МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри, канд. техн. наук,
доцент _____ Є. О. Давиденко
підпис

«___» _____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА
ВЕБЗАСТОСУНОК ПОТОКОВОЇ ПЕРЕДАЧІ МУЗИКИ
Спеціальність «Інженерія програмного забезпечення»
121 – КРБ.1 – 409. 21910909

Студента _____ О. І. Євдокімов
підпис
«__»_____2023 р.

Керівник PhD, ст. викладач _____ І. О. Кандиба
підпис
«__» _____2023 р.

Консультант к.т.н, доцент кафедри екології _____ А. О. Алексєєва
підпис
«__» _____2023 р.

м. Миколаїв – 2023 рік

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ
Зав. кафедри
_____ Є. О. Давиденко
«_____» _____ 2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи бакалавра

Видано студенту групи 409 факультету комп'ютерних наук

Євдокімову Олександру Ігоровичу

(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи : « Вебзастосунок потокової передачі музики »

Затверджена наказом по ЧНУ від «17» березня 2023 р. № 60

2. Строк представлення кваліфікаційної роботи «___»_____20___р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: вебзастосунок потокової передачі музики .

4. Перелік питань, що підлягають розробці:

- дослідження аналогів;
- проведення аналізу вимог до вебзастосунку;
- проведення аналізу сучасних інструментів для реалізації системи;
- проектування плану розробки вебзастосунку;
- реалізація та тестування вебзастосунку .

5. Перелік графічних матеріалів: презентація.

6. Завдання до спеціальної частини: Аналіз охорони праці на робочих місцях фахівців з інформаційних технологій.

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
Алексєєва А. О.	Кафедра екології Медичного інституту ЧНУ ім. Петра Могили	Розділ охорони праці

Керівник роботи PhD, старший викладач Кандиба Ігор Олександрович

(посада, прізвище, ім'я, по батькові)

_____

(підпис)

Завдання прийнято до виконання

Євдокімов Олександр Ігорович

(прізвище, ім'я, по батькові студента)

Дата видачі завдання «22» лютого 2023 р.

Календарний план виконання кваліфікаційної роботи
КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

Тема: Вебзастосунок потокової передачі музики

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання навиконання КРБ	01.12.2022	18.12.2022	виконано
2.	Огляд літератури за темою роботи	18.12.2022	18.01.2023	виконано
3.	Складання календарного плану КРБ	18.01.2023	19.01.2023	виконано
4.	Аналіз предметної області	20.01.2023	01.02.2023	виконано
5.	Розробка проєктних рішень	02.02.2023	10.02.2023	виконано
6.	Моделювання та конструювання ПЗ	11.02.2023	23.02.2023	виконано
7.	Кодування, тестування та апробаціярозробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	23.02.2023	15.04.2023	виконано
8.	Розробка спеціальної частини з охорони праці	15.04.2023	30.04.2023	виконано
9.	Відгук керівника КРБ	01.04.2023	02.04.2023	виконано
10.	Оформлення КРБ та презентації	02.04.2023	03.04.2023	виконано
11.	Попередній захист	06.06.2023	07.06.2023	виконано
12.	Рецензування	14.06.2023	19.06.2023	виконано
13.	Завершення оформлення КРБ та презентації	19.06.2023	25.06.2023	виконано
14.	Захист кваліфікаційної роботи	26.06.2023	27.06.2023	виконано

Розробив студент Євдокімов Олександр Ігорович _____

(*прізвище, ім'я, по батькові)* *(підпис)*

«____» 2023р.

Керівник роботи PhD, ст. викладач Кандиба Ігор Олександрович _____

(*посада, прізвище, ім'я, по батькові)* *(підпис)*

«__» 2023р.

АНОТАЦІЯ

до кваліфікаційної роботи бакалавра

«Вебзастосунок потокової передачі музики»

Студент 409 гр.: Євдокімов Олександр Ігорович

Керівник: PhD, ст. викладач Кандиба Ігор Олександрович

Об’єктом дослідження даної роботи є процес конструювання вебзастосунку потокової передачі музики.

Предметом дослідження є програмні засоби та методи конструювання вебзастосунку.

Метою роботи є популяризація композиторів, що мають малу аудиторію за рахунок розробки системи рекомендацій музики в сервісі потокової передачі музики.

Для досягнення визначної мети необхідно вирішити такі завдання:

– дослідження аналогів;

– провести аналіз вимоги до вебзастосунку;

– провести аналіз сучасних інструментів для реалізації системи;

– проектування плану розробки вебзастосунку;

– реалізація та тестування вебзастосунку.

Кваліфікаційна робота бакалавра складається з вступу, трьох розділів, висновків та переліку джерел посилань.

У першому розділі описується процес аналізу предметної області та специфікацій вимог.

У другому розділі описуються технології та архітектура розробки програмного забезпечення. Також розглядається діаграми UML.

У третьому розділі проводиться демонстрація функцій, контролерів, сервісів та компонентів програмного забезпечення.

У висновках описується аналіз розробленої роботи та готових результатів.

Ключові слова: *дистрибуція музики, рівень абстракції, бізнес-логіка, токен доступу, проміжна функція, система рекомендацій.*

ABSTRACT

of the Bachelor’s Thesis

"Music streaming app"

Student of group 409: Yevdokimov Oleksandr Ihorovich

Supervisor: Phd, s. lecture Kandyba Ihor Oleksandrovich

The research object of this work is the process of designing a music streaming web application.

The subject of the research is software tools and methods of designing a web application.

The purpose of the work is to popularize composers with a small audience through the development of a music recommendation system in a music streaming service. To achieve a significant goal, the following tasks must be solved:

– study of analogues;

– analyze the requirements for the web application;

– conduct an analysis of modern tools for implementing the system;

– designing a web application development plan;

– implementation and testing of the web application.

The bachelor's thesis consists of an introduction, three chapters, conclusions and a list of reference sources.

The first chapter describes the process of domain analysis and requirements specifications.

The second chapter describes software development technologies and architecture. UML diagrams are also considered.

In the third chapter, the functions, controllers, services and software components are demonstrated.

The conclusions describe the analysis of the developed work and the finished results.

Keywords: *music distribution, abstraction level, business logic, access token, intermediate function, recommendation system.*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ПЗ – програмне забезпечення

DSP – direct artist platforms

CTR – click-through rate

CD – compact disk

BFF – backend for frontend

MVVM – Model – View – Viewmodel

GUI – Graphical user interface

IT – Information Technology

JSON – JavaScript Object Notation

JWT – JSON Web Token

AMQP – Advanced Message Queuing Protocol

HTTP – Hypertext Transfer Protocol

TCP – Transmission Control Protocol

IP – Internet Protocol

API – Application Programming Interface

ODM – Object Data Modeling

ORM – Object – Relational Mapping

CRUD – create read update delete

UML – Unified Modeling Language

ВСТУП

Найбільш популярні сервіси в інтернеті, якими користуються користувачі – це музичні сайти. Для деяких людей музика – це частина життя. Музичні сервіси є зручними не тільки для слухачів, а також для композиторів. Саме тому значна кількість слухачів знаходиться на музичних сервісах. Невідомі композитори теж можуть стати популярними завдяки системі рекомендації музики.

Об’єктом кваліфікаційної роботи є процес конструювання вебзастосунку потокової передачі музики.

Предметом кваліфікаційної роботи є програмні засоби та методи конструювання вебзастосунків.

Метою роботи є популяризація композиторів, що мають малу аудиторію шляхом розробки системи рекомендацій музики в сервісі потокової передачі музики.

Для досягнення визначної мети необхідно вирішити такі завдання:

– дослідження аналогів;

– проведення аналізу вимоги до вебзастосунку;

– проведення аналізу сучасних інструментів для реалізації системи;

– проектування плану розробки вебзастосунку;

– реалізація та тестування вебзастосунку.

Сучасною проблемою музичних вебзастосунків є погана система рекомендацій, людина буде слухати ту музику, яка має найбільшу кількість прослуховувань і не має значення це якісна музика чи навпаки неякісна. Вирішення проблеми надає можливість користувачу стати композитором, і випускати власну музику.

Основними проектними рішеннями являються:

– створення приємного інтерфейсу;

– придбання підписки для прослуховування трендової музики;

– фільтрування музики за жанром, тривалістю, композитором та датою;

– можливість безкоштовно завантажувати музику з сайту та на сайт;

– змога користувачів слідкувати за користувачами.

У результаті конструювання нового музичного вебсервісу, який буде використовуватися слухачами, для прослуховування музики та композиторами для популяризації. Одна із основних цілей застосунку є вихід на ринок музичних сервісів.

1 АНАЛІЗ СЕРВІСІВ ПОТОКОВОЇ ПЕРЕДАЧІ МУЗИКИ

1.1 Дистрибуція музики

Стрімінгові сервіси працюють за принципом передачі контенту від провайдера до користувача. Весь контент вже завантажений на сторонньому сервері, кінцевому користувачу не потрібно нічого завантажувати для перегляду або прослуховування [1].

Контент транслюється в режимі реального часу, швидкість завантаження залежить від швидкості інтернету користувача. З нинішнім навіть найпростішим підключенням до мережі Інтернет можна без проблем прослуховувати музику та переглядати відео зі стрімінгових сервісів.

Стрімінгові сервіси спонукають композиторів писати музику популярних жанрів. Чим експерементальніші пісні пише виконавець, тим менше з них отримує або взагалі не заробляє нічого, але музикантові потрібен зв'язок із аудиторією.

Переваги стрімінгових сервісів:

– зручність використання – не потрібно заздалегідь закачувати на картку пам'яті всю бібліотеку;

– якість – на послуги не потрапляють низькоякісні або обрізані записи;

– добірки – на платформах передбачено алгоритм, який допомагає вам знайти саме ту музику, яка вам близька;

– підтримка виконавців – всі люблять слухати музику безкоштовно, але справжнім шанувальникам завжди приємно зробити невеликий внесок у розвиток музиканта.

Декілька років тому єдиним способом поширення музики було укладення контракту. Не кажучи вже про просування радіо та маркетингу, щоб допомогти продати запис. Але зараз все по-іншому. Музиканти можуть створювати професійні треки, що звучать, використовуючи лише ноутбук і мікрофон.

Музиканти можуть почути свою музику у всьому світі, навіть не випускаючи жодної платівки чи диску на фізичному носії. CD дисками, так само як і dvd плеєрами перестали користуватися, вже декілька років тому.

Замість них, прийшли стримінгові сервіси, або сервіси потокової передачі музизи. Музику стало легко популярізувати, треба лиш завантажити аудіо файл на спеціалізований сервіс та найняти дистриб'ютора. Контент, що розповсюджується в мережі, може передаватися в потоковому режимі або завантажуватись. Спеціалізовані мережі, відомі як мережі доставки контенту, допомагають розповсюджувати контент через Інтернет, забезпечуючи як високу доступність, так і високу продуктивність.

Ще один спосіб розповсюдити свою музики додати її у фільм або гру, але є мінус користувач може непомітити цю музику.

Сьогодні зробити пісню доступною для слухачів по всьому світу так само просто, як і завантажити файл в інтернеті. Але існує потреба найняти посередника для поширення.

Дистрибуція на цифрових сервісах – це доставка творчості на всі цифрові вітрини та майданчики для того, щоб слухачі могли слухати музику без обмежень, а виконавець отримувати дохід. Музичні компанії пропонують цю послугу для автоматизації цього процесу для артиста без безпосередньої участі. Для того щоб цей процес проходив якомога плавніше, існують компанії, які орієнтовані саме на ці послуги. Компанії займають своє місце на музичному ринку і є тим посередником між кінцевим споживачем і самим продуктом. Для успішного поширення треку, навіть серед аудиторії можливих слухачів, потрібні достатні знання та позитивний досвід дистрибуції інших треків. А також чітке розуміння аудиторії виконавця та усвідомлення того, яке місце сам виконавець та його пісня уже займають у цьому сегменті.

Дистриб'ютори є одною з головних частин у популярізяції музики. Дистриб'ютори виконують три основні ролі:

1. поширення релізів на DSP;
2. розподіл роялті;

3. стратерігя розподілу і торговий маркетинг.

Перша роль полягає у тому що метадані релізу відповідають вимогам платформи. Друга роль полягає у розподілі роялті, або іншими словами платежів за користування належним правовласникам. Третя роль полягає у тому щоб кожен користувач сервісу побачить цей самий реліз [2].

1.2 Система рекомендацій стримінгових сервісів

Система рекомендацій – це система фільтрації даних, мета цієї системи передбачити перевагу, яку саме продукту користувач віддасть. Це ядро величезних механізмів, які працюють за алгоритмами і пропонують користувачу окремий набір елементів.

Типовий механізм рекомендацій складається з чотирьох етапів, а саме збору, зберігання, аналізу та фільтрації даних. Збір даних: само собою зрозуміло, що для ефективного функціонування будь-якої рекомендації знадобиться велика кількість даних.

Зібрані дані можуть мати явний або неявний характер. Явні дані – це дані, зібрані з Інтернету на основі дій користувача в Інтернеті, як-от оцінки, огляди телевізійних шоу, фільмів, пісень тощо. Неявні дані – це дані, отримані з дій клієнтів, як-от історія замовлень, історія повернень, перегляди сторінок, кліки – прохідні та інші дії. Дані можна зібрати безпосередньо з дій користувача, не вимагаючи жодних дій з боку користувача.

Для створення механізму рекомендацій потрібні величезні обсяги даних. Дані збираються зі звичок перегляду споживача. Типові зібрані дані складаються з:

– дата і час перегляду контенту;

– розташування та поштовий індекс;

– рейтинги;

– відгуки;

– поведінка перегляду;

– коли музику призупинено, перемотується назад або вперед;

Є дві популярні системи рекомендацій:

– на основі контенту – рекомендації, які базуються на схожості контенту або атрибутів елементу);

– спільні – рекомендації основані на подібності вподобань користувачів і використання матриць з рейтингами для кожного фрагменту контенту).

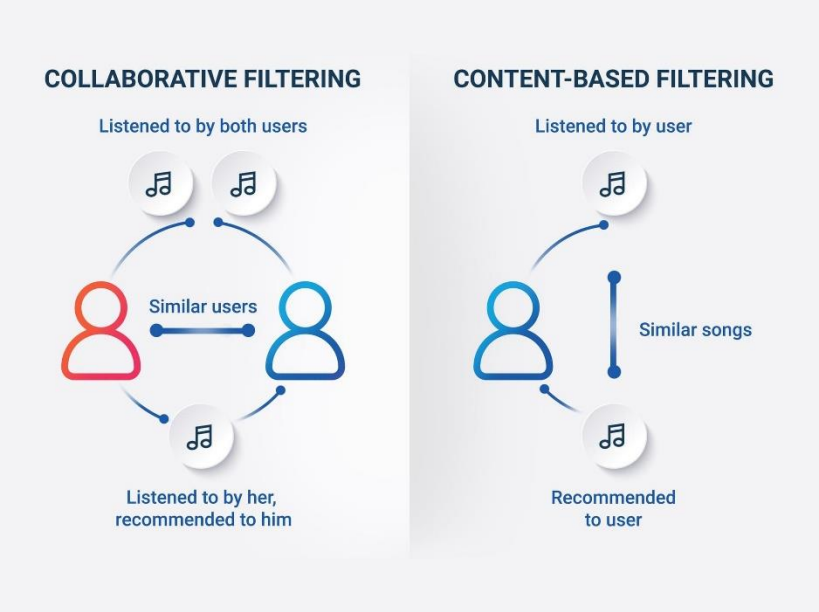

Рисунок 1.1 – Схеми робіт систем рекомендацій

Контентний підхід базується на схожості окремих елементів. Коли користувач користується потоковим музичним сервісом, і якщо музика, автор, жанр, або інші елементи сервісу, які сподобалися йому, переходять у список відтворення. Основна ідея системи рекомендацій на основі контенту полягає в тому, щоб отримати ключові слова з опису пісні, які подобається користувачеві та порівняти їх з ключовими словами з інших пісень та на основі цих порівнянь вивести систему рекомендацій.

