open.umn.edu/opentextbooks/textbooks/581

16. Designing Data-Intensive Web Applications. Elsevier, 2003р. [Електронний ресурс] – Режим доступу: https://doi.org/10.1016/b978-1-55860-843-6.x5000-2.

17. Hillar G. C. Django RESTful Web Services: The easiest way to build Python RESTful APIs and web services with Django. Packt Publishing, 2018 p. 326 c.

18. Crockford D. JavaScript: The good parts. Sebastopol, CA : O'Reilly, 2008 p. 153 c.

ДОДАТОК А
ПРОГРАМНИЙ КОД СЕРВЕРНОЇ ЧАСТИНИ

Файл settings.py – головний файл з усіма налаштуваннями проєкту:

```
import datetime
import os
from pathlib import Path

BASE_DIR = Path(__file__).resolve().parent.parent
SECRET_ getenv('DJANGO_PROJECT_KEY')
DEBUG = True
ALLOWED_HOSTS = []

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'rest_framework',
    'drf_yasg',
    'djoser',

    'src.oauth',
    'src.audio_library',
]

MIDDLEWARE = [
        'django.middleware.security.SecurityMiddleware',
        'django.contrib.sessions.middleware.SessionMiddleware',
'django.middleware.common.CommonMiddleware',
'django.middleware.csrf.CsrfViewMiddleware',
        'django.contrib.auth.middleware.AuthenticationMiddleware',
```

```
            'django.contrib.messages.middleware.MessageMiddleware',
            'django.middleware.clickjacking.XFrameOptionsMiddleware',
        ]
ROOT_URLCONF = 'config.urls'
TEMPLATES = [
            {
                'BACKEND': 'django.template.backends.django.DjangoTemplates',
                'DIRS': [os.path.join(BASE_DIR, 'templates')],
                'APP_DIRS': True,
                'OPTIONS': {
'context_processors': [
                        'django.template.context_processors.debug',
                        'django.template.context_processors.request',
                        'django.contrib.auth.context_processors.auth',
                        'django.contrib.messages.context_processors.messages',
                    ],
                },
            },
        ]

WSGI_APPLICATION = 'config.wsgi.application'

DATABASES = {
            'default': {
                'ENGINE': 'django.db.backends.postgresql',
                'NAME': getenv('DB_NAME'),
                'HOST': getenv('DB_HOST'),
                'PORT': getenv('DB_PORT'),
                'USER': getenv('DB_USER'),
                'PASSWORD': getenv('DB_PASSWORD'),
                # 'OPTIONS': {'sslmode': 'require'},
                'TEST': {
                    # 'NAME': getenv('DB_NAME'),
                    'NAME': 'tests',
                },
            },
        }

        AUTH_PASSWORD_VALIDATORS = [
```

```
        {
            'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
        },
        {
            'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
        },
        {
            'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
        },
        {
            'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
        },
    ]

    LANGUAGE_CODE = 'en-us'

    TIME_ZONE = 'UTC'

    USE_I18N = True

    USE_TZ = True



    STATIC_URL = 'static/'
    MEDIA_URL = '/media/'
    MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

    DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

    ALGORITHM = 'HS256'
    ACCESS_TOKEN_EXPIRE_MINUTES = 60 * 24

    GOOGLE_CLIENT_ID                    =                    '291011338200-
2r3jv9d0fe6j4ieb8tjf7plqp5jaibsd.apps.googleusercontent.com'
    GOOGLE_SECRET_KEY = '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'
```

```
    SPOTIFY_CLIENT_ID = 'c348ed49d711407c982f093f3dd25b49'
    SPOTIFY_SECRET_ID = 'c8bedae09ef44edfb562d898fe4b26db'

    CSRF_TRUSTED_ORIGINS = ['https://*.herokuapp.com', 'http://*localhost',
'http://localhost:3000']


    CORS_ORIGIN_ALLOW_ALL = True
    CSRF_COOKIE_NAME = "XSRF-TOKEN"
    CORS_ALLOW_HEADERS = (
        'xsrfheadername',
        'xsrfcookiename',
        'content-type',
        'XSRF-TOKEN',
        'authorization',
    )

REST_FRAMEWORK = {
        'DEFAULT_RENDERER_CLASSES': [
            'rest_framework.renderers.JSONRenderer',
            'rest_framework.renderers.BrowsableAPIRenderer',
        ],
'DEFAULT_AUTHENTICATION_CLASSES': (
                'src.oauth.services.auth_backend.AuthBackend',
                # 'rest_framework.authentication.SessionAuthentication',
                'rest_framework.authentication.TokenAuthentication',
                'rest_framework_simplejwt.authentication.JWTAuthentication',
            ),
        'DEFAULT_PERMISSION_CLASSES': (
            'rest_framework.permissions.AllowAny',
        ),
        'DEFAULT_PARSER_CLASSES': [
            'rest_framework.parsers.JSONParser',
        ],
        'DEFAULT_FILTER_BACKENDS': [
            'django_filters.rest_framework.DjangoFilterBackend'
        ],
    }DJOSER = {
        'LOGIN_FIELD': 'email',
```

```
          'PASSWORD_RESET_CONFIRM_URL':
'#/password/reset/confirm/{uid}/{token}',
          'USERNAME_RESET_CONFIRM_URL':
'#/username/reset/confirm/{uid}/{token}',
          'ACTIVATION_URL': '#/activate/{uid}/{token}',
          'SEND_ACTIVATION_EMAIL': True,
          'SERIALIZERS': {
              "user_create": "src.oauth.serializer.UserDjoserSerializer",
              "user": "src.oauth.serializer.UserSerializer",
              "current_user": "src.oauth.serializer.UserSerializer",
              "user_delete": "src.oauth.serializer.UserDjoserSerializer",
          },
'PERMISSIONS': {
              'user_list': ['rest_framework.permissions.AllowAny'],
              'user': ['rest_framework.permissions.AllowAny'],
              'set_username': ['src.base.permissions.IsAuthor'],
              'user_create': ['rest_framework.permissions.AllowAny'],
              'user_delete': ['djoser.permissions.CurrentUserOrAdmin'],
          },
          "HIDE_USERS": False
      }

EMAIL_BACKEND= 'django.core.mail.backends.smtp.EmailBackend'
      EMAIL_HOST = 'smtp.gmail.com'
      EMAIL_USE_TLS = True
      EMAIL_HOST_USER = 'ibot.supp@gmail.com'
      EMAIL_HOST_PASSWORD = '1003kv2000'
      EMAIL_PORT = 587


      SWAGGER_SETTINGS = {
          'SECURITY_DEFINITIONS': {
              'Bearer': {
                  'type': 'apiKey',
                  'name': 'Authorization',
                  'in': 'header'
              }
          }
      }
```

```
AUTH_USER_MODEL = 'oauth.AuthUser'

SIMPLE_JWT = {
          'ACCESS_TOKEN_LIFETIME': datetime.timedelta(days=7),
          'REFRESH_TOKEN_LIFETIME': datetime.timedelta(days=7),
          'AUTH_HEADER_TYPES': ('Bearer',),
          'BLACKLIST_AFTER_ROTATION': False,
          'ALGORITHM': 'HS256',
      }
```

