Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Колесника Олександра Олександровича*
(ПІБ)

академічної групи *121-19-2*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: *Розробка додатку аудіоплеєра за допомогою мови програмування JavaScript*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*проф. Мещеряков Л.І.*			
розділів:				
спеціальний	*проф. Мещеряков Л.І.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*ст.викл. Мартиненко А.А.*			

Дніпро
2023

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-19-2* (група) *Колесника О.О.* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка додатку аудіоплеєра за допомогою мови програмування JavaScript*

затверджена наказом ректора НТУ «ДП» від *16.05.2023* № *350-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2023 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2023 р.***

Завдання видав (підпис) *проф. Мещеряков Л.І.* (посада, прізвище, ініціали)

Завдання прийняв до виконання (підпис) *Колесник О.О.* (прізвище, ініціали)

Дата видачі завдання: *14.01.2023 р.*

Термін подання кваліфікаційної роботи до ЕК: *12.06.2023 р.*

РЕФЕРАТ

Пояснювальна записка: 62 с., 2 рис., 11 джерел.

Об'єкт розробки: Аудіоплеєр на мові програмування JavaScript з використанням фреймворку React та платформи Node.js.

Мета кваліфікаційної роботи: написання додатку, який надає користувачу доступний аудіо-плєер з можливістю програвання усіх найпопулярніших аудіоформатів.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточняється постановка завдання.

У першому розділі проаналізовано предметну галузь, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проаналізовані наявні рішення, обрано платформи для розробки, виконано проектування і розробка програми, описана робота програми, алгоритм і структура її функціювання, а також виклик та завантаження програми, визначено вхідні і вихідні дані, охарактеризовано склад параметрів технічних засобів.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення полягає у створенні програмного додатка, що дозволяє зручно та комфортно слухати аудіофайли різних форматів у одному додатку. Актуальність даного додатку стоїть у тому, щоб створити мультиформатний аудіоплеєр з компактним дизайном, який спростить життя звичайному слухачу.

Список ключових слів: АУДІОПЛЕЄР, МУЛЬТИФОРМАТНИЙ, СПИСОК, КОМП’ЮТЕР, ІНТЕРФЕЙС, ДОДАТОК, VISUAL STUDIO.

ABSTRACT

Explanatory note: 62 p., 2 figs., 3 apps., 11 sources.

Object of development: Audio player coded in JavaScript programming language using React framework and Node.js platform.

The purpose of the qualification work: coding an application that provides the user with an audio player with the ability to play all the most popular audio formats and combines a unique and practical design.

The introduction examines the current problem statement, specifies the purpose of the qualification work and the area of its application, justifies the relevance of the topic and specifies the problem statement.

In the first section carries out the analysis of the subject area, determines the relevance of the task and the dedication of the development, creates task statement, the software and hardware requirements of the product, specifies technologies and tools for development.

In the second section analyzes the existing solutions, chose the platform for development, creates design and finish development of the product, describes the algorithm, structure and architecture solutions in the system, defines the input and output data, describes characteristics of the technical resources used, describes how to run a program, features of user interaction, differences between serve and client parts of product.

The economic section defines the complexity of the information system, calculates the cost of work on creating the application and defines the time for its creation. The practical significance is creation a product which provides an opportunity to improve knowledge and increase knowledge. The relevance of this application is to create a multi-format audio player with a compact design that will simplify the life of the average listener.

Keywords: AUDIO PLAYER, MULTI-FORMAT, PLAYLIST, COMPUTER, INFORMATION SYSTEM, INTERFACE, APPLICATION, VISUAL STUDIO.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

HTML (HyperText Markup Language) – основна мова розмітки веб-сторінок.

JS (JavaScript) – високорівнева мова програмування, яка переважно використовується для веб-розробки.

React – відкрита бібліотека JavaScript для розробки інтерфейсів користувача.

JSX (JavaScript XML) – розширений синтаксис, що використовується в React для написання JavaScript-коду, який нагадує синтаксис HTML/XML.

CSS – мова стилів, яка використовується для визначення вигляду і форматування веб-сторінок.

ВСТУП

Ще з початку часів музика, у тому чи іншому вигляді, акомпонувала кожному кроку розвитку людства. З плином розвитку технологій, люди винаходили все більше способів записувати та зберігати музику. Спочатку це було на папері, потім на воску, ще пізніше - на платівках. Прихід цифрових технологій також приніс із собою новий та зручний спосіб зберігання музики - у файлі. Зберігання її такою, якою вона була зіграна. Більше не потрібно громіздких програвачів та довгих проводів, тим паче не потрібно збирати оркестр кожного разу, коли тобі хочеться щось послухати. Сьогодні, комп'ютери та смартфони дозволяють бути нерозлучними з улюбленою музикою. Але, як і для усього, для програвання аудіофайлів потрібні спеціальні додатки - плеєри. Саме тому дана кваліфікаційна робота націлена на проектування та реалізацію додатку-плеєра, який зможе поєднати в собі такі важливі аспекти як функціональність, зручність та не займати багато місця у пам'яті сучасних гаджетів.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної області

Музичний веб-плеєр - це електронний пристрій або програмне забезпечення, яке відтворює аудіофайли на веб-сайті та може бути доступний з будь-якого пристрою, який має доступ до Інтернету.