Система спільного контенту будується на основі пересічення уподобань користувачів і оцінок музики. Припускається, якщо користувач А та користувач Б висловлюють одинакові переваги, цим користувач рекомендується схожий контент. Тобто якщо користувачеві А подобається певна пісня, цілком ймовірно, що ця пісня також сподобається користувачеві Б і навпаки. Спільні системи

рекомендацій вважаться більш точними, оскільки спираються на користувача з системою, а не на схожість вмісту.

Ці системи не потребують додаткових дій з боку користувача. Людина встановлює додаток або реєструється, слухає улюблені пісні, складає плейлісти. І чим більше даних про вподобання користувача у сервісу, тим точніші стають рекомендації музики. Це означає, що зі зростанням точності алгоритмів користувач буде насолоджуватися музикою і ще більше задоволений музичним сервісом.

Сьогодні люди звикли до систем рекомендацій у всіх сферах: стримінгові сервіси, стрічки новин, потокові аудіо та відео контент, інтернет-магазини.

Служби підписки на музику використовують системи рекомендацій, щоб формувати окрему музику у списку відтворення за певними критеріями. Це для зручності користувача та підвищення рівня задоволеності клієнта.

Причини чому потрібна система рекомендацій:

– збільшення залученість і задоволеність клієнтів;

– персоналізація платформи;

– забезпечення якісного досвіду потового передавання музики;

– формування довіри та звичок слухати своїх виконавців;

– забезпечення зручності використання сервісу;

– збільшення прослуховування аудіо контенту;

– популярізація невідомої музики;

– підвищення CTR і конверсії.

Як і будь-який інший тип системи рекомендацій, сучасна система рекомендацій щодо музики побудована з використанням машинного навчання та штучного інтелекту. Створення такої системи вимагає високої експертизи в цих областях, набору даних про користувачів і пісні, які вони слухають. Використовуючи ці дані, є можливість комбінувати моделі та створювати індивідуальні рішення для конкретного бізнес-кейсу.

Наприклад, можна побудувати і запрограмувати алгоритми, які будуть орієнтуватися на час, коли користувач слухає певну пісню. Завдяки цьому ми можемо генерувати персоналізовані списки відтворення на основі часових критеріїв, наприклад списки відтворення ранкової музики чи списки відтворення на вихідні. Отже, незважаючи на те, що основні принципи систем рекомендацій щодо музики подібні, їх прикладна цінність може бути різною.

1.3 Аналіз аналогів

Для проведення аналізу предметної галузі досліджено 3 застосунки аналоги.

Характеристики Spotify

Назва: Spotify: Music, Podcasts, Lit

Розробник: Spotify

Архітектура: Microservices architecture [3]

Мова реалізації: Python , Java

Перелік функцій, характеристик:

1. вибір і відтворення будь-якого треку на мобільному телефоні;
2. завантаження музики на сайт;
3. скачування музики та підкастів з сайту;
4. можливість вибрати улюблений жанр та автора музики;
5. технологія «Spotify Connect» – можливість керування віддаленим прослуховуванням;
6. можливість підписуватися на іншого користувача;
7. преміум статус;
8. можливість перегляду тексту музики .

Аналіз переваг та недоліків:

Переваги :

1. висока якість відтворення музики;
2. приємний інтерфейс;
3. можливість вибрати улюблений жанр та автора музики;

4. можливість прослуховування трендів (Spotify Charts);

5. персональна статистика прослуханої музики;

6. групове прослуховування музики.

Недоліки :

1. безкоштовний період всього місяць;

2. без покупки Premium статусу, користувач слухає рекламу;

3. без Premium статусу неможливо скачати з сайту музику або завантажити на сайт;

4. без Premium статусу немає можливості повторного прослуховування треку.

Характеристики SoundCloud

Назва: SoundCloud

Розробник: SoundCloud

Архітектура: BFF pattern [4]

Мова реалізації: Go, Scala, Ruby, JavaScript

Перелік функцій, характеристик:

1. вибір і відтворення будь-якого треку на мобільному телефоні;

2. завантаження музики на сайт;

3. скачування музики з сайту;

4. можливість вибрати улюблений жанр та автора музики;

5. можливість підписуватися на іншого користувача;

6. можливість додавати музику до свого плейлісту;

7. покупка Premium статусу.

Аналіз переваг та недоліків:

Переваги :

1. непогана якість музики;

2. безкоштовне прослуховування музики;

3. безкоштовне завантаження музики на сайт створення свого плейлісту з улюбленої музики;

4. коментування треку.

Недоліки :

1. можливість завантажувати та скачувати файли з лімітом до 5 ГБ;

2. можливість скачувати тільки якщо є Premium статус;

3. немає точної статистики прослуховування користувача.

Характеристики Deezer

Назва: Deezer

Розробник: Access Industries

Архітектура: MVVM architecture [5]

Мова реалізації: PHP, JavaScript, C#, Python

Перелік функцій, характеристик:

1. вибір і відтворення будь-якого треку;

2. можливість вибрати улюблений жанр та автора музики;

3. можливість підписуватися на іншого користувача;

4. можливість додавати музику до свого плейлісту;

5. придбання підписок.

Аналіз переваг та недоліків:

1. великий каталог музики;

2. якість передачі музики 320 кбіт/с;

3. хороші добірки плейлистів;

4. автоматичний плейлист Flow;

5. зручний інтерфейс;

6. підтримка розширень усередині сервісу;

7. підтримка великої кількості пристроїв та платформ;

8. можна слухати музику оффлайн;

9. гнучке управління підключеними пристроями;

10. є безкоштовна версія.

Недоліки:

1. висока вартість передплати в Україні;

2. у безкоштовній версії якість музики урізана до 128 кбіт/с;

3. можна підключити сервіс лише до трьох пристроїв.

Аналіз системи, що розробляється

Основні задачі :

– можливість прослуховувати музику не завантажуючи її з сайту;

– завантажувати музику на сайт;

– завантаження музики;

– пошук музики;

– сортування музики за властивостями;

– редагування музики;

– коментування музики;

– реєстрація користувача;

– авторизація користувача;

– редагування профіля;

– чат підтримки;

– статистика прослуховування музики;

– статистика вподобань користувачів;

– рекомендації для користувача;

– створення плейлістів.

Користувачі системи:

– user;

– support;

– compositor;

– admin.

Special Requirements:

1.1 система перекладу мов на декілька мов;

1.2 система правил надання користувачу прав композитора.

Technology and Data Variations List:

1) для реєстрації обов'язково треба мати e-mail;

2) для завантаження музики, розмір файлу повинно бути не більше 20мб та не менше 5мб.

Frequency of Occurrence: музичній сервіс буде працювати цілодобово, окрім днів тех-обслуговування.

Miscellaneous (Open Issues):

– сервіс має систему підтримки, де користувач може задати питання;

– сервіс має можливість змінити тему інтерфейсу на вибір користувача;

– користувач може сам встановити жанр завантаженої музики.

Основні таблиці БД :

– users;

– roles;

– musics;

– geners;

– recommends;

– comments;

– playlists;

– albums;

– questions.

Засоби апаратної та програмної реалізації :

– мова програмування JavaScript;

– бібліотека React;

– програмна платформа Node.js;

– бібліотека Express;

– бібліотека Redux;

– HTTP client Axios;

– бібліотека Music meta-data ;

– бібліотека Nodemailer;

– бібліотека JWT;

– бібліотека Moongose;

– база даних MongoDB;

– операційна система Windows 10;

– бібліотека Ant design;

– браузер Google.

Вихідні дані :

– файл музики;

– файл обкладинки;

– список музики;

– список плейлістів;

– список бібліотек;

– список коментарів;

– список питань;

– список ролей;

– список жанрів;

– список користувачів.

1.4 Специфікація вимог

Призначення та межі проекту :

– призначення застосунку: популяризація композиторів, що мають малу аудиторію;

– погодження, що ухвалені в програмній документації: документація React, документація Redux, документація Express, документація moongoose;

– межі проєкту ПЗ: розробка системи рекомендацій, можливість підтримки відповідати на питання, можливість адміністратора видаляти, додавати і редагувати користувачів та музику.

Загальний опис:

– сфера застосування: мультимедійна сфера, IT сфера;

– характеристики користувачів: наявність смартфону, планшету або ПК та доступу до мережі Інтернет;

– загальна структура і склад системи: клієнт, сервер та база даних;

– загальні обмеження: користувач повинен бути зареєстрований для прослуховування музики, користувач має пам'яти електронну пошту та мати доступ до пошти.

Функції системи:

– опис функції: додавання, видалення, вподобання, прослуховування та редагування музики, генерація системи рекомендацій, додавання плейлістів та бібліотек;

– вхідна та вихідна інформація: список музики, список користувачів, список альбомів, список плейлістів.

Вимоги до програмного забезпечення:

– архітектура програмного забезпечення: клієнт – сервер;

– системне програмне забезпечення: ОС Windows, платформа node.js;

– мережне програмне забезпечення: Ubuntu;

– мова і технологія розробки ПЗ: мова розробки JavaScript, бібліотека React, Redux, Express, Moongoose.

Вимоги до зовнішніх інтерфейсів:

– інтерфейс користувача – веб-інтерфейс користувача, який приймає дані, забезпечує виведення даних та відображення React компонентів;

– програмний інтерфейс: RestFul API;

– комунікаційний протокол: TCP – IP.

Висновки до розділу 1

Під час дослідження предметної галузі були описані і проаналізовані структурні та функціональні особливості процесу конструювання вебзастосунку потокової передачі музики. Розглянуто переваги стримінгових сервісів.

Розглянуті застосунки аналоги, архітектуру, мову розробки, розробників які розробляли цей застосунок. Був проаналізований функціонал цих застосунків з функціоналом застосунку що розроблюється.

Розглянуто методи дистрибуції, поширення музики через стримінгові сервіси, кіно і ігрові застосунки. Було розглянуті додаткові вимоги до користувача, та поради реалізації деяких кроків. А також написано специфікацію вимог до ПЗ. В якій вказано межі проекту, обмеження у проекті, функції, архітектуру, вимоги до програмного забезпечення та вимоги до зовнішніх інтерфейсів.

2 ПРОЕКТУВАННЯ ВЕБЗАСТОСУНКУ

2.1 Архітектура вебзастосунку

Побудова схеми архітектури ПЗ

Для покращення зручності обслуговування та гнучкості застосунки часто діляться на кілька логічних рівнів. Рівень є абстракцією, призначеною для задоволення певної потреби бізнесу.

Кожному рівню надається певний набір обов'язків, і він може мати доступ лише до нижчого рівня або на тому самому рівні (тобто розділення завдань).

На практичному рівні цей низхідний підхід дозволяє розробникам легко організовувати свій код і змінювати деталі реалізації одного або кількох рівнів, не впливаючи на всю програму [6].

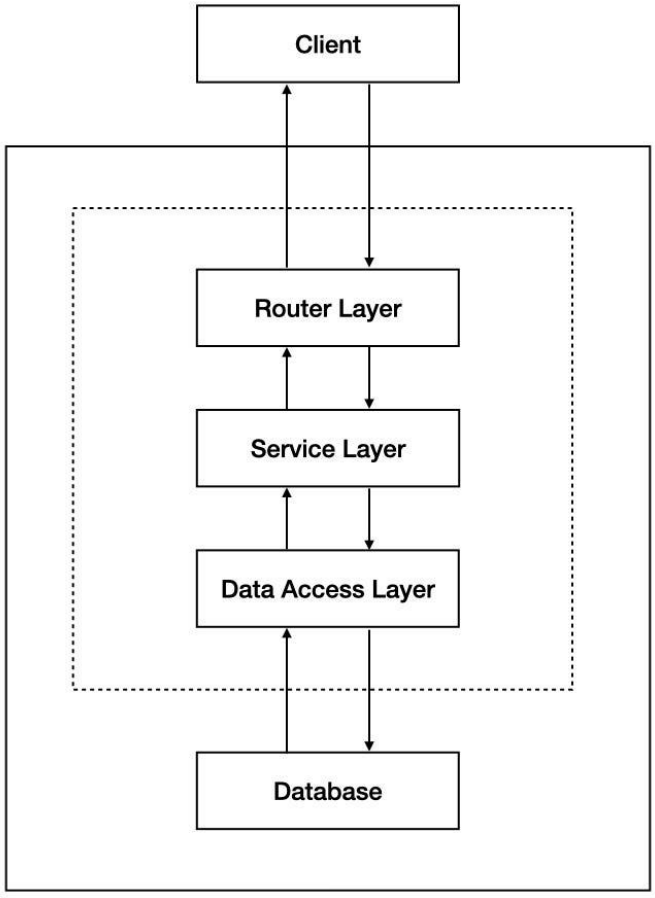

Рисунок 2.1 – Трирівнева архітектура

Клієнтська частина

Займає верхній рівень і відображає інформацію, пов’язану зі службами, які зазвичай доступні у веббраузері або вебзастосунку, у формі графічного інтерфейсу користувача (GUI).

Клієнт являє собою зовнішній рівень програми та інтерфейс, з яким кінцеві користувачі взаємодіють безпосередньо.

Цей рівень зазвичай побудовано на основі веб-розробки, наприклад CSS або JavaScript, і взаємодіє з іншими рівнями, надсилаючи результати в браузер та інші.

Серверна частина

Цей рівень — також званий середнім рівнем, рівнем логіки, бізнес-логіки або рівнем логіки — вилучається з рівня презентації.

Коли користувачі взаємодіють із веб-сайтом, вводячи вхідні дані, як-от клацання або введення тексту, серверна частина диктує програмування виходів, як-от текст, який з’являється на екрані. Внутрішня веб-архітектура визначає, як різні елементи коду взаємодіють для обробки вхідних даних від користувача та створення відповідних виходів.

Створюючи архітектуру на серверній частині веб-сайту, можливо створювати оптимізовані веб-сайти, на яких користувачі зможуть легко переміщатися сторінками та використовувати різні функції, не дивлячись на складний код і не використовуючи зовнішні пристрої. Це також дозволяє користувачам отримувати доступ до веб-сайтів без використання виключно обчислювальної потужності свого комп’ютера, що робить веб-сайти більш доступними.

Частина бази даних

Містить сервери баз даних, де інформація зберігається та отримується. Дані на цьому рівні зберігаються незалежно від серверів додатків або бізнес-логіки, а доступ до них здійснюється за допомогою програм, таких як MongoDB, Oracle, MySQL і Microsoft SQL Server.

2.2 Структура серверу

Рівень маршрутизатора

У трирівневій архітектурі рівень маршрутизатора є першим рівнем, який містить маршрути API програми та відповідає за [6]:

– розбір і перевірка корисного навантаження вхідних запитів, надісланих клієнтом, щоб позбавити їх будь-яких властивостей, специфічних для HTTP;

– пересилання проаналізованих даних на рівень обслуговування, відповідальний за обробку бізнес-логіки програми;

– перетворення результату виклику, зробленого на рівень обслуговування, у дійсну відповідь HTTP перед надсиланням назад клієнту.

Сервісний рівень

Сервісний рівень розташований між рівнем маршрутизатора та рівнем доступу до даних. Він абсолютно незалежний від будь-якого транспортного механізму, такого як HTTP або AMQP, що означає, що він може отримувати дані з багатьох джерел і ефективно їх обробляти.

Цей рівень містить бізнес-логіку програми та відповідає за:

– виконання завдань, пов’язаних із програмою, з використанням проаналізованих і підтверджених даних, надісланих рівнем маршрутизатора, відповідно до визначеного набору бізнес-правил (наприклад, створення маркерів нових сеансів, надсилання електронних листів тощо);

– виклик рівня доступу до даних у випадку, якщо йому потрібно зв’язатися із зовнішнім компонентом, таким як база даних.

Рівень доступу до даних

Рівень доступу до даних — це рівень, який відповідає за виконання операцій введення/виведення поза межами програми, наприклад зв’язок із базою даних для виконання операцій CRUD (тобто створення, читання, оновлення, видалення).

Одним із найпростіших способів досягти цього є використання ORM або ODM, бібліотеки, яка автоматизує передачу даних, в об’єкти, які частіше

використовуються в застосунках. Тому ORM забезпечує високорівневу абстракцію бази даних, що дозволяє розробникам писати об’єктно-орієнтований код замість запитів до бази даних, або збережених процедур для виконання операцій CRUD.

2.3 Опис дійових осіб

User – користувач авторизується у системі для пошуку, прослуховування, скачування, завантаження на сайт та видалення.