Файл src/oauth/models.py – файл з моделями користувача:

```
from django.contrib.auth.base_user import AbstractBaseUser, BaseUserManager
from django.contrib.auth.models import PermissionsMixin
from django.core.validators import FileExtensionValidator
from django.db import models

from src.base.services import get_path_upload_avatar, validate_size_image

class CustomUserManager(BaseUserManager):
    """  Custom user model manager where email is the unique identifiers  for
authentication instead of usernames.  """
    def create_user(self, email, password, **extra_fields):
        """  Create and save a User with the given email and password.  """
        if not email:
            raise ValueError('The Email must be set')
        email = self.normalize_email(email)
        user = self.model(email=email, **extra_fields)
        user.set_password(password)
        user.save()
        return user

    def create_superuser(self, email, password, **extra_fields):
        """  Create and save a SuperUser with the given email and password.  """
        extra_fields.setdefault('is_staff', True)
        extra_fields.setdefault('is_superuser', True)
        extra_fields.setdefault('is_active', True)
```

```
        if extra_fields.get('is_staff') is not True:
            raise ValueError('Superuser must have is_staff=True.')
        if extra_fields.get('is_superuser') is not True:
            raise ValueError('Superuser must have is_superuser=True.')
        return self.create_user(email, password, **extra_fields)

class AuthUser(AbstractBaseUser, PermissionsMixin):
    """ Model for user in platform """

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = []

    objects = CustomUserManager()

    email = models.EmailField(max_length=150, unique=True)
    password = models.CharField(max_length=150)
    join_date = models.DateTimeField(auto_now_add=True)
    country = models.CharField(max_length=30, blank=True, null=True)
    city = models.CharField(max_length=30, blank=True, null=True)
    bio = models.TextField(max_length=2000, blank=True, null=True)
    display_name = models.CharField(max_length=30, blank=True, null=True)
    avatar = models.ImageField(
        upload_to=get_path_upload_avatar,
        blank=True,
        null=True,
        validators=[FileExtensionValidator(allowed_extensions=['jpg']),
validate_size_image]
    )

    is_active = models.BooleanField(default=False)
    is_superuser = models.BooleanField(default=False)
    is_staff = models.BooleanField(default=False)

    class Meta:
        verbose_name = 'Профіль'
        verbose_name_plural = 'Профілі'

    @property
    def is_authenticated(self):
```

```
        """ It always return True. It's a way to know that user is authenticated
"""
        return True

    def __str__(self):
        return self.email

class Follower(models.Model):
    """ Follower Model """
    user      =      models.ForeignKey('AuthUser',      on_delete=models.CASCADE,
related_name='owner')
    subscriber    =    models.ForeignKey('AuthUser',    on_delete=models.CASCADE,
related_name='subscribers')

    def __str__(self):
        return f'{self.subscriber} followed on {self.user}'

class SocialLink(models.Model):
    """ Model for link on user's social links """
    user      =      models.ForeignKey('AuthUser',      on_delete=models.CASCADE,
related_name='social_links')
    link = models.URLField(max_length=100)

    def __str__(self):
        return f'{self.user}'
```