Support – підтримка авторизується у системі, для того щоб перевірити кількість питань, та відправити відповідь на ці питання.

Admin – адміністратор авторизується у системі, для редагування користувачів, видалення застарілих профілів, та перевірки даних в базі даних, а також для виправлення помилок у системі.

Compositor – копмозитор авторизується у системі, для загрузки нової музики та альбомів або їх видаленні.

2.4 Опис варіантів використання

Сценарії роботи системи:

– користувач хоче послухати музику на сайті (короткий сценарій): для цього користувач авторизувався на сайті під своїм обліковим записом, вибрав у пошуку потрібну музику, система видала потрібну музику, користувач увімкнув музику;

– користувач захотів послухати музику на сайті (поверхневий сценарій):

– головний сценарій (успішний): користувач ввів пароль та електрону пошту, після цього користувач успішно авторизувався на сайті під своїм обліковим записом, вибрав у пошуку потрібну музику, система успішно видала потрібну музику, користувач успішно увімкнув музику;

– альтернативні сценарії:

1. користувач ввів неправильний email;

2. користувач ввів нерпавельний пароль;

3. користувач забув пароль, треба сбросити пароль через e-mail;

4. користувача не зареєструвався;

5. користувач не підтвердив реєстрацію на пошті;

6. музики які ввів користувач у пошуку не існує;

7. при натисканні кнопки «Play», музика не грає.

– користувач захотів завантажити музику з сайту :

Scope: system of music service

Level: мета користувача (user-goal)

Primary Actor: user

Stakeholders and interests:

1. user: завантажити музику з сайта з якісним звуком;

2. support: допомогти зорієнтуватися та відповідати на скарги, запитання користувачів;

3. compositor: зацікавити користувача музикою;

4. admin: виправити технічні проблеми сайту.

Preconditions: користувач зайшов на сайт за посиланням

Main Success Scenario:

1. користувач ввів email та пароль;

2. система перевіряє правильність вводу та дозволяє користувача авторизуватися;

3. користувач авторизувався;

4. користувач переходить у поле пошуку;

5. користувач вводить у пошуку назву музики;

6. користувач вибирає жанр музики;

7. система перевіряє наявність музики;

8. користувач знаходить музику у списку пошуку;

9. користувач натискає на кнопку «завантажити»;

10. користувач обирає у якому форматі завантажити музику;

11. система завантажує файл, на пристрій користувача;

12. користувач покидає сайт.

Extensions:

a) користувач ввів неправильно email:

1) система запропонує перевірити правильність email;

2) система запропонує зареєструвати новий аккаунт;

3) система запропонує написати у підтримку;

b) користувач ввів неправильно пароль:

1) система запропонує перевірити правильність паролю;

2) система запропонує сбросити пароль;

3) система надсилає код підтвердження на email;

4) користувач заходить на email;

5) користувач знаходить лист з кодом:

5a) користувач не знаходить лист з кодом:

1a) лист потрапляє у спам;

1b) система не надсилала листа на пошту;

5b) користувач знаходить лист у спамі;

6) користувач вводить код;

7) система пропонує створити новий пароль;

8) користувач ввів новий пароль;

9) система пропонує підтвердити пароль;

10) користувач підтвердив пароль;

c) система не знайшла музику за назвою:

1) система запропонувала перевірити правильність написання музики;

2) система запропонувала знайти музику за жанром, датою, композитором:

2a) користувач вибирає музику за жанром;

2b) користувач вибирає музику за датою появи;

2c) користувач вибирає музику за композитором;

3) користувач знайшов музику;

d) музика не завантажується у користувача:

1) система перевіряє проблеми з мережею;

2) система перевіряє чи авторизований користувач;

3) система перевіряє чи існує посилання для завантаження:

3a) посилання не існує, помилка записується у звітність;

3b) запит на посилання успішний.

Кінець сценаріїв роботи системи.

2.5 Побудова діаграм UML

На рисунку 2.2 показана діаграма використання що демонструє як саме використовується система декількома дійовими особами.

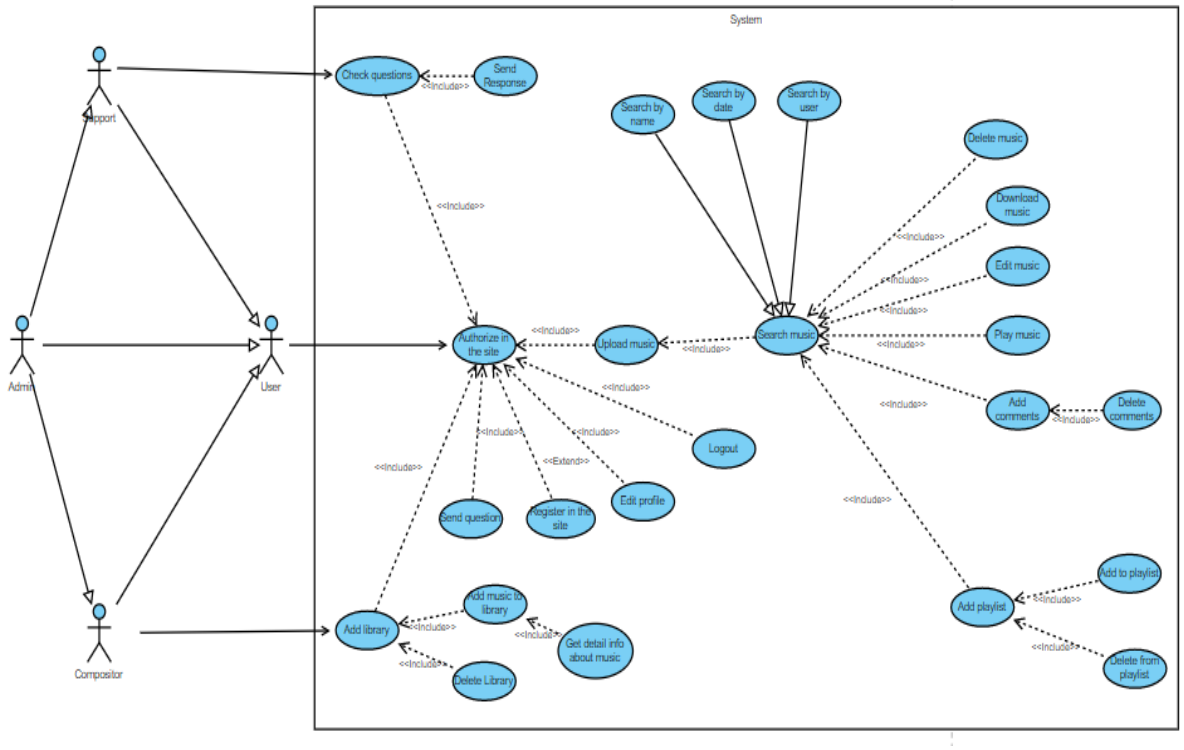

Рисунок 2.2 – Повна діаграма варіантів використання для інтернет-магазину

Побудова діаграм взаємодій

На рисунку 2.3 зображено діаграма послідовності «Пошуку музики», для цього користувачу треба авторизуватися, після цього система перевірить пароль та дозволить зайти на домашню сторінку. Користувач вводить ім'я музики, система перевіряє це ім'я, та повертає список користувачеві. Користувач має можливість відфільтрувати список, система повертає йому відфільтрований список. Користувач натискає на музику, система перенаправляє користувача на сторінку з вибраною музикую [7].

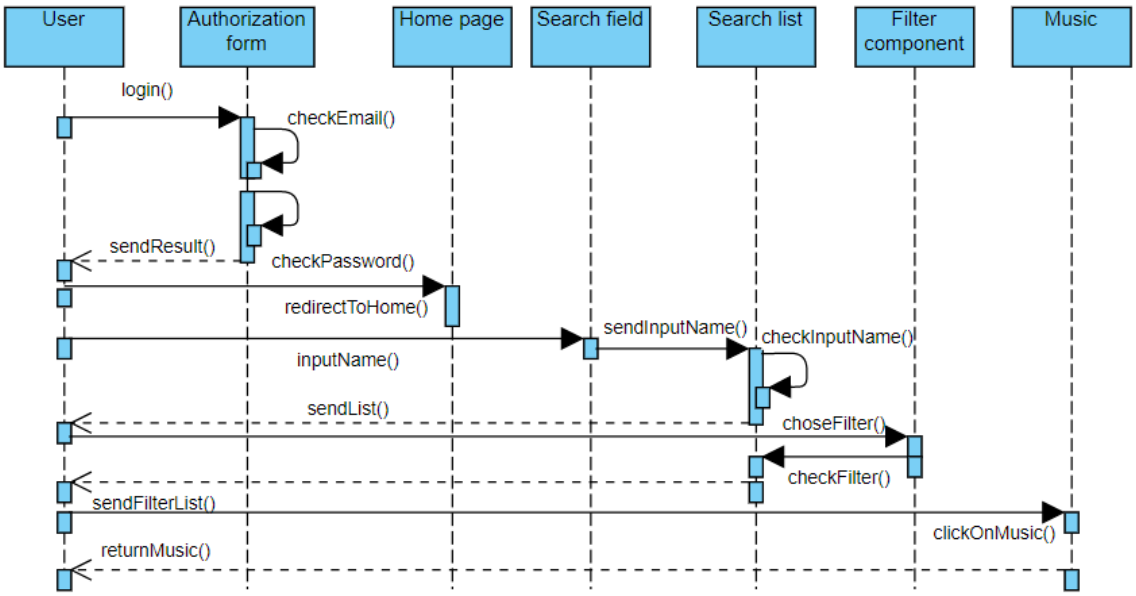

Рисунок 2.3 – Діаграма послідовності «Пошук музики»

На рисунку 2.4 зображується діаграма послідовності «Відповідь на питання користувача». Користувач надсилає на питання, система перевіряє на валідність та відправляє, питання у чат підтримці. Підтримка отримує питання та перевіряє обліковий запис користувача, після чого надсилає відповідь на питання, система перевіряє на валідність та надсилає відповідь користувачеві, а підтримці звіт що система успішно відправила відповідь [7].

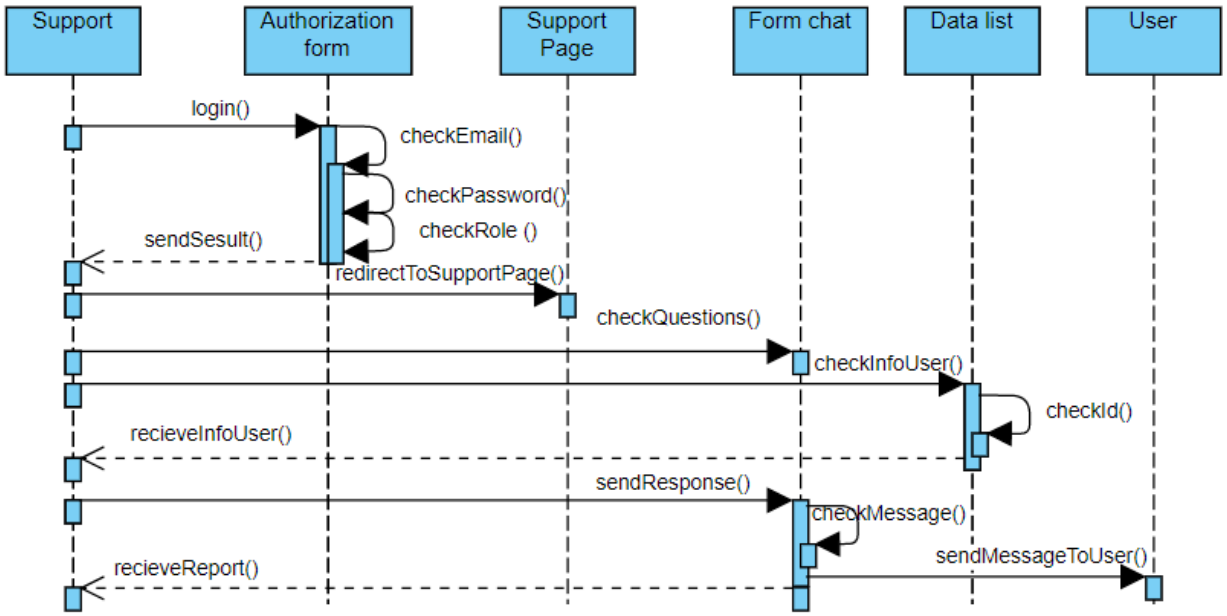

Рисунок 2.4 – Діаграма послідовності «Відповідь на питання користувача»

На рисунку 2.5 зображено діаграма послідовності «Додавання бібліотеки». Композитор авторизується, система перевіряє вірність введених даних, та надсилає відповідь композитору. Після авторизації, композитор переходить на сторінку композитора, після цього дивиться список бібліотек, система повертає список бібліотек. Композитор переходить до форми створення музикальної бібліотеки, вводить дані, система перевіряє введені дані, та повертає створену бібліотеку композитору.

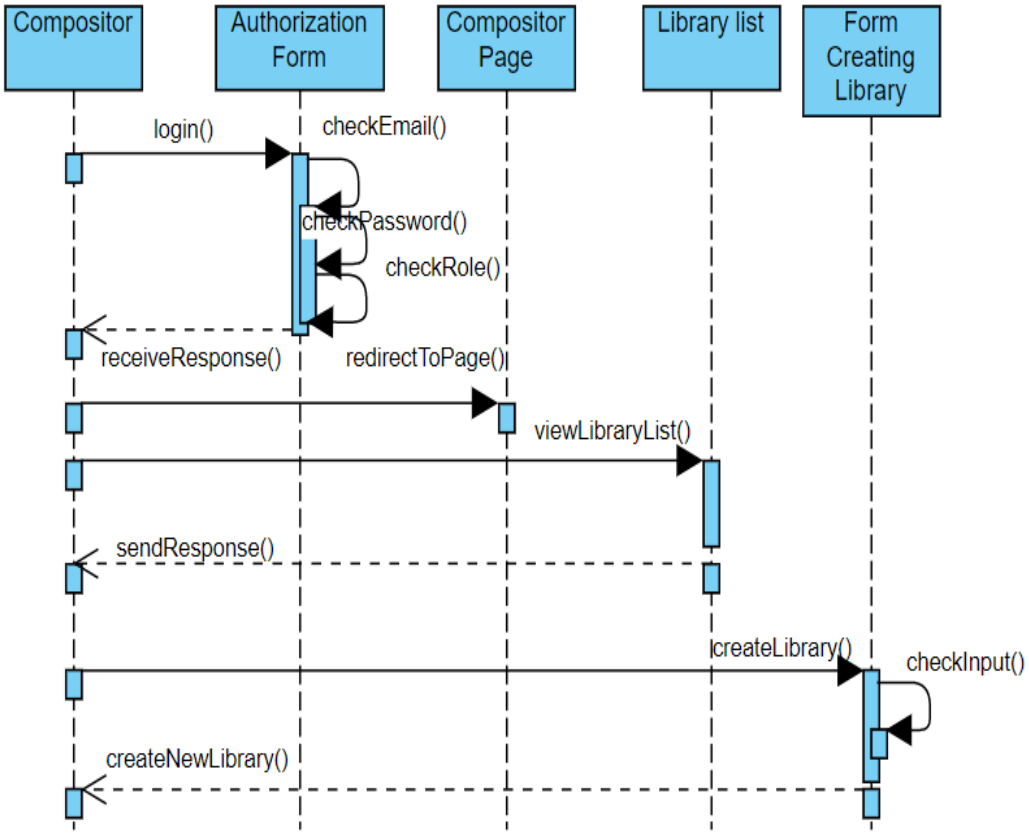

Рисунок 2.5 – Діаграма послідовності «Додавання бібліотеки»

Побудова діаграми станів та переходів

На рисунку 2.6 зображено діаграму станів та переходів. Коли користувач обирає форму вводу, він може вибрати що саме йому треба ввести, а також як відфільтрувати музику. Якщо введдені дані або вибрані опції не валідні, то користувач має змогу заново вести дані у поле вводу.