Файл src/audio_library/models.py – файл з моделями аудіо підпрограми:

```
from django.core.validators import FileExtensionValidator
from django.db import models

from src.base.services import (
    validate_size_image,
    get_path_upload_cover_album,
    get_path_upload_track,
    get_path_upload_cover_playlist,
    get_path_upload_cover_track,
```

```
)
from src.oauth.models import AuthUser

class License(models.Model):
    """ Модель лицензий треков пользователя
    """
    user      =      models.ForeignKey(AuthUser,      on_delete=models.CASCADE,
related_name='licenses')
    text = models.TextField(max_length=1000)


class Genre(models.Model):
    """ Модель жанров треков
    """
    name = models.CharField(max_length=25, unique=True)

    def __str__(self):
        return self.name


class Album(models.Model):
    """
        ORM Model for track's albums
    """
    user      =      models.ForeignKey(AuthUser,      on_delete=models.CASCADE,
related_name='albums')
    name = models.CharField(max_length=50)
    description = models.TextField(max_length=1000)
    private = models.BooleanField(default=False)
    cover = models.ImageField(
        upload_to=get_path_upload_cover_album,
        blank=True,
        null=True,
        validators=[FileExtensionValidator(allowed_extensions=['jpg']),
validate_size_image]
    )

class Track(models.Model):
    """ Модель треков
```

```
    """
    user = models.ForeignKey(AuthUser, on_delete=models.CASCADE,
related_name='tracks')
    title = models.CharField(max_length=100)
    license = models.ForeignKey(License, on_delete=models.PROTECT,
related_name='license_tracks', null=True, blank=True)
    genre = models.ManyToManyField(Genre, related_name='track_genres')
    album = models.ForeignKey(Album, on_delete=models.SET_NULL, blank=True,
null=True)
    link_of_author = models.CharField(max_length=500, blank=True, null=True)
    file = models.FileField(
        upload_to=get_path_upload_track,
        validators=[FileExtensionValidator(allowed_extensions=['mp3', 'wav'])]
    )
    create_at = models.DateTimeField(auto_now_add=True)
    plays_count = models.PositiveIntegerField(default=0)
    download = models.PositiveIntegerField(default=0)
    likes_count = models.PositiveIntegerField(default=0)
    user_of_likes = models.ManyToManyField(AuthUser,
related_name='likes_of_tracks', blank=True)
    private = models.BooleanField(default=False)
    cover = models.ImageField(
        upload_to=get_path_upload_cover_track,
        blank=True,
        null=True,
        validators=[FileExtensionValidator(allowed_extensions=['jpg']),
validate_size_image]
    )

    is_cover = models.BooleanField(default=False)
    is_remix = models.BooleanField(default=False)

    cover_on_music = models.CharField(max_length=255, null=True, blank=True)
    remix_on_music = models.CharField(max_length=255, null=True, blank=True)

    def __str__(self):
        return f'{self.user} - {self.title}'
```

```
class Comment(models.Model):
    """ Модель комментариев к треку
    """
    user       =       models.ForeignKey(AuthUser,       on_delete=models.CASCADE,
related_name='comments')
    track       =       models.ForeignKey(Track,       on_delete=models.CASCADE,
related_name='track_comments')
    text = models.TextField(max_length=1000)
    create_at = models.DateTimeField(auto_now_add=True)


class PlayList(models.Model):
    """ Модель плейлистов пользователя
    """
    user       =       models.ForeignKey(AuthUser,       on_delete=models.CASCADE,
related_name='play_lists')
    title = models.CharField(max_length=50)
    tracks = models.ManyToManyField(Track, related_name='track_play_lists')
    cover = models.ImageField(
        upload_to=get_path_upload_cover_playlist,
        blank=True,
        null=True,
        validators=[FileExtensionValidator(allowed_extensions=['jpg']),
validate_size_image]
    )
```