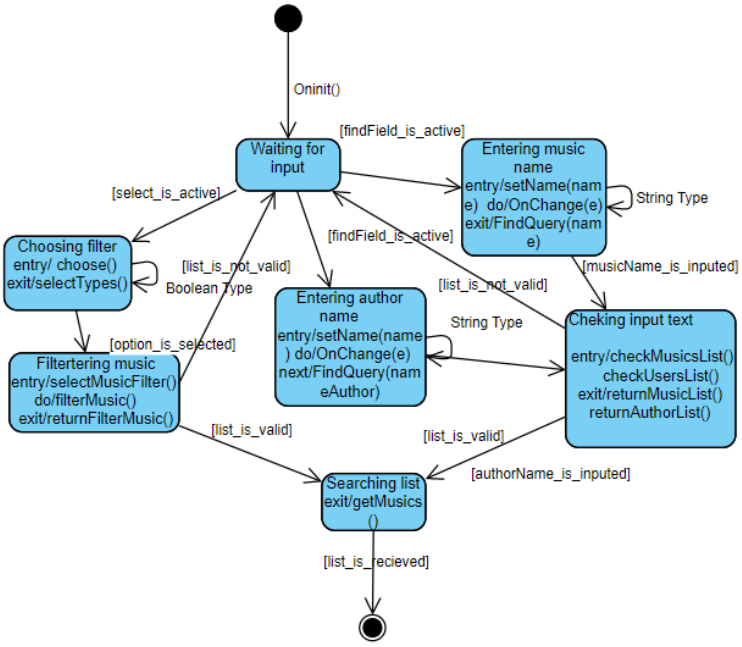

Рисунок 2.6 – Діаграма станів та переходів

Побудова діаграм діяльності

На рисунку 2.7 зображено діаграму діяльності авторизації користувача. Діаграма показує, що саме буде, якщо користувач не зміг авторизуватися по різним причинам, та можливість відновлення паролю завдяки електронній пошті.

Користувач повинен пройти кілька етапів відновлення паролю. Перший етап це введення своєї електронної пошти, наступний етап отримання коду на введену пошту та підтвердження за допомогою коду надісланого на пошту, та створення нового паролю.

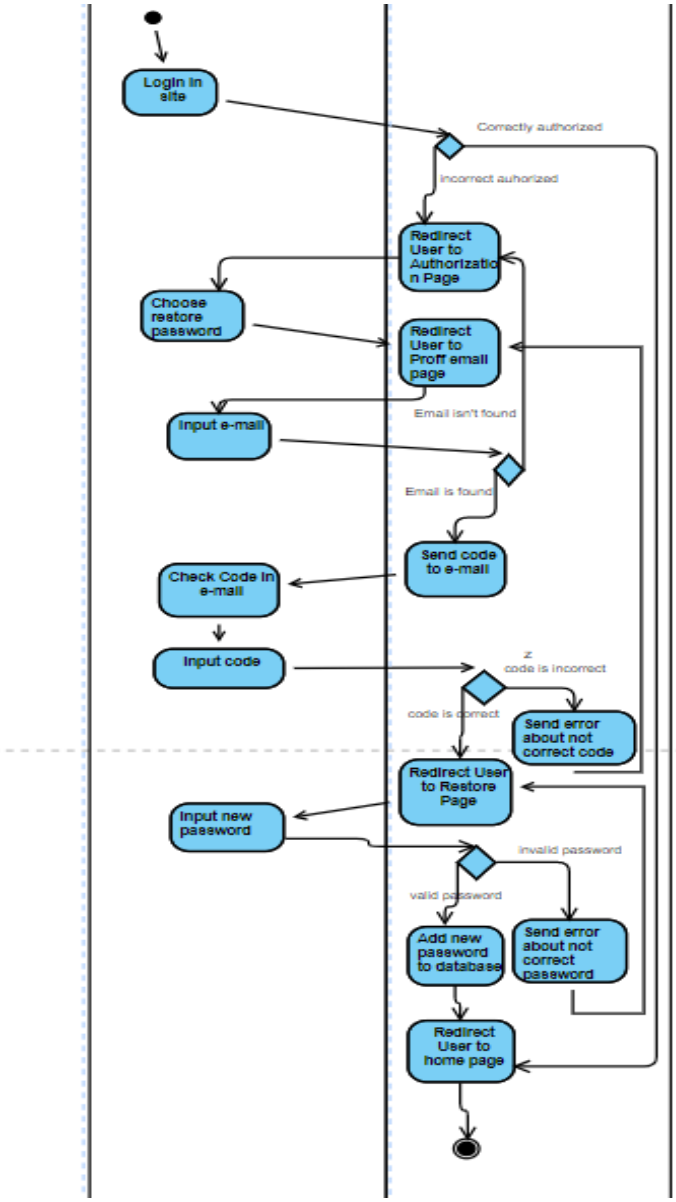

Рисунок 2.7 – Діаграма діяльності «Авторизація користувача»

Наступна діаграма зображена на рисунку 2.8, показає як саме користувач може шукати музику, і що саме буде якщо користувач не знайде потрібної музики. Користувач вводить дані про музику, цими даними можуть бути назва музики або назва автора цієї музики. Також користувач може фільтрувати дані за датою або за жанром. Якщо система не знайшла жодної музики, то система виведе пустий список, або повідомлення що потрібна інформація не знайдена.

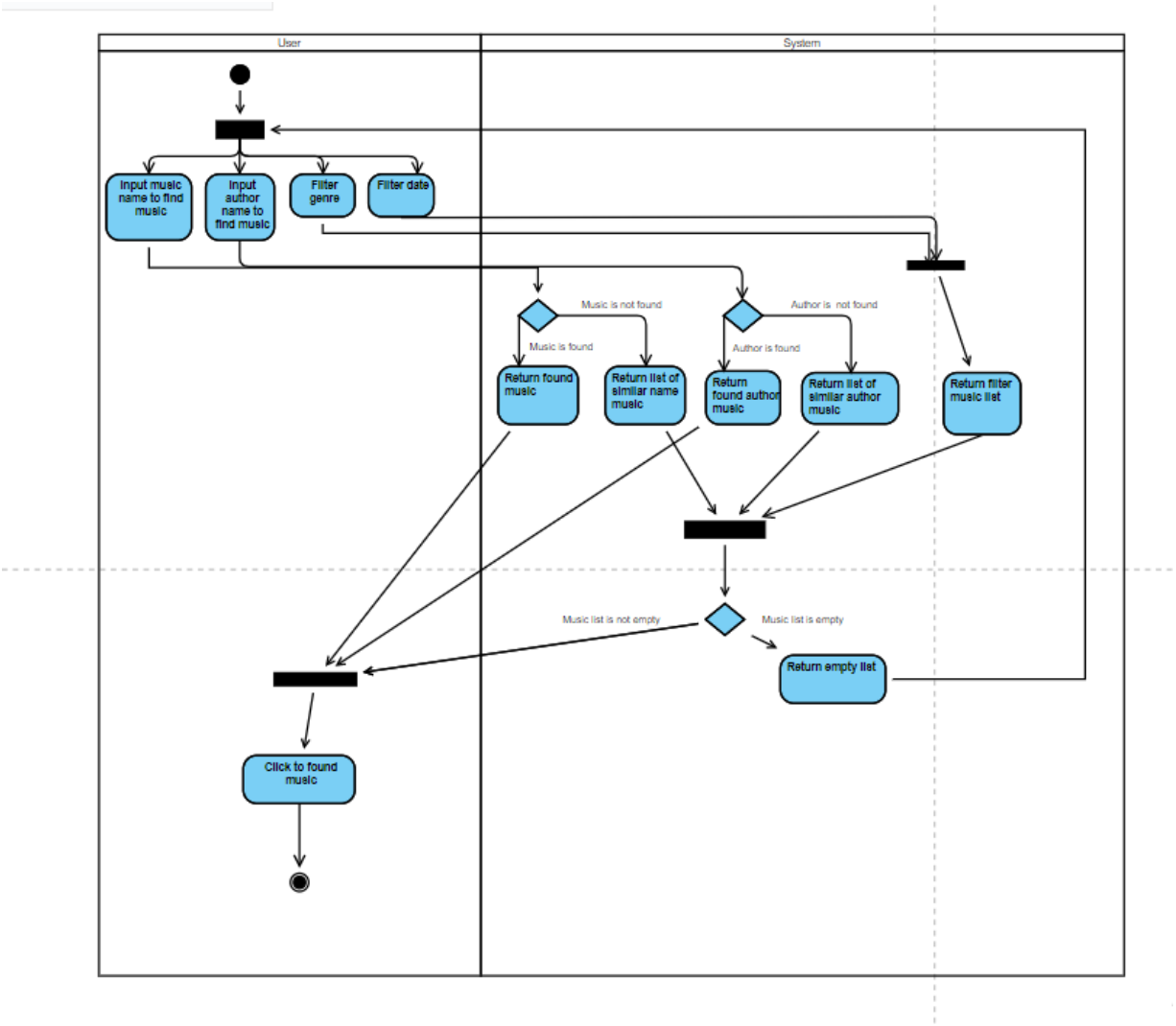

Рисунок 2.8 – Діаграма діяльності «Пошуку музики»

Наступна діаграма зображена на рисунку 2.9, показає як саме користувач взаємодіє з підтримкою та системою , коли треба задати питання. Введений текст перевіряється на коректність, як збоку користувача так і з боку підтримки. Підтримка бачить питання в списку питань, та відповідає на нього. Користувач отримує відповідь на це питання .

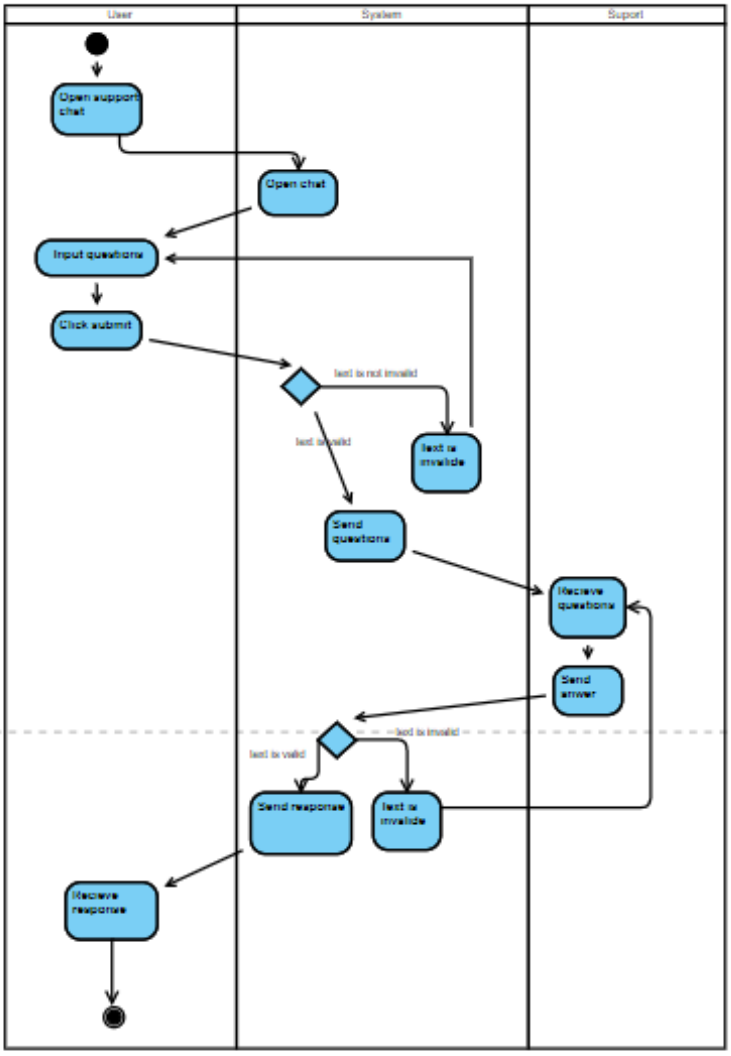

Рисунок 2.9 – Діаграма діяльності «Користувач задає питання»

Побудова діаграми пакетів

Діаграма пакетів демонструє взаємодію пакету сервера та клієнта. На сервері в пакеті routers, вказано endpoints, при виклику запиту на ці endpoints виконується певні проміжні функції, після цього запит обробляється контролером, далі сервісом, а сервіс взаємодіє з моделлю. Після обробки запиту сервер відправляє відповідь клієнту. В actions отримані дані записуються в глобальне сховище.

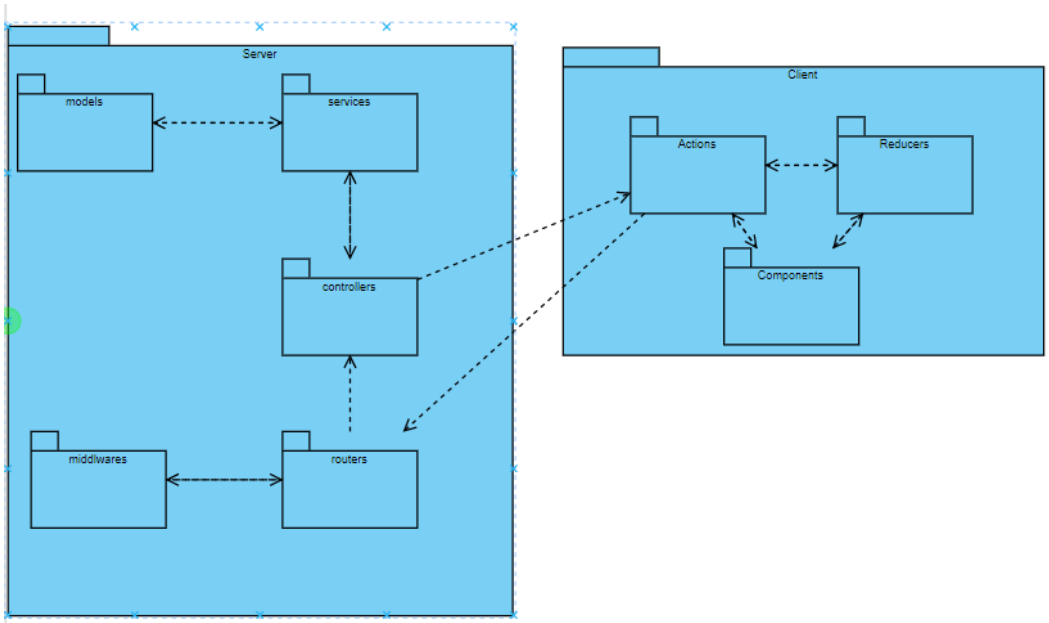

Рисунок 2.10 – Діаграма пакетів

Побудова діаграми розгортання

В якості клієнту виступає пристрій користувача. На пристрої є веб браузер, на якому можливо завантажити сайт. Для взаємодії з сервером, використовується HTTP протокол. Сервер написаний на node.js з використанням бібліотеки express. А також для взаємодії з сервером баз даних використовується ODM – бібліотека moongoose та протокол TCP/IP.

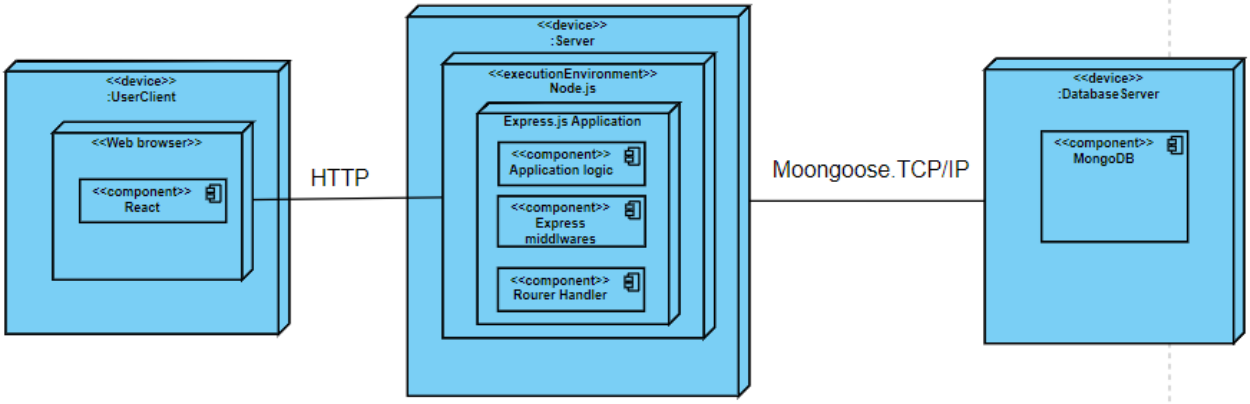

Рисунок 2.11 – Діаграма розгортання

Висновки до розділу 2

Під час виконання другого розділу було розглянуто архітектуру, бібліотеки, мову програмування програмного забезпечення. Було розглянуто та побудовано декілька діаграм мови UML.