Файл src/audio_library/views.py – файл контрелерів музичної підпрограми:

```
import os

from django.http import FileResponse, Http404, HttpResponse
from django.shortcuts import get_object_or_404
from django_filters.rest_framework import DjangoFilterBackend
from djoser import serializers
from rest_framework import generics, viewsets, parsers, views

from . import models, serializer
from ..base.classes import MixedSerializer, Pagination
```

```
from ..base.permissions import IsAuthor
from ..base.services import delete_old_file


class GenreView(generics.ListAPIView):
    """ Список жанров
    """
    queryset = models.Genre.objects.all()
    serializer_class = serializer.GenreSerializer


class LicenseView(viewsets.ModelViewSet):
    """
        CRUD for author's licenses
    """
    model = models.License
    serializer_class = serializer.LicenseSerializer

class UnitViewSet(viewsets.ModelViewSet):
    """ API ViewSet for Units """

    queryset = models.Unit.objects.exclude(owner=None)
    serializer_class = serializers.UnitSerializer
    parser_classes = [MultipartJsonParser, JSONParser]
    pagination_class = Pagination
    filter_backends = [DjangoFilterBackend]
    filterset_fields      =      ['category',      'services__name',      'name',
'owner__first_name']
    permission_classes = [IsAuthor]

    def get_queryset(self):
        return models.License.objects.filter(user=self.request.user)

    def perform_create(self, serializer):
        serializer.save(user=self.request.user)

class AlbumView(viewsets.ModelViewSet):
    """ CRUD альбомов автора
    """
```

```
    parser_classes = (parsers.MultiPartParser,)
    serializer_class = serializer.AlbumSerializer
    # permission_classes = [IsAuthor]

    def get_queryset(self):
        return models.Album.objects.filter(user=self.request.user)

    def perform_create(self, serializer):
        serializer.save(user=self.request.user)

    def perform_destroy(self, instance):
        delete_old_file(instance.cover.path)
        instance.delete()

class PublicAlbumView(generics.ListAPIView):
    """ Список публичных альбомов автора
    """
    serializer_class = serializer.AlbumSerializer

    def get_queryset(self):
        return        models.Album.objects.filter(user__id=self.kwargs.get('pk'),
private=False)

class TrackView(MixedSerializer, viewsets.ModelViewSet):
    """ CRUD треков
    """
    parser_classes = (parsers.MultiPartParser,)
    permission_classes = [IsAuthor]
    pagination_class = Pagination
    filter_backends = [DjangoFilterBackend]
    filterset_fields = ['name', 'album', 'genre']
    serializer_class = serializer.CreateAuthorTrackSerializer
    serializer_classes_by_action = {
        'list': serializer.AuthorTrackSerializer
    }

    def get_queryset(self):
        return models.Track.objects.filter(user=self.request.user)
```

```
    def perform_create(self, serializer):
        serializer.save(user=self.request.user)

    def perform_destroy(self, instance):
        delete_old_file(instance.cover.path)
        delete_old_file(instance.file.path)
        instance.delete()

class PlayListView(MixedSerializer, viewsets.ModelViewSet):
    """ CRUD плейлистов пользователя
    """
    parser_classes = (parsers.MultiPartParser,)
    permission_classes = [IsAuthor]
    serializer_class = serializer.CreatePlayListSerializer
    serializer_classes_by_action = {
        'list': serializer.PlayListSerializer
    }

    def get_queryset(self):
        return models.PlayList.objects.filter(user=self.request.user)

    def perform_create(self, serializer):
        serializer.save(user=self.request.user)

    def perform_destroy(self, instance):
        delete_old_file(instance.cover.path)
        instance.delete()

class TrackListView(generics.ListAPIView):
    """ Список всех треков
    """
    queryset = models.Track.objects.filter(album__private=False, private=False)
    serializer_class = serializer.AuthorTrackSerializer
    pagination_class = Pagination
    filter_backends = [DjangoFilterBackend]
    filterset_fields    =    ['title',    'user__display_name',    'album__name',
'genre__name']

class AuthorTrackListView(generics.ListAPIView):
```

```
    """ Список всех треков автора
    """
    serializer_class = serializer.AuthorTrackSerializer
    pagination_class = Pagination
    filter_backends = [DjangoFilterBackend]
    filterset_fields = ['title', 'album__name', 'genre__name']

    def get_queryset(self):
        return models.Track.objects.filter(
            user__id=self.kwargs.get('pk'), album__private=False, private=False
        )

class CommentAuthorView(viewsets.ModelViewSet):
    """ CRUD комментариев автора
    """
    serializer_class = serializer.CommentAuthorSerializer
    # permission_classes = [IsAuthor]

    def get_queryset(self):
        return models.Comment.objects.filter(user=self.request.user)

    def perform_create(self, serializer):
        serializer.save(user=self.request.user)
```

Файл src/oauth/views.py – файл контрелерів користувача підпрограми:

```
from rest_framework import viewsets, parsers, permissions

from .. import serializer, models
from ...base.permissions import IsAuthor

class UserView(viewsets.ModelViewSet):
    """ User CRUD """
    parser_classes = (parsers.MultiPartParser,)
    serializer_class = serializer.UserSerializer
    permission_classes = [permissions.IsAuthenticated]

    def get_queryset(self):
```

```
        return self.request.user

    def get_object(self):
        return self.get_queryset()

class AuthorView(viewsets.ReadOnlyModelViewSet):
    """ The list of Authors """
    queryset = models.AuthUser.objects.all().prefetch_related('social_links')
    serializer_class = serializer.AuthorSerializer

class SocialLinkView(viewsets.ModelViewSet):
    """ The CRUD for SocialLinks """
    queryset = models.SocialLink.objects.all()
    serializer_class = serializer.SocialLinkSerializer
    permission_classes = [IsAuthor]

    def get_queryset(self):
        return self.request.user.social_links.all()

    def perform_create(self, serializer):
        serializer.save(user=self.request.user)
```

Файл src/oauth/urls.py – файл кінцевих для контролерів користувача точок (url):

```
from django.urls import path

from .endpoint import views, auth_views

urlpatterns = [
    path('me/', views.UserView.as_view({'get': 'retrieve', 'put': 'update'})),

    path('author/', views.AuthorView.as_view({'get': 'list'})),
    path('author/<int:pk>', views.AuthorView.as_view({'get': 'retrieve'})),

    path('social/', views.SocialLinkView.as_view(
        {'get': 'list', 'post': 'create'}
    )),
    path('social/<int:pk>', views.SocialLinkView.as_view(
```

```
        {'put': 'update', 'delete': 'destroy'}
    )),

    path('google/', auth_views.google_auth),
    path('spotify-callback/', auth_views.spotify_auth),

    path('spotify-login/', auth_views.spotify_login),
    path('', auth_views.google_login),
]
```

Файл src/audio_library/urls.py – файл кінцевих для контролерів аудіо підпрограми точок (url):

```
from django.urls import path
from . import views

urlpatterns = [
    path('genre/', views.GenreView.as_view()),

    path('license/',    views.LicenseView.as_view({'get':    'list',    'post':
'create'})),
    path('license/<int:pk>/',    views.LicenseView.as_view({'put':    'update',
'delete': 'destroy'})),

    path('album/', views.AlbumView.as_view({'get': 'list', 'post': 'create'})),
    path('album/<int:pk>/', views.AlbumView.as_view({'put': 'update', 'delete':
'destroy'})),

    path('author-album/<int:pk>/', views.PublicAlbumView.as_view()),

    path('track/', views.TrackView.as_view({'get': 'list', 'post': 'create'})),
    path('track/<int:pk>/', views.TrackView.as_view({'put': 'update', 'delete':
'destroy'})),

    path('stream-track/<int:pk>/', views.StreamingFileView.as_view()),
    path('download-track/<int:pk>/', views.DownloadTrackView.as_view()),
```

```
    path('stream-author-track/<int:pk>/',
views.StreamingFileAuthorView.as_view()),

    path('track-list', views.TrackListView.as_view()),
    path('author-track-list/<int:pk>/', views.AuthorTrackListView.as_view()),

    path('comments/',  views.CommentAuthorView.as_view({'get':  'list',  'post':
'create'})),
    path('comments/<int:pk>/', views.CommentAuthorView.as_view({'put': 'update',
'delete': 'destroy'})),

    path('comments_by_track/<int:pk>/',         views.CommentView.as_view({'get':
'list'})),

    path('playlist/',    views.PlayListView.as_view({'get':    'list',    'post':
'create'})),
    path('playlist/<int:pk>/',    views.PlayListView.as_view({'put':    'update',
'delete': 'destroy'})),
]
```