Перша діаграма – діаграма послідовності , яка показує життєвий цикл взаємодій акторів та інформаційної системи. Друга діаграма класів , яка демонструє усю ієрархію класів. Третя діаграма станів та переходів визначає змінну стану об'єктів у часі. Четверта діаграма – діаграма діяльності, що показає основні дії що показані на діаграмі станів та переходів. П'ята діаграма пакетів показана для демонстрації пакетів програмного забезпечення. Та шоста діаграма розгортання яка описує зв'язки фізичних компонентів та їх розподілення.

У результаті продемонстровано архітектуру, бібліотеки та технології програмного забезпечення. А також побудовано 10 діаграм UML для проекту «Вебзастосунок потокової передачі музики».

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Технології розробки та мова програмування

Node.js

Node – це програмна платформа, яка за допомогою спеціальних інструментів, Node дозволяє перетворювати js у машинний код.

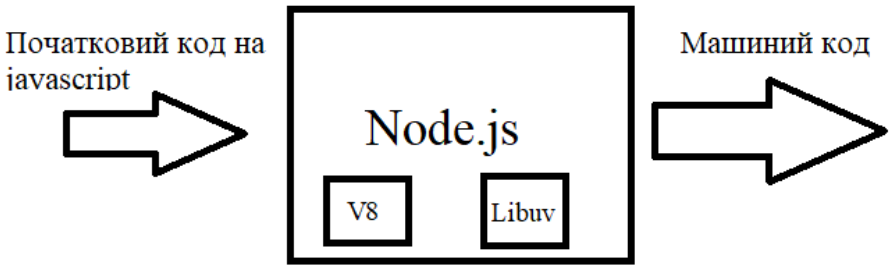

Рисунок 3.1 – Схема роботи Node

JavaScript

JavaScript – це мова програмування, яку розробники використовують для створення інтерактивних вебсторінок. Функції JavaScript можуть покращити зручність взаємодії користувача з вебсайтом: від оновлення стрічки новин у соціальних мережах до відображення анімації та інтерактивних карт. JavaScript є мовою програмування розробки скриптів для виконання на стороні клієнта, що робить його однією з базових технологій у всесвітній мережі Інтернет.

3.2 Вибір компонентів ПЗ

React

React.js – це бібліотека JavaScript з відкритим вихідним кодом, яка використовується для створення інтерфейсів користувача спеціально для односторінкових програм. Також використовується для обробки шару перегляду для веб-програм і мобільних додатків. React також дозволяє створювати багаторазові компоненти інтерфейсу користувача [8].

У React набір незмінних значень передається засобу відтворення компонентів як властивості в його тегах HTML. Компонент не може безпосередньо змінювати будь-які властивості, але може передавати функцію зворотного виклику, за допомогою якої можемо вносити зміни.

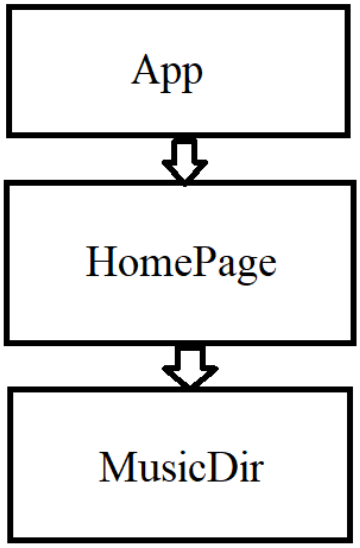

Рисунок 3.2 – Структура компонентів у React

Redux

Бібліотека Redux – це спосіб керування станом програми. Ключові моменти Redux [9]:

– сховище (store): зберігає стан програми;

– дії (actions): деякий набір інформації, що походить від додатку до сховища і який вказує, що саме потрібно зробити, для передачі у сховища викликається метод dispatch();

– творці дій (action creators): функції, що створюють дії;

– reducer: функція (або кілька функцій), яка отримує дію і відповідно до цієї дії змінює стан сховища.

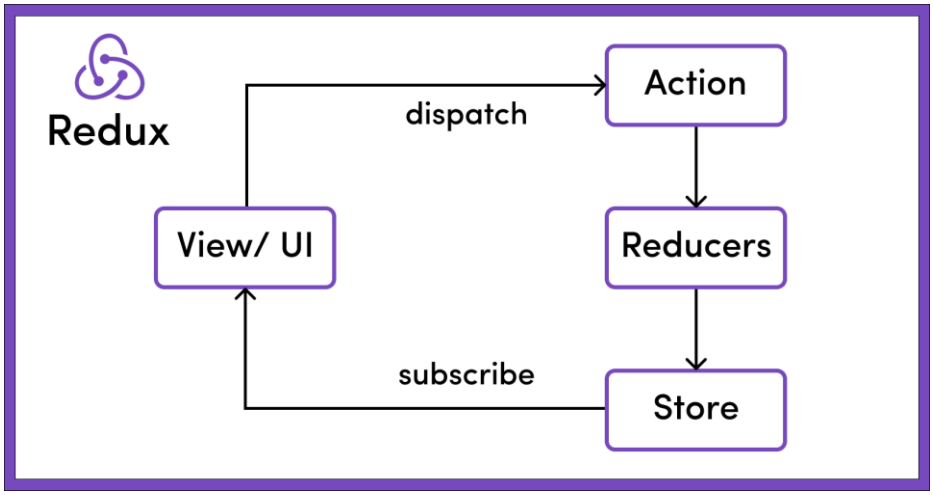

Рисунок 3.3 – Структура роботи бібліотеки Redux

Бібліотека Express

Express.js є найпопулярнішим веб-фреймворком для Node.js. Бібліотека призначена для створення веб-додатків і API і була названа де-факто стандартною серверною структурою для Node.js.

Створення бекенду з нуля для програми на Node.js може бути виснажливим і трудомістким. Від налаштування портів до обробників маршрутів, написання всього шаблонного коду позбавляє того, що справді має значення, а саме написання бізнес-логіки для програми.

Бібліотека Moongoose

Mongoose – це засіб відображення об’єктних документів (ODM). Це означає, що Mongoose дозволяє визначати об’єкти зі строго типізованою схемою, яка відображається в документі MongoDB.

Mongoose надає неймовірну кількість функціональних можливостей для створення та роботи зі схемами.

3.3 Розробка серверної частини

За допомогою Node.js є можливість розробити вебсервер та обробляти HTTP запити з клієнта і взаємодіяти з операційною системою яка знаходиться на серверному пристрої. Для запуску самого серверу використовується менеджер пакетів npm та інструмент розробника nodemon. Nodemon – це інструмент, який допомагає розробляти програми на основі Node.js шляхом автоматичного перезапуску програми node, коли виявляються зміни файлів у каталозі. Сам запуск серверу налаштовується у конфігураціях [10].

```
"scripts": {
  "start": "node index.js",
  "dev": "nodemon index.js"
},
```

Рисунок 3.4 – Налаштування запуску сервера

Використовуючи бібліотеку Express можливо створити обробники для запитів з різними HTTP методами для різних маршутів URL посилань. Також налаштування портів підключення до серверу та до бази даних.

За допомого функції use є можливість покласти функцію у конвейр функцій проміжної обробки:

```
const app=express()
const PORT =config.get("serverPort")

app.use(fileUpload({}))
app.use(corsMiddleware)
app.use(express.json())
app.use(express.static('static/avatars'))
app.use(express.static('static/musicImg'))
app.use(express.static('static/albumsImg'))
app.use("/api/auth/edit-user",authRouter)
app.use("/api/auth/avatar",authRouter)
app.use("/api/auth",authRouter)
app.use("/api/musics",musicRouter)
app.use("/api/genres",genreRouter)
app.use("/api/playlists",playlistRouter)
app.use("/api/albums",albumRouter)
app.use("/api/roles",roleRouter)
```

За допомогою бібліотеки moongoose можливо взаємодіяти з базою MongoDB [11]. Для підключення до бази даних використовується метод connect. А для запуску серверу на певному порті використовується функція listen:

```
const start = async () => {
  try{
    await mongoose.connect(config.get("dbUrl"))

    app.listen(PORT,()=>{
      console.log("Server started on PORT", PORT)
    })
  }
  catch (e){
    console.log(e)
  }
}
start()
```

Для того щоб отримувати дані с бази даних треба створити моделі, також за допомогою бібліотеки mongoose:

```
const {model,Schema,ObjectId}=require('mongoose')

const Music=new Schema({
    name:{
        type:String, required:true
    },
    genre:[{type:String, ref:"Genre",default:""}],
    path:{type:String, default:""},
    time:{type:Number,default:0},
    description:{type:String,default: ""},
    date:{type:Date, default:Date.now()},
    image:{type:String,default:""},
    listens:[{type:ObjectId,ref:"User"}],
    likes:[{type:ObjectId,ref:"User"}],
    author:{type:ObjectId, ref:'User'},
    playlists:[{type:ObjectId,ref:"Playlist"}],
    album:{type:ObjectId,ref:"album"}
})

module.exports=model("Music",Music)
```

Одним з важливих аспектів це кінцеві точки за якими здійснюється запити на сервер з клієнта. За маршутизацію відповідає маршутизатор:

```
const Router = require("express")
const router = new Router()
const authMiddleware = require('../middleware/auth.middleware')
const musicController = require('../controllers/musicController')

router.post('',authMiddleware,musicController.createMusicDir)
router.post('/upload',authMiddleware,musicController.uploadMusic)
router.post('/comments',authMiddleware,
musicController.addCommentMusic)
router.post('',authMiddleware,musicController.fetchMusics)
router.patch('/edit/:id',authMiddleware,musicController.editMusic)
router.get('',authMiddleware,musicController.fetchMusics)
router.get('/like',authMiddleware,musicController.likeMusic)
router.get('/play',authMiddleware, musicController.playMusic)
router.get('/musicPage/:id',authMiddleware,
musicController.getMusic)
router.get('/download',authMiddleware,musicController.downloadMusi
c)
router.get('/search',authMiddleware,musicController.searchMusic)
router.get('/getAuthor',authMiddleware,musicController.findAuthorB
yId)
router.get("/comments",authMiddleware,musicController.getComments)
router.get("/recommends",authMiddleware,musicController.getRecomme
ndMusic)
router.get("/recommends/search",authMiddleware,musicController.sea
rchRecommendMusic)
router.get("/profile/:id",authMiddleware,musicController.getMusics
Profile)
router.get("/search/profile",authMiddleware,musicController.search
ProfileMusic)
router.delete('/',authMiddleware,musicController.deleteMusic)

module.exports=router
```

Наступним для перевірки даних потрібна проміжна функція або іншими словами middleware. Для перевірки токена доступу та нової генерації цього токена треба використати проміжну функцію. За допомогою JWT токена можливо безпечно передавати дані з серверу на клієнт.

Для створення необхідно визначити заголовок (header) із загальною інформацією по токену, корисні дані (payload), такі як id користувача, його роль та підписи (signature):

```
const jwt = require ("jsonwebtoken")
const config= require("config")

module.exports = (req,res,next)=>{
    if(req.method==='OPTIONS'){
        return next()
    }
    try {
        const token = req.headers.authorization.split(' ')[1]
        if(!token){
            return res.status(401).json({message:"Auth error"})
        }
        const decode=jwt.verify(token,config.get('secretKey'))
        req.user = decode
        next()
    } catch (e) {
        return res.status(401).json({message:'Auth error'})
    }
}
```

Далі створюється контролер який обробляє запити, та відправляє відповіді на клієнт. Одним з таких контролерів являється контролер музики. Та один з методів цього контролеру завантажує на сайт музику:

```
async uploadMusic( req , res ,next) {
    try {
        const music = req.body
        const dbMusic = await
musicService.uploadMusic(req.files,req.user.id,music)
        return res.json ( dbMusic )
    } catch ( e ) {
        next(e)
    }
}
```

У результі виконання методу музика завантажується на сайт та відправляє відповідь на клієнт в якості формату json .

У цьому методі контролера викликається метод сервісу музики. В методі сервісу виконується сама бізнес логіка та взаємодія з базою даних. Метод в якості параметрів приймає файли, id та музику як об'єкт. Метод findOne находить користувача який завантажив цю музику, якщо користувача не знаходить, тоді метод повертає помилку:

```
async uploadMusic( files , id , music ) {
    const user = await User.findOne ( { _id: id } )

    let path = await FileService.createMusicFile ( user ,
files.music );
    const imgName = await FileService.createMusicImage (
files.image )

    const time = await FileService.calculateDuration ( files.music
)

    const genresArray = await GenreService.findGenreByValue (
music )


    const dbMusic = new Music ( {
        name: music.name ,
        genre: genresArray ,
        path: path ,
        time: time ,
        image: imgName ,
        description: music.description ,
        author: user._id ,
    } )
    await dbMusic.save ()
    await user.save ()
    return dbMusic
}
```

У результаті виконується методи findOne для пошуку користувача, createMusicFile для створення музичного файлу, createMusicImage для створення обкладинки музики, calculateDuration для розрахунку часу тривалості музики та функція findGenreByValue для пошуку музики за жанром.

```
async createMusicImage(file){
    if(file) {
        const imgName = UUid.v4 () + ".jpg"
        file.mv ( config.get ( 'staticMusicImg' ) + "\\" + imgName
)
        return imgName
    }
    return ""
}
```

Для створення музичного файлу в директорії користувача, треба спочатку перевірити тип файлу та отримати шлях до директорії. Після цього якщо шлях

існує то видати помилку. Якщо шляху не існує тоді створюємо файл та перемістимо його до директорії, як це продемонстровано в методі:

```
async createMusicFile(user,file){
    const type = file.name.split ( '.' ).pop ()
    if (type === "mp3" | type === "wma" | type === "mp2" || type
=== "amr") {
        let path
        path = `${ config.get ( 'musicPath' ) }\\${ user._id }\\${
file.name }`

        if (fs.existsSync ( path )) {
            throw new ApiError.BadRequest ( "music already exist"
)
        }
        file.mv ( path )
        return path
    } else {
        throw new ApiError.BadRequest ( "User has tried upload
incorrect format" )
    }
}
```

Для розрахунку часу тривалості музичного файлу, треба зчитати мета дані з цього файлу за допомогою бібліотеки music-metadata:

```
async calculateDuration(musicFile){
    const metadata = await (
        await metadataMusic).parseBuffer(musicFile.data,
'audio/mpeg');
    const duration = await metadata.format.duration
    return duration
}
```

У результаті виконання методу повертається тривалість часу музики у мілісекундах.

Для створення рекомендацій, треба написати метод генерації рекомендацій. Спочатку перевіряється середня кількість прослуховувань. І весь масив знайдених пісень ділиться на дві частини. Перша частина це частина масиву, яка має меншу кількість прослуховування ніж середня кількість, а друга частина має більшу кількість прослуховування. У той частини що мають більшу кількість прослуховування, більше вірогідність першою потрапити до системи рекомендацій. Далі метод перевіряє, які саме жанри і музику віддає перевагу користувач. Якщо користувач заходить вперше тоді система генерує новий масив рекомендованої музики. За ці дії відповідає метод generateRecommendMusic:

```
async generateRecommendMusic( userId ) {
    const allMusics = await musicService.getAllMusic ()
    let pullRecommendations = new Set ()
    const avgListens = await Music.aggregate ( [{
        $group: {
            _id: null , avgLength: { $avg: { $size: "$listens" } }
        }
    }] )
    const total = await Music.countDocuments ( {} )
    while(pullRecommendations.size<total) {
        allMusics.map ( async ( music ) => {
            if (this.checkMusicToAdd ( music.listens.length <=
avgListens[0].avgLength , 25 )) {
                pullRecommendations.add ( music._id )
            }
            if (this.checkMusicToAdd ( music.listens.length >=
avgListens[0].avgLength , 15 )) {
                pullRecommendations.add ( music._id )
            }
            if (music.listens.includes ( userId ) &&
music.likes.includes ( userId )) {
                const genres = music.genres
                const author = music.author
                const recommendMusics = await Music.find ( {
genres: { $in: genres } , author: author } )

                if (Math.floor ( Math.random () * 100 ) < 50) {
                    recommendMusics.map ( recommendMusic => {
                        pullRecommendations.add (
recommendMusic._id )
                    } )
                }
            }
        } )
    }
    const newArrayPullRecommendations = Array.from (
pullRecommendations )

    return newArrayPullRecommendations
}
```

У цьому методі перевіряється яку музику вподобав користувач і яку музики треба згенерувати користувачу для прослуховування.