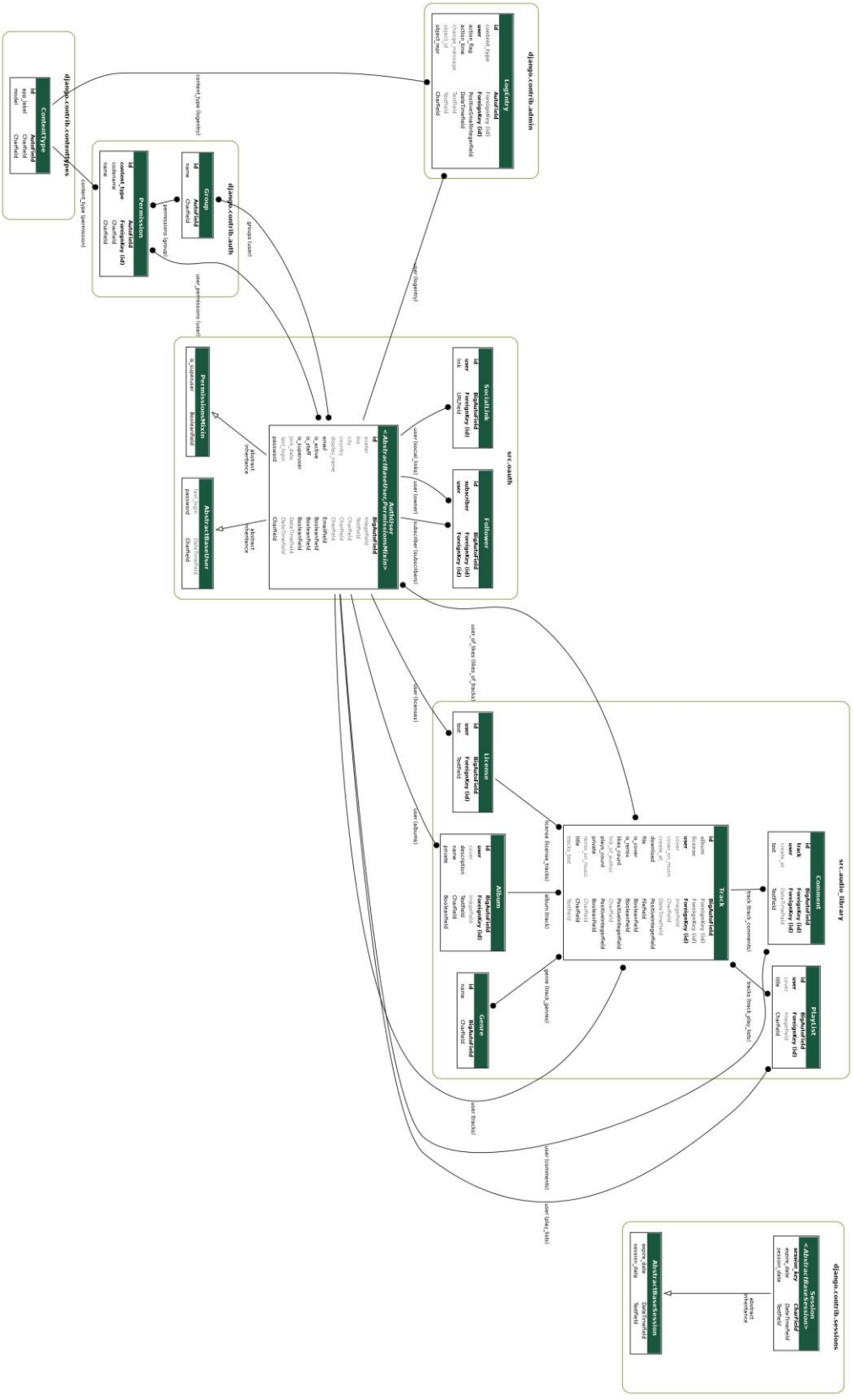

Рисунок 1 – повна діаграма класів моделей проєкту

ДОДАТОК Б
ПРОГРАМНИЙ КОД КЛІЄНТСЬКОЇ ЧАСТИНИ

Файл main.js у корні директорії src, файл з головними налаштуваннями:

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './router'
import store from "./store";

createApp(App).use(store).use(router).mount('#app')
```

Файл App.vue у корні директорії src, файл з головними налаштуваннями до html:

```
<template>
  <router-view/>
</template>

<script>
export default {
  name: 'App',
  components: {

  }
}
</script>
```

Файл router, файл налаштуваннями для роботи з url шляхами:

```
import { createRouter, createWebHistory } from 'vue-router'
//Main
import Main from '../views/Main.vue'
//Login
import Login from '../views/Login.vue'
//Profile
```

```
import Profile from '../views/Profile.vue'
import ProfileUpdate from '../views/ProfileUpdate.vue'
//Users
import Users from '../views/Users.vue'
import TargetUser from '../views/TargetUser.vue'
//Albums
import Album from '../views/Album.vue'
import AlbumCreate from '../views/AlbumCreate.vue'
import AlbumAuthor from '../views/AlbumAuthor.vue'
import AlbumUpdate from '../views/AlbumUpdate.vue'
//Tracks
import Tracks from '../views/Tracks.vue'
import TrackCreate from '../views/TrackCreate.vue'
import TrackUpdate from '../views/TrackUpdate.vue'
//PlayList
import PlayList from '../views/PlayList.vue'
import PlayListCreate from '../views/PlayListCreate.vue'
import PlayListRetrieve from '../views/PlayListRetrieve.vue'


import Player from '../views/Player.vue'


const routes = [
  {
    path: '/',
    name: 'Main',
    component: Main
  },
  {
    path: '/login',
    name: 'Login',
    component: Login
  },
  {
    path: '/me',
    name: 'Profile',
    component: Profile
  },
```

```
{
  path: '/me/update',
  name: 'ProfileUpdate',
  component: ProfileUpdate
},
{
  path: '/users',
  name: 'Users',
  component: Users
},
{
  path: '/user/:id',
  name: 'TargetUser',
  component: TargetUser,
  props: true
},
{
  path: '/albums',
  name: 'Album',
  component: Album
},
{
  path: '/album/create',
  name: 'AlbumCreate',
  component: AlbumCreate
},
{
  path: '/user/album/:id',
  name: 'AlbumAuthor',
  component: AlbumAuthor,
  props: true
},
{
  path: '/album/update/:id',
  name: 'AlbumUpdate',
  component: AlbumUpdate,
  props: true
},
{
```

```
  path: '/tracks',
  name: 'Tracks',
  component: Tracks,
  props: true
},
{
  path: '/track/create',
  name: 'TrackCreate',
  component: TrackCreate,
  props: true
},
{
  path: '/track/update',
  name: 'TrackUpdate',
  component: TrackUpdate,
  props: true
},
{
  path: '/playlist',
  name: 'PlayList',
  component: PlayList,
  props: true
},
{
  path: '/playlist/create',
  name: 'PlayListCreate',
  component: PlayListCreate,
  props: true
},
{
  path: '/playlist/retrieve',
  name: 'PlayListRetrieve',
  component: PlayListRetrieve,
  props: true
},
{
  path: '/player',
  name: 'Player',
  component: Player,
```

```
    props: true
  },
]

const router = createRouter({
  history: createWebHistory(),
  routes
})

export default router
```

Файл store, файл налаштуваннями для роботи з зберіганням статичних даних:

```
import Vuex from 'vuex'

const store = new Vuex.Store({
    state: {
        backendUrl: "http://127.0.0.1:8000/api/v1"
    },
    mutations: {},
    actions: {},
    modules: {},
    getters: {
        getServerUrl: state => {
            return state.backendUrl
        }
    }
})
export default store
```

Файл Login.vue, файл де зберігається логіка реєстрації та автентифікації користувача:

```
<script>

import Header from "../components/Header"
import Footer from "../components/Footer"

import axios from 'axios'
import { ref } from 'vue'
```

```
import { useStore } from 'vuex'

export default {

  name: "Login",
  components: {
    Header,
    Footer,
  },

  setup() {

    const store = useStore()

    const login = ref('')
    const password = ref('')
    const res = ref('')

    const postLogin = async () => {
      const data = {
        "email": login.value,
        "password": password.value
      }
      const config = {
        headers: {
          'Content-Type': 'application/json'
        }
      }
      // post login and password to get jwt access token in response
      await axios.post(
      `${store.getters.getServerUrl}/auth/jwt/create/`,
        data, config
      ).then(response => res.value = response.data)
    console.log(res.value.access)

      // Add jwt token to localStorage
      localStorage.setItem('token', res.value.access)
      const token = localStorage.getItem('token')
      console.log('Token:', token)
```

```
        }

        return {
          login,
          password,
          res,
          postLogin
        }

      }

    }
    </script>
```

Файл Profile.vue, файл де зберігається логіка реєстрації та автентифікації користувача:

```
    <script>

    import Header from "../components/Header"
    import Footer from "../components/Footer"

    import axios from 'axios'
    import { ref, onMounted } from 'vue'
    import { useStore } from 'vuex'

    export default {
      name: "ProfileUpdate",
      components: {
        Header,
        Footer,
      },

      setup() {

        const store = useStore()
        const avatar = ref('')
        const bio = ref('')
        const country = ref('')
```

```
const city = ref('')
const display_name = ref('')

const profile = ref([])

const getProfileInfo = async () => {

       await axios.get(
         `${store.getters.getServerUrl}/auth/me/`, {
         headers: {
           Authorization: 'Bearer ' + localStorage.getItem('token')
         }
        }
       ).then(response => profile.value = response.data)
       console.log(profile.value['avatar'])
       bio.value = profile.value['bio']
       country.value = profile.value['country']
       city.value = profile.value['city']
       display_name.value = profile.value['display_name']
}

const fileSend = (event) => {
  avatar.value = event.target.files[0]
  console.log(avatar.value)
}

const updateProfileInfo = async () => {
  const formData = new FormData();
  formData.append("bio", bio.value)
  formData.append("country", country.value)
  formData.append("city", city.value)
  formData.append("avatar", avatar.value)
  formData.append("display_name", display_name.value)

  const config = {
    headers: {
      'Content-Type': 'multipart/form-data; boundary=something',
       Authorization: 'Bearer ' + localStorage.getItem('token')
    }
```

```
      }

       await axios.put(
         `${store.getters.getServerUrl}/auth/me/`,
         formData,
         config
       )
    }

    onMounted(getProfileInfo)
    return {
      getProfileInfo,
      profile,
      fileSend,
      updateProfileInfo,
      bio,
      country,
      city,
      avatar,
      display_name,
    }
  }

}
</script>
```

Файл Player.vue, файл де зберігається логіка потокової аудіо-передачі:

```
<template>
  <Header />
  <div class="container">
      <div class="music">
        <div class="music_cover">
          <div class="cover_img">
            <img     class="music_img"     src="https://1.bp.blogspot.com/-
Ql66Km9KC4c/X9udKGnzTuI/AAAAAAAABTw/_oN4G22NsagyjdB2RauumnIlmg1Lc5P6gCLcBG
AsYHQ/s800/slipknot.jpg" alt="">
            <p class="music_title">{{target_title}}</p>
```

```
          </div>
          <div class="cover_text">
            <span name="name">{{target_text}}</span>
          </div>
        </div>
        <div class="music_con" :key="id" v-for="(t, id) in tracks">
          <div class="music_plates">
            <a  class="btn_play"  href="#"  @click="getTargetTrack(t.id)"><i
class="fa-solid fa-play"></i></a>
            <p class="btn_title">{{t.title}}</p>
            <img      class="btn_img"      src="https://1.bp.blogspot.com/-
Ql66Km9KC4c/X9udKGnzTuI/AAAAAAAABTw/_oN4G22NsagyjdB2RauumnIlmg1Lc5P6gCLcBG
AsYHQ/s800/slipknot.jpg" alt="">
          </div>
        </div>
      </div>
      <!-- <a href="#" @click="goTo()">Create new album</a> -->
  </div>
  <audio controls
  ref="target_track"
  >
  <source
src="https://cdn1.sefon.pro/prev/hjhWujIJgQCli4WZkkdjRw/1654061691/82/Slip
knot%20-%20All%20Out%20Life%20%28192kbps%29.mp3" type="audio/mpeg">
</audio>
  <Footer />
</template>


<script>

import Header from "../components/Header"
import Footer from "../components/Footer"
import axios from 'axios'
import { ref, onMounted } from 'vue'
import { useStore } from 'vuex'

export default {
```

```
name: "Player",
components: {
  Header,
  Footer,
},

setup() {

  const store = useStore()
  const tracks = ref([])
  const target_track = ref('')
  const target_img = ref('')
  const target_text = ref('')
  const target_title = ref('')

  const getAllTracks = async () => {

         await axios.get(
           `${store.getters.getServerUrl}/audio/track/`, {
           headers: {
             Authorization: 'Bearer ' + localStorage.getItem('token')
           }
          }
         ).then(response => tracks.value = response.data)
         getTargetTrack(tracks.value[0].id)
  }
  const getTargetTrack = async (id) => {
    for (let track of tracks.value ) {
      if (track.id == id) {
        target_track.value = track.file
        target_img.value = track.cover
        target_text.value = track.tracks_text
        target_title.value = track.title
      }
    }
  }

  onMounted(getAllTracks)
  return {
```

```
      tracks,
      target_track,
      target_img,
      target_text,
      target_title,
      getTargetTrack
    }
  }
}
</script>
```