Для отримання рекомендованої музики користувачу, виконується метод getRecommendMusic, який якості параметру приймає id користувача, сторінку, ліміт повернення даних, жанри та метод сортування даних.

```
async getRecommendMusic(userId , page , limit, genres,sort){
    const total = await Music.countDocuments ( {} )
    const recommends = await Recommends.findOne ( { user: userId }
)
    let generatedMusics = recommends.musics
    const now = Date.now()
    const hours = (now - recommends.updateDate)/3600000;
    if(hours>2) {
        generatedMusics = await this.createRecommendMusic(userId,
page, limit)
    }
    const musics = await Music.find({_id:{$in:generatedMusics},
genre:{$in:genres}}).sort(sort).skip ( page * limit ).limit (
limit )
    return {
        musics:musics,
        total:total
    }
}
```

Для створення рекомендацій є метод createRecommendMusic:

```
async createRecommendMusic( userId , page , limit ) {
    const recommends = await Recommends.findOne ( { user: userId }
)
    if (! recommends) {
        const musics = await this.generateRecommendMusic ( userId
)
        let recommend = new Recommends ( {
            musics: musics ,
            user: userId
        } )
        await recommend.save ()
        return  recommend.musics.slice ( page * limit , (
                page + 1) * limit )
    }
    const now = Date.now()
    const musics = await this.generateRecommendMusic ( userId )
    await
Recommends.findOneAndUpdate({user:userId},{musics:musics,updateDat
e:now})

    return recommends.musics
}
```

В результаті користувач отримує рекомендовано музику.

3.4 Розробка клієнтської частини

Використовуючи бібліотеку React є можливість динамічно змінювати стан елементів сторінки. У реакті функціональні компоненти повертаються в форматі jsx. Цей формат означає що у html коді є можливість використати javascript. Компонент продемонстрований далі:

```
const MusicDir =
({applyFilter,search,musicIsLoaded,searchName,sort,checkedList,set
CheckedList,setSort}) => {
    const dispatch = useDispatch()
    const page = useSelector(state => state.musics.page)
    const totalCountPages =
useSelector(state=>state.musics.totalPage)

    const changePage = (page) => {
        dispatch(setPage(page))
    }

    return (

            <div className="body wrapper clear">
                <div className="content">
                    <div style={{display:"flex"}}>
                        <div className="search-block">
                            <img width={20} height={20}
src={FindLogo} alt="Search"/>
                            <MyInput
style={{borderRadius:"50px",marginLeft:"2px"}} value={searchName}
onChange={e=>search(e)} className="find-input"
placeholder="Search..."/>
                            <FilterMusic applyFilter={applyFilter}
checkedList={checkedList} setCheckedList={setCheckedList}/>
                        </div>
                        <SelectMusics sort={sort}
onChange={(value)=>setSort(value)}/>
                        <NavLink className="upload" to="/upload">
<PlusCircleOutlined  style={{color:"black",fontSize:35,
marginTop:"10px"}}/> </NavLink>
                    </div>
                    {musicIsLoaded ? <Loader/> : <MusicList/>
                    }
                </div>
                <Pagination defaultCurrent={page+1}
onChange={(page) => changePage(page-1)}
total={totalCountPages*10} />
            </div>
    );};
```

Використовуючи бібліотеку Redux , яка надає розробнику більший спектр дій, тобто більше вільної взаємодій з компонентами. Redux створює глобальні змінні, з якими можливо взаємодіяти у будь-якому компоненті або точці застосунку. Взаємодіяти з цими змінами можливо через виклик дій. Дія може змінити стан цих глобальних змін або об'єктів. Самі дії зберігаються у редюсері. Так чином є можливість розділити логіку програми на декілька частин. Один з прикладів редюсерів, редюсер музики продемонстрований:

```
export default function musicReducer(state=defaultState, action) {
    switch (action.type) {
        case SET_MUSICS : return {...state,musics: action.payload}
        case GET_AUTHOR: return {...state,currentAuthor:
action.payload}
        case ADD_MUSIC: return {...state, musics:
[...state.musics, action.payload]}
        case GET_MUSIC: return {...state, currentMusic:
action.payload}
        case UPDATE_MUSIC: return {...state,musics:
[...state.musics.map((music)=>
                music._id === action.payload._id ? music =
action.payload : music)]}
        case DELETE_MUSIC: return
{...state,musics:[...state.musics.filter(music=>music._id!==action
.payload)]}
        case SET_PAGE: return {...state,page: action.payload}
        case SET_TOTAL_PAGE : return
{...state,totalPage:action.payload}
        default:
            return state
    }
}
```

У результаті виконання будь-яких подій повертається змінений глобальний стан об'єкта. Для зберігання усіх редюсерів в єдиний системі, і щоб можливо було дізнатися де зберігається та чи інша змінна чи за що відповідає певна дія. Для цього усі дані редюсери зберігаються у головному редюсері, який має назву «rootreducer». Методом createStore створюється глобальне сховище, першим параметром вказується головний редюсер, другим параметром передається метод composeWithDevTools це покращений метод compose, який автоматично додає інструменти розробки, а метод applyMiddleware приймає в якості параметру middlware. Thunk є middleware для асинхронних дій.

```
const rootReducer= combineReducers({
    user:userReducer,
    musics:musicReducer,
    player:playerReducer,
    comments:commentReducer,
    message:messageReducer,
    genre:genreReducer,
    playlist:playlistReducer,
    roles:roleReducer,
    albums:albumReducer
})
export const store =
createStore(rootReducer,composeWithDevTools(applyMiddleware(thunk)
))
```

Використовуючи HTTP сервер axios для відправки HTTP запитів на сервер.

За допомогою методу getMusics, отримаємо список музики в якості json формату та dispatch який викликає певний діє створювач .

```
export function getMusics( sort , limit, page,genres ) {
    return async dispatch => {
        try {
            let url = `http://localhost:5000/api/musics`
            if (sort) {
                url = `http://localhost:5000/api/musics?sort=${
sort }`
            }
            const response = await axios.get ( url , {
                headers: {
                    Authorization: `Bearer ${ localStorage.getItem
( 'token' ) }`
                } ,
                params:{
                    limit:limit,
                    page:page,
                    genres:genres
                }
            } )
            dispatch ( setMusics ( response.data.musics ) )

dispatch(setTotalPage(getPageCount(response.data.total,limit))
} catch ( e ) {console.log ( e.response.data.message )}}}
```

За допомогою методу uploadMusic можливо завантажити музику на вебзастосунок, в якості аргументу приймається файл музики, музична обкладинка, музичні жанри, назва та опис музики.

У заголовках в якості параметра Authorization передаємо токен доступу. Який зберігається в локальному сховище.

```
export function uploadMusic( musicFile , musicImage, musicGenres ,
musicName , musicDescription ) {
    return async dispatch => {
        try {
            const formDataMusic = new FormData ()
            formDataMusic.append ( 'music' , musicFile )
            formDataMusic.append ( 'image' , musicImage )
            formDataMusic.append("genres",musicGenres)
            formDataMusic.append ( "name" , musicName )
            formDataMusic.append ( "description" ,
musicDescription )

            const response = await axios.post (
'http://localhost:5000/api/musics/upload' , formDataMusic,{
                headers: { Authorization: `Bearer ${
localStorage.getItem ( 'token' ) }` }
            } );

            dispatch ( addMusic ( response.data ) )

        } catch ( e ) {
            alert ( e.response.data.message )
        }
    }
}
```

За допомогою методу downloadMusic, користувач може завантажити з сайту музику.

```
export async function downloadMusic( music ) {
    try {
        const response = await fetch (
`http://localhost:5000/api/musics/download?id=${ music._id }` , {
            headers: {
                Authorization: `Bearer ${ localStorage.getItem (
"token" ) }`
            }
        } )
        if (response.status === 200) {
            const blob = await response.blob ();
            const downloadUrl = window.URL.createObjectURL ( blob
);
            const link = document.createElement ( 'a' );
            link.href = downloadUrl
            link.download = music.name
            document.body.appendChild ( link )
            link.click ()
            link.remove ()
        }

    } catch ( e ) {
        alert ( e.response.data.message )}
}
```

За допомогою методу playCurrentMusic, користувач може програвати музику з сайту. Так само за допомогою методу fetch та посиланням на адрес сервера отримаємо інформацію про музику яка передається в якості параметру id. Також в якості заголовків передаємо токен доступу для перевірки. Функцією dispatch викликаємо дію setLink, яка записує посилання в глобальну змінну:

```
export function playCurrentMusic( music ) {
    return async dispatch => {
        try {
            const response = await fetch (
`http://localhost:5000/api/musics/play?id=${ music._id }` , {
                headers: { Authorization: `Bearer ${
localStorage.getItem ( 'token' ) }` }
            } )
            const blob = await response.blob ();
            const downloadUrl = window.URL.createObjectURL ( blob
);
            dispatch ( setLink ( downloadUrl ) )
        } catch ( e ) {
            console.log ( e )
        } }
}
```

Для отримання рекомендованої музики користувачу, викликається метод getRecommendMusics який приймає у параметрах метод сортування, сторінку, ліміт даних та фільтрацію по жанрам:

```
export function getRecommendMusics(sort,page,limit, genres){
    return async dispatch =>{
        try {
            const response = await axios.get (
`http://localhost:5000/api/musics/recommends` , {
                headers: {
                    Authorization: `Bearer ${ localStorage.getItem
( "token" ) }`
                },
                params:{
                    sort:sort,
                    limit:limit,
                    page:page,
                    genres:genres
                }
            } )
            dispatch(setMusics(response.data.musics))

dispatch(setTotalPage(getPageCount(response.data.total,limit)))
        } catch ( e ) {
            alert ( e.response.data.message )
        }}}
```

В результаті виконання функцій, виконується функція dispatch, яка викликає дію setMusics, ця дія привласнює музику отриману з сервера.

Головний компонент який відповідає за всі компоненти у проекті на клієнті:

```
const App = () => {

    return (
        <BrowserRouter>
            <div className="app">
                <Navbar/>
                <LoaderHome/>
                <AppRouter/>
                <BottomBar/>
            </div>
        </BrowserRouter>
    );
}
```

Також для щоб користувач міг переходити за різними посиланням треба налаштувати компонент маршутизації. Який буде перевіряти чи авторізований користувач, а також якої ролі користувач.

За допомогою хука useEffect, який запускає рендер сторінки, при зміні значення в квадратних дужках, та методу navigate який перенаправляє користувача на іншу сторінку, коли той авторизувався:

```
useEffect( () => {
    dispatch ( auth () )
    if(isAuth){
        navigate("/home")
    } else {
        navigate("/login")
    }
} , [isAuth])
```

Коли користувач заходить на сайт. Якщо він не авторизований, то йому потрібно авторизуватися. А поки система маршутизації йому не дасть на іншу сторінку. Також якщо роль користувача не відповідає маршуту який вказав користувач, тоді система перекинить на домашню сторінку користувача.

За маршутизацію відповідає головний маршутизатор:

```
const AppRouter = () => {
    const dispatch = useDispatch ()
    const isAuth = useSelector(state=>state.user.isAuth)
    const isAdmin = useSelector(state=>state.user.isAdmin)
    const navigate = useNavigate ()

    useEffect( () => {
        dispatch ( auth () )
        if(isAuth){
            navigate("/home")
        } else {
            navigate("/login")
        }
    } , [isAuth])
    return (
        <div>
            { isAuth ?
                <Routes>
                    <Route exact path="/home" element={ <Home/>
}/>
                    <Route exact path="/editUser" element={
<EditUser/> }/>
                    <Route path="/upload"
element={<UploaderPage/>}/>
                    <Route exact path="/recommends"
element={<Recommends/>}/>
                    <Route exact path="/albums"
element={<Albums/>}/>
                    <Route exact path="/createAlbums"
element={<FormAddAlbum/>}/>
                    <Route path="musics/:id"
element={<MusicId/>}/>
                    <Route path="editMusic/:id"
element={<EditMusic/>}/>
                    <Route path="profile/:id"
element={<ProfileUser/>}/>
                    <Route path="editProfile/:id"
element={<EditUserPanel/>}/>
                    {isAdmin && <Route path="/panel"
element={<AdminPanel/>} />}
                </Routes>
                :
                <Routes>
                    <Route path="registration" element={
<Registration/> }/>
                    <Route path="/login" element={ <Login/> }/>
                </Routes>
            }
        </div> );
};
```

Довге опитування (long polling) потребує опитування запитів та відповідей HTTP і робить його ефективнішим, оскільки повторні запити до сервера витрачають ресурси. Наприклад, встановлення нового з'єднання, аналіз HTTP-заголовків, запит на отримання нових даних, генерація та доставка відповіді та, нарешті, закриття та очищення з'єднання.

Щоб уникнути цих зусиль, під час тривалого опитування сервер вирішує утримувати з'єднання клієнта відкритим якомога довше та надає відповідь, коли нові дані стають доступними або якщо досягнуто порогового значення часу очікування. Коли час спливає виконується реверсивна функція:

```
export function subscribe(){
    return async dispatch =>{
        try {
            const response = await
axios.get('http://localhost:5000/api/messages/getMessage',{
                headers: {
                    Authorization: `Bearer
${localStorage.getItem("token")}`
                }
            })
            dispatch(addMessage(response.data))
           await subscribe()
            console.log(response.data)
        } catch (e) {
            await setTimeout(()=>{
                 subscribe()
            },500)
        }
    }
}
```

Метод підписки додає повідомлення користувачу, коли той надіслав запитання в чат підтримки.

3.5 Тестування програмного забезпечення

Для демонстрації роботи програмного забезпечення треба авторизуватися. На наступних рисунках продемонстровано реєстрація та авторизація на сайті.

Рисунок 3.5 – Реєстрація на сайті

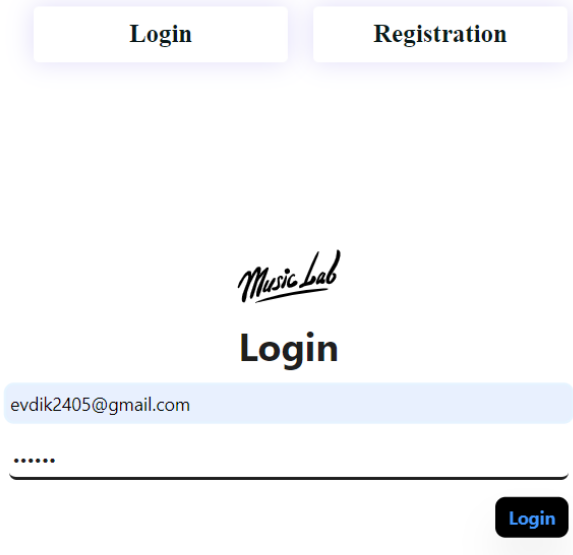

Рисунок 3.6 – Авторизація на сайті

Коли користувач авторизувався маршутизатор повинен перекинути користувача в домашню директорію. Домашня директорія показана на рисунку 3.4.

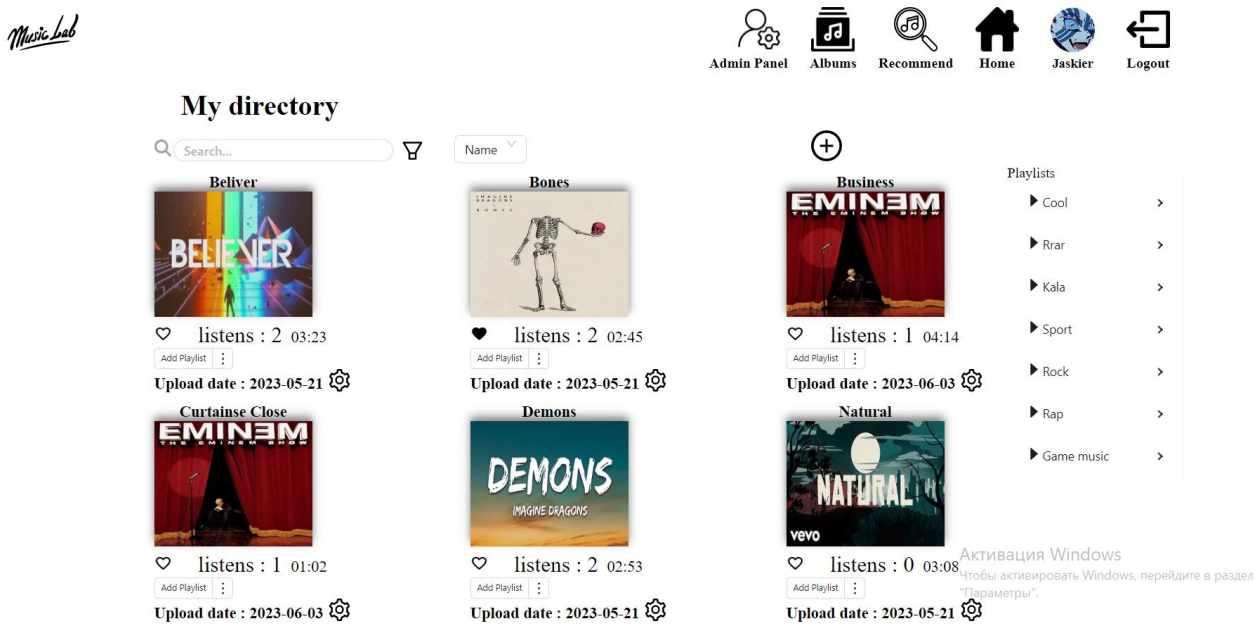

Рисунок 3.7 – Директорія користувача

Для завантаження музики користувач повинен натиснути на плюсик, біля списку музики. Продемонстровано це на рисунку 3.5.

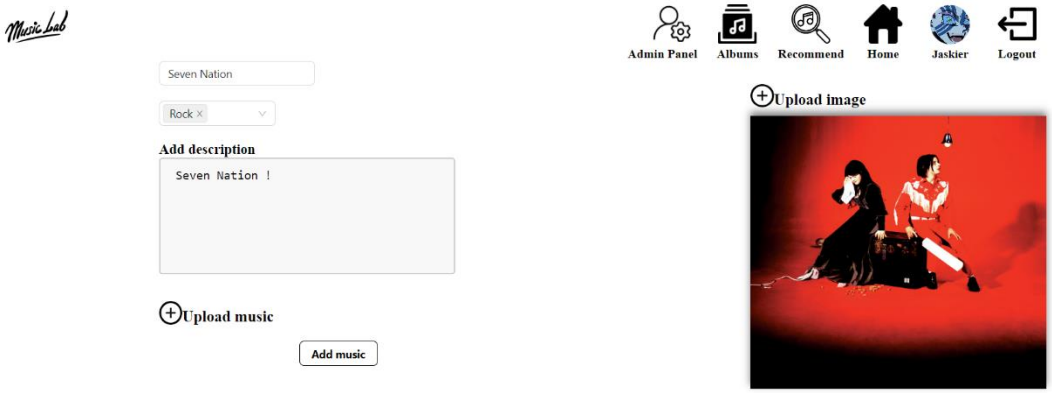

Рисунок 3.8 – Завантаження музики на сайт

Для пошуку музики треба вести назву музики у поле input. Це продемонстровано на рисунку 3.9.

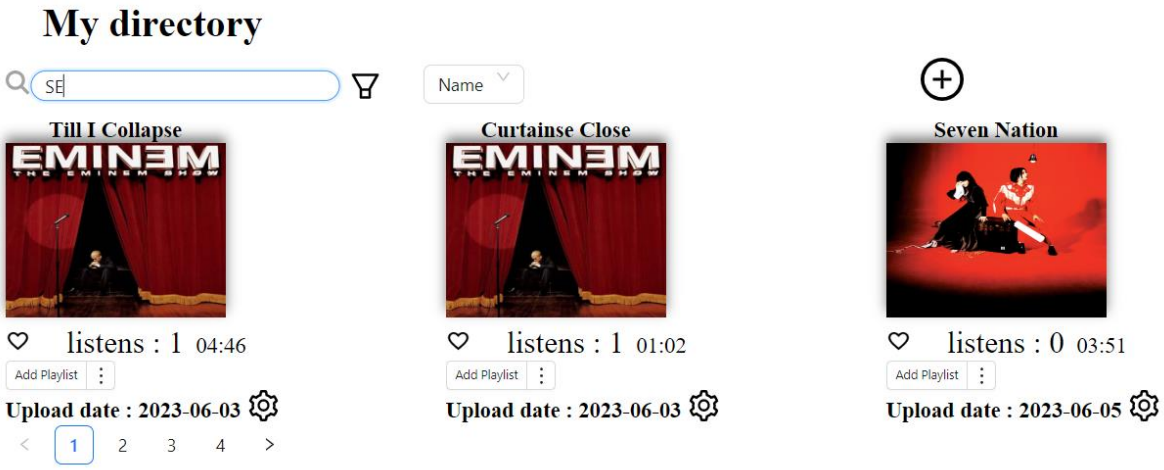

Рисунок 3.9 – Пошук музики на сайті

Для фільтрації музики треба вибрати фільтр і жанр. Показаний приклад на рисунку 3.7.

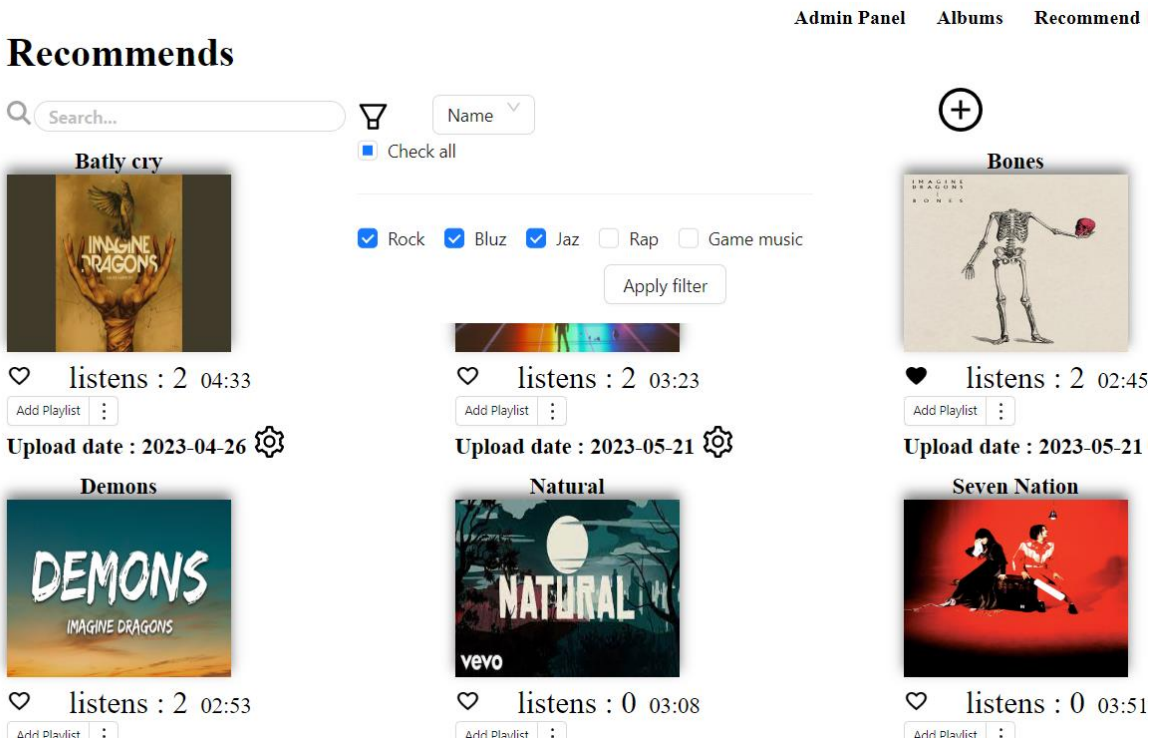

Рисунок 3.10 - Фільтрація музики

Для того щоб прийняти параметри фільтрації, треба натиснути кнопку «прийняти фільтрацію».

Висновки до розділу 3

Під час виконання розділу третього було розглянуто написання програмного забезпечення як на серверній частині так і клієнтській. Також були розглянуті технології розробки та мову програмування застосунку. Були розглянуті певні функції, сервіси, контролери, компоненти, проміжні функції, конфігурації, пакети які використовувалися для розрахування часу тривалості аудіо файлу та створення директорій для розміщення цих аудіо файлів. Також було показано методи HTTP запитів на сервер та компоненти які відповідальні за маршутизацію в програмному забезпеченні. Була розглянута розробка системи рекомендацій і логіка по якій працює ця система.

Програмне забезпечення було продемонстроване, усі функції які відповідають за виконання ти чи інших дій, були успішно протестовані після розробки програми. Було продемонстрована реєстрація та авторизація на сайті, також продемонстроване виконання пошук, фільтрація, завантаження, домашня сторінка користувача.

ВИСНОВКИ

Під час виконання робити з кваліфікаційної роботи бакалавр була досягнута мета роботи по популяризації композиторів, що мають малу аудиторію за рахунок розробки системи рекомендацій музики в сервісі потокової передачі музики.

Для досягнення визначної мети вирішено такі завдання:

– розглянуті застосунки аналоги та основні функції застосунку що розробляється;

– продемонстровано архітектуру та патерн проектування, в якості якого виступає патерн «шари абстракції»;

– досліджені бібліотеки та інстурменти які використовуються при розробці ПЗ;

– спроектований план вебзастосунку.

Були виконані такі проектні рішення як:

– створення приємного інтерфейсу;

– придбання підписки для прослуховування трендової музики;

– фільтрування музики за жанром, тривалістю, композитором та датою;

– можливість безкоштовно завантажувати музику з сайту та на сайт;

– змога користувачів слідкувати за користувачами.

Результатом конструювання нового музичного вебсервісу є користування слухачами, для прослуховування музики та композиторами для популяризації музики. Також одна з основних цілей є вихід на ринок стримінгових сервісів.

Також було побудовано діаграми мови UML для моделювання бізнес-процесів, системного проектування та відображення структури програмного забезпечення. Були сконструйовані такі діаграми: діаграма варіантів використання, діаграма взаємодій, діаграма станів та переходів, діаграма діяльності, діаграма пакетів та діаграма розгортання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Стримінгові сервіси. URL: http://ipkey.com.ua/faq/932-streaming-service.html (Last accessed: 15.05.2023)

2. Дистрибуція музики. URL: https://arefyevstudio.com/uk/2020/04/12/shho-take-distribuciya-muziki-i-yak-ce-pracyuye/ (Last accessed: 05.05.2023)

3. Spotify architecture. URL: https://www.infoq.com/news/2022/07/spotify-system-model-c4/ (Last accessed: 03.04.2023)

4. SoundCloud architecture. URL: https://www.techaheadcorp.com/blog/decoding-software-architecture-of-spotify-how-microservices-empowers-spotify/ (Last accessed: 02.04.2023)

5. Deezer programming language. URL: https://www.quora.com/Whats-the-language-Deezer-is-programmed-with (Last accessed: 03.04.2023)

6. Трирівнева архітектура застосунку на Node.js. URL: https://blog.devgenius.io/the-three-layer-architecture-for-node-js-applications-ce32a3a30fa6 (Last accessed: 02.06.2023)

7. Patrick Grässle, Henriette Baumann, Philippe Baumann UML 2.0 in Action : Birmingham , 2005 . 225 p.

8. Alex Banks, Eva Porcello Learning React: Functional Web Development with React and Redux First Edition : Sebastopol, 2017. 329 p.

9. Alex Banks, Eva Porcello Learning React: Functional Web Development with React and Redux Second Edition : Sebastopol, 2020 . 289 p.

10. Mike Cantelon, Marc Harter, T.J. Holowaychuk, Nathan Rajlich: Node.js in action : Shelter Island, 2014 . 343 p.

11. Moongoose. URL: https://mongoosejs.com/ (Last accessed: 02.06.2023)

ДОДАТОК А

Діаграма класів

Основна ідея діаграми класів це взаємодія з моделями через сервіси , де реалізовується бізнес логіка застосунку. Тобто через сервіси, буде взаємодія з базою даних та з моделями: оновлення, зберігання, видалення, читання бази даних. Діаграма зображена на рисунку А1.

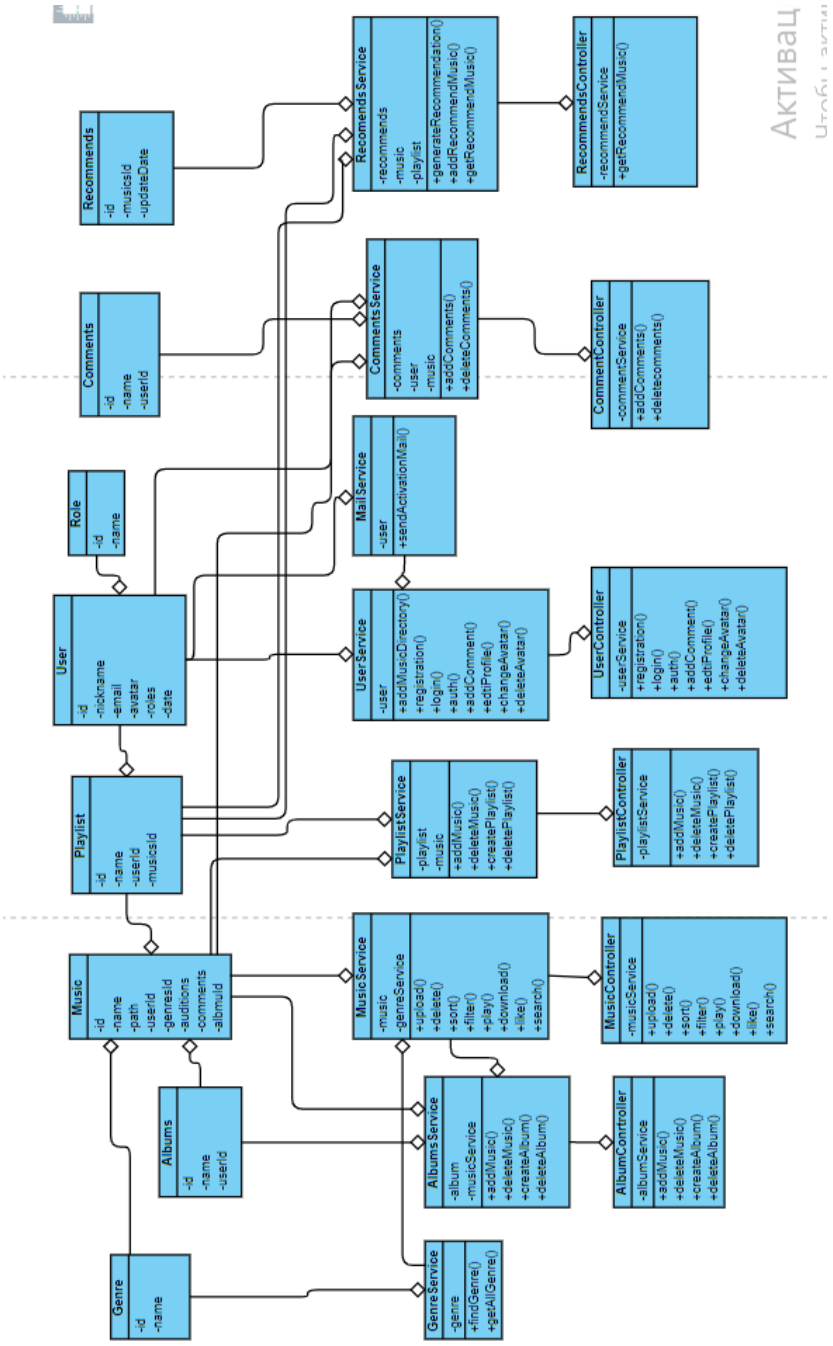

Рисунок А.1 – Діаграма класів

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

ВЕБЗАСТОСУНОК ПОТОКОВОЇ ПЕРЕДАЧІ МУЗИКИ

СПЕЦІАЛЬНА ЧАСТИНА З ОХОРОНИ ПРАЦІ

ОХОРОНА ПРАЦІ НА РОБОЧИХ МІСЦЯХ ФАХІВЦІВ З ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Спеціальність 121 «Інженерія програмного забезпечення»

121 – КРБ.1 – 409.21910909

Студент _____О. І. Євдокімов
підпис
«__» _____ 2023 р.

Консультант канд. техн. наук, доцент _____А. О. Алексєєва
підпис
«__» _____ 2023 р.

Миколаїв – 2023

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ОП – охорона праці

ПК – персональний комп'ютер

ВЧ – високочастотне випромінювання

УВЧ – ультрависокочастотне випромінювання

НВЧ – надвисокочастотне випромінювання

IT – information technology

ВСТУП

Для забезпечення комфортних умов праці, роботодавець повинен врахувати стандарти безпеки та умови праці. Робоче місце програміста є основною ланкою трудового процесу, де зосереджено матеріально-технічні елементи виробництва та здійснюється його трудова діяльність. Для того, щоб людина працювали ефективно, треба мінімізувати вплив негативних зовнішніх факторів.

Основною метою охорони праці є зменшити кількість небезпечних чинників для працівника за допомогою спеціально визначених норм. Основний процес в цій галузі є нормалізація робочого часу та робочого середовища працівників.

На сьогодні кожен інженер повинен мати певний обсяг знань у галузі ОП. Кожний елемент цієї галузі вирішує певні проблеми відносин працівника та роботодавця та безпеки усіх членів компанії на робочому процесі.

Працюючи з комп'ютером людина піддається впливу низки небезпечних і шкідливих виробничих чинників: електромагнітних полів (діапазон радіочастот: ВЧ, УВЧ і НВЧ), інфрачервоного та іонізуючого випромінювань, шуму та вібрації, статичної електрики та інші.

Основні питання які розглядаються в галузі охорони праці це – оцінка умов робочого місці програміста; заходи забезпечення безпечних умов праці; пожежна безпека.

Також дуже важливим чинником є параметри мікроклімату. Параметри мікроклімату можуть змінюватися у межах, тоді як необхідною умовою життєдіяльності людини є підтримання сталості температури тіла завдяки терморегуляції. Принцип нормування мікроклімату – створення оптимальних умов теплообміну тіла людини з довкіллям [1].

Так, у першому розділі проаналізовано умови праці фахівців з ІТ – технологій у робочому приміщені.

У другому розділі розглянуті основні правила пожежної безпеки.

1 ОЦІНКА УМОВ ПРАЦІ ФАХІВЦІВ З ІТ-ТЕХНОЛОГІЙ У РОБОЧОМУ ПРИМІЩЕННІ

1.1 Площа приміщення

Площа приміщення повинна бути не менше 6,0 м² на 1 робоче місце; робочі місця повинні бути розташовані на відстані не менше ніж 1 м від стіни з вікном, і 1,4 м від звичайної стіни; відстань між бічними поверхнями комп‘ютерів має бути не меншою за 1,2 м; відстань між тильною поверхнею одного комп‘ютера та екраном іншого не повинна бути меншою 2,5м [2].

1.2 Облаштування робочого місця працівника

Запорука ефективної роботи програміста – це не тільки глибокі знання своєї галузі, але ще зручні меблі та комфортна обстановка.

Зручний стіл повинен відповідати наступним параметрам [3]:

– широка стільниця, яка має прогинатися під монітор;

– комфортна висота, яка залежить від висоти стільця та вашого зростання;

– ширина від 75 см, щоб монітор не стояв близько;

– підставка під руку, яка працює з мишею.

Багато програмістів також зручні столи, які регулюються по висоті: за ними можна працювати як сидячи, так і стоячи. Вони мають плюс – можна розім'ятися не відходячи від робочого місця. І мінус те що така фішка швидко набридає, і більшість часу за столом доводиться працювати сидячи.

Вибираючи стілець, багато хто забуває, що його основні функції – підтримувати спину і близько присуватися до столу. Тому правильно підібраний стілець повинен [3]:

– підходити під поставу, е всіх вона різна, тому важливо підібрати стілець саме під кожного індивідуально: щоб він підтримував спину та поперек, не був занадто жорстким і мав нахил від 110 градусів;

– провітрювати спину, важливо, щоб під час роботи спина не потіла від неякісних матеріалів або відсутності сіток, що пропускають повітря;

– мати зручні підлокітники, вони повинні підтримувати лікті у розслабленому стані, а висота підлокітників має дозволяти вам заїжджати під стільницю;

– не стискати стегна, хороший стілець повинен бути ширшим за ваші стегна на 15-20 см;

– мати важіль регулювання по висоті;

– мати підтримку для голови – на ваше бажання.

Незалежно від того, на чому ви працюватимете, на ноутбуці або ПК, потрібен такий монітор, щоб людина могла бачити код та інші відкриті файли. Багато програмістів навіть працюють за двома моніторами, щоб бачити кожну важливу деталь.

У будь-якому випадку монітор повинен розташовуватися:

– на комфортній відстані від очей головне, щоб голова не дивилася вниз при роботі, погляд прямо чи трохи вище – допустимо;

– там, куди не потрапляє сонячне чи штучне світло;

– яскравість і роздільна здатність – за вашим бажанням;

– головне завдання хорошого монітора – не повинен напружувати очі.

Лампа повинна вирівнювати контраст картинки на моніторі та навколишнього простору. Правильно підібрана лампа знизить навантаження на очі і не дасть перенапружитися м'язам ока.

Лампа повинна:

– регулюватись за висотою;

– мати абажур, щоб не світло не зліпило очі;

– розташовуватися ліворуч на робочому столі.

1.3 Способи освітлення робочого місця

Раціональне освітлення робочого місця є одним із найважливіших факторів, що впливають на ефективність трудової діяльності людини, що запобігають травматизму та професійним захворюванням. Правильно організоване висвітлення створює сприятливі умови праці, підвищує працездатність та продуктивність праці. Висвітлення на робочому місці програміста має бути таким, щоб працівник міг без напруження зору виконувати свою роботу. Стомлюваність органів зору залежить від низки причин:

– недостатність освітленості;

– надмірна освітленість;

– неправильний напрямок світла.

Недостатність освітлення призводить до напруги зору, послаблює увагу, призводить до настання передчасної втоми. Надмірно яскраве освітлення викликає засліплення, роздратування та різь в очах. Неправильний напрямок світла робочому місці може створювати різкі тіні, відблиски, дезорієнтувати працюючого. Всі ці причини можуть призвести до нещасного випадку або профзахворювань, тому важливий правильний розрахунок освітленості.

Розрахунок освітленості робочого місця зводиться до вибору системи освітлення, визначення необхідної кількості світильників, їх типу та розміщення. Процес роботи програміста в умовах, коли природне освітлення недостатньо чи відсутня. Виходячи з цього, розрахуємо параметри штучного висвітлення [4].

1.4 Методи захисту робочого місця від шуму

Інтенсивний виробничий шум може стати причиною таких професійних захворювань, як туговухість або глухота. Крім того, у працівників, які щодня перебувають під його впливом:

– знижується продуктивність праці;

– ослаблюється увага та уповільнюється реакція, спостерігається запаморочення, дратівливість, знижується працездатність, гострота зору;

– зростає кров’яний тиск, змінюється ритм дихання та серцевої діяльності, порушується працездатність клітин кори головного мозку тощо.

Звичним для людини є шумовий фон з рівнем звукового тиску в частотах 15-35 дБ.

При збільшенні рівня звукового тиску до 40-70 дБ спостерігається деяке зниження продуктивності праці та погіршення самопочуття (голосна музика, шум технічного устаткування та інше).

Рівень звукового тиску в межах 75-120 дБ спричиняє враження органів слуху і серцево-судинної системи.

Постійний шум з рівнем звукового тиску понад 120 дБ може призвести до акустичної травми (значне зниження слуху) [5].

На робочому місці програміста джерелами шуму, як правило, є технічні засоби, як-от комп'ютер, принтер, вентиляційне обладнання, а також зовнішній шум. Вони видають досить незначний шум, тому в приміщенні достатньо використовувати звукопоглинання. Зменшення шуму, що проникає в приміщення ззовні, досягається ущільненням по периметру притворів вікон та дверей.

Під звукопоглинанням розуміють властивість акустично оброблених поверхонь зменшувати інтенсивність відбитих ними хвиль за рахунок перетворення звукової енергії на теплову. Звукопоглинання є досить ефективним заходом для зменшення шуму. Найбільш вираженими звукопоглинаючими властивостями володіють волокнисто-пористі матеріали: фібролітові плити, скловолокно, мінеральна вата, поліуретановий поропласт, пористий полівінілхлорид та інших. До звукопоглинаючих матеріалів відносяться лише ті, коефіцієнт звукопоглинання яких не нижче 0.2.

Звукопоглинаючі облицювання із зазначених матеріалів (наприклад, мати із супертонкого скловолокна з оболонкою зі склотканини потрібно розмістити на стелі та верхніх частинах стін). Максимального звукопоглинання буде досягнуто при облицюванні не менше 60% загальної площі поверхонь приміщення, що огороджують [6].

1.5 Облаштування вентиляції на робочому місці

Системи опалення та кондиціонування слід встановлювати так, щоб ні тепле, ні холодне повітря не направлялося на людей. На виробництві рекомендується створювати динамічний клімат із певними перепадами показників. Температура повітря біля поверхні підлоги і на рівні голови не повинна відрізнятися більш ніж на 5 градусів.

У виробничих приміщеннях крім природної вентиляції передбачають припливно-витяжну вентиляцію.

У приміщеннях на робочих місцях повинні забезпечуватись оптимальні значення параметрів мікроклімату: температури, відносної вологості та рухливості повітря відповідно до ДСН 3.3.6.042-99 [7].

Таблиця 1 – Оптимальні характеристики мікроклімату

Період року	Категорія робіт	Температура повітря, С	Відносна вологість, %	Швидкість руху повітря, м/с
		оптимальна	оптимальна	оптимальна
Холодний період року	легка – 1 а	22 – 24	40 – 60	0,1
	легка –1 б	21 – 23	40 – 60	0,1
Теплий період року	легка – 1 а	23 – 25	40 – 60	0,1
	легка – 1 б	22 – 24	40 – 60	0,2

Ці норми встановлюються залежно від пори року, характеру трудового процесу та характеру виробничого приміщення.

2 ПОРЯДОК ДІЙ ПЕРСОНАЛУ У НАДЗВИЧАЙНІЙ СИТУАЦІЇ, ЩО ПОВ'ЯЗАНА З ПОЖЕЖНОЮ НЕБЕЗПЕКОЮ

2.1 Дії співробітників під час пожежі

При появі запаху гару, задимлення чи вогню потрібно відразу ж зателефонувати рятувальникам за номером 101 (залежить від оператора). Повідомите диспетчеру такі дані:

– адреса офісної будівлі;

– місце виявлення пожежі;

– рівень (масштаби) загрози;

– чи є постраждалі;

– свої контактні дані.

Після виклику пожежної служби повідомити про пожежу охорону, керівника та інших співробітників. Це можна зробити телефоном, за допомогою ручних пожежних сповіщувачів або примусового увімкнення сигналізації.

Краще не намагатись боротися з вогнем самостійно. Це може призвести до опіків або електротравм. Керівництво компанії насамперед має подбати про злагоджену евакуацію людей [8].

2.2 Дії керівника компанії під час пожежі

Керівник організовує евакуацію людей із офісу. Також має зробити таке [8]:

– перевірити, чи поінформовані пожежні служби про цей інцидент;

– контролювати оповіщення всіх працівників;

– відкрити евакуаційні шляхи;

– зупинити виробництво та вивести персонал та відвідувачів: саме вони, а не матеріальне майно евакуюються насамперед;

– увімкнути автоматичні протипожежні системи захисту, якщо є;

– відключити електрику, по можливості закрити вентиляцію та пожежні двері.

При цьому головне завдання керівника – не допустити поширення паніки серед працівників. Інструкція щодо дій під час пожежі в офісі також зобов'язує начальника компанії призначити співробітників, які зустрічатимуть рятувальників, зможуть вказати місце знаходження гідрантів, описати планування внутрішніх приміщень, при необхідності показати, як краще під'їхати до будівлі тощо.

2.3 Засоби пожежогасіння

Первинні засоби пожежогасіння – це технічні засоби, речовини або їх комплекс, придатні для використання людиною для локалізації та (або) ліквідації пожежі на її початковій стадії [9].

Відповідно до Правил пожежної безпеки України (п.3.6 розділ V), затверджених наказом МВС від 30.12.2014р. № 1417 [10], первинні засоби пожежогасіння мають бути фактично на будь-якій території та в будь-якому поміщенні.

До первинних засобів пожежогасіння належать:

– вогнегасники;

– ящики із піском;

– бочки із водою;

– покривала з негорючого теплоізоляційного матеріалу;

– пожежні відра, совкові лопати, пожежний інструмент – кирки, сокири, багри, ломи тощо.

Кількість та тип вогнегасників залежить від площі приміщень, їх категорії щодо пожежної небезпеки, а також фізико-хімічних та пожежонебезпечних властивостей речовин, що в них зберігаються або використовуються.

ВИСНОВКИ

В спеціальній частині кваліфікаційної роботи бакалавра розглянуто та проаналізовано фактори, які становлять небезпеку для зручного робочого процесу або життя працівника.

Під час виконання роботи було засвоєно знання правил облаштування робочого місця програміста, мікроклімат в офісі або комп'ютерному класі були розглянуті та продемонстровані оптимальні характеристики чинників мікроклімату. У результаті всі фактори задовольняють санітарні вимоги. Також, було розглянуто методи боротьби з шумом і джерела що створюють цей шум. Хочеться зосередити увагу на освітлені приміщення, і які саме пристрої треба використати для чудового освітлення.

Описано можливі фактори та ризики виникнення пожежі. Для запобігання високих ризиків загорання, треба дотримуватись інструкцій, які описані. Також описаний план евакуації працівників та роботодавців на підприємстві під час пожежі. Були продемонстровані засоби гасіння пожежі. За підведеними раніше підсумками, основне завдання щодо нормалізації робочого процесу працівника на підприємстві була виконаною.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Сприятливі умови праці програміста. URL: https://studbooks.net/2029280/informatika/harakteristiki_usloviy_truda_programmista (дата звернення: 18.05.2023)

2. Площа для зручності програмування. URL: https://te.dsp.gov.ua/robota-v-ofisi-osnovni-sanitarno-gigiyenichni-vymogy/ (дата звернення: 18.05.2023)

3. Облаштування робочого місця програміста. URL: https://habr.com/ru/companies/maxilect/articles/466333/ (дата звернення: 18.05.2023)

4. Забезпечення освітлення робочого місця. URL: https://vuzlit.com/162510/osveschennost_rabochego_mesta (дата звернення: 18.05.2023)

5. Забезпечення робочого місця від шуму. URL: https://pro-op.com.ua/article/1071-zasobi-zahistu-vd-shumu-vbrats (дата звернення: 18.05.2023)

6. Методи захисту шуму. URL: https://vuzlit.com/162511/metody_zaschity_shuma (дата звернення: 18.05.2023)

7. Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99. URL: https://zakon.rada.gov.ua/rada/show/va042282-99# (дата звернення: 18.05.2023)

8. Евакуація при пожежі. URL: https://oib.com.ua/ua/algorytm-dij-pid-chas-pozhezhi-v-ofisnij-budivli/ (дата звернення: 18.05.2023)

9. Засоби пожежогасіння. URL: https://ohrana-truda.kiev.ua/ua/vognegasniki-kran-komplekti-shhiti-shho-vidnosyat-do-pervinnih-zasobiv-pozhezhogasinnya-ta-yak-yih-obrat/ (дата звернення: 18.05.2023)

10. Наказ №1417 Про затвердження Правил пожежної безпеки України. URL: https://zakon.rada.gov.ua/laws/show/z0252-15 (дата звернення: 18.05.2023)