Основними перевагами музичного веб-плеєра є зручність та доступність. Користувач може відтворювати музику безпосередньо на веб-сторінці, не потребуючи встановлення додаткового програмного забезпечення на своєму пристрої. Крім того, музичний веб-плеєр може мати додаткові функції, такі як можливість створення власних плейлистів, налаштування звуку та рівня гучності, а також функції, які дозволяють ділитися музикою з друзями та родичами.

Музичні веб-плеєри можуть пропонувати різноманітний музичний контент, включаючи популярні хіти, новинки та старі класики в різних жанрах. Крім того, на деяких веб-плеєрах можна знайти радіостанції та плейлисти, створені користувачами.

Підприємці, які бажають створити свій власний музичний веб-плеєр, можуть скористатися різноманітними готовими платформами, які дозволяють створити сайт без додаткових знань програмування. Крім того, вони можуть скористатися послугами спеціалізованих компаній, які займаються розробкою та просуванням музичних веб-плеєрів.

Музичні веб-плеєри стали невід'ємною частиною сучасної музичної індустрії, і вони продовжують зростати в популярності. Якщо вони

використовуються ефективно, то можуть стати вигідним та зручним варіантом для меломанів та підприємців.

1.2. Призначення розробки та область застосування

Веб-додаток який розробляється, призначений для використання клієнтами по всій території України, які зацікавлені у відтворенні музики через Інтернет. Основною областю застосування є прослуховування музики безпосередньо на веб-сторінці без необхідності встановлення додаткового програмного забезпечення на пристрої. Крім того, користувачі можуть створювати власні плейлисти, налаштовувати звук та рівень гучності, а також ділитися музикою з друзями та родичами. Окрім цього, музичний веб-плеєр може пропонувати різноманітний музичний контент в різних жанрах, включаючи популярні хіти, новинки та старі класики, а також радіостанції та плейлисти, створені користувачами. Додатково можуть бути запропоновані послуги консультації користувачів щодо вибору музики та різноманітних акцій та знижок.

1.3. Підстава для розробки

Підставами для розробки та виконання кваліфікаційної роботи є:

- освітня програма 121 «Інженерія програмного забезпечення»;
- навчальний план та графік навчального процесу;
- наказ ректора Національного технічного університету «Дніпровська політехніка» № 350-с від 16.05.2023 р;
- завдання на кваліфікаційну роботу на тему «Розробка додатку аудіоплеєра за допомогою мови програмування JavaScript».

1.4. Постановка завдання

Розробити веб-додаток музичного плеєра на React, TypeScript, бази даних MongoDB та сервера на Node Js має на меті задоволення потреб користувачів у прослуховуванні музики через Інтернет. Основними завданнями розробки є:

- розробка зручного та інтуїтивно зрозумілого інтерфейсу для користувачів, який дозволяє швидко знайти потрібну музику та відтворити її;
- розробка функціоналу для управління музичним контентом, включаючи можливість додавання нових пісень, редагування інформації про них, видалення старих;
- розробка системи оплати, яка дозволяє користувачам зручно та безпечно оплачувати доступ до преміум-контенту;
- розробка функціоналу для управління особистим кабінетом користувача для додавання інформації, яка потрібна для оплати та вибору музичного контенту на їх смак.

1.4.1. Функціональні особливості, актуальність сайту та можливості

Основне функції та можливості додатку:

- програвання файлів формату аудіо;
- надання функціонального блоку навігації;
- сумісність з ОС Windows версій 7, 8, 10 та 11;
- підтримка найпоширеніших аудіоформатів.

1.4.2. Опис інтерфейсу користувача

Інтерфейс складається з кількох основних компонентів:

- Компонент для програвання музики, який включає в себе кнопки керування відтворенням, поле для відображення назви і тривалості поточної композиції, а також смугу прокрутки для перемотування треку;

- Компонент пошуку, який включає в себе поле для введення назви композиції або виконавця, а також кнопку для запуску пошуку;
- Компонент для складання плейлистів, який містить список плейлистів і кнопки для їх створення, редагування та видалення;
- Компонент, який використовує базу даних для зберігання інформації про композиції та плейлисти.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Музичний плеєр має бути зручним та доступним для користувачів. Основна функція музичного плеєра - відтворення аудіофайлів, але веб-музичний плеєр може мати багато додаткових функцій, які зроблять його більш зручним та привабливим для користувачів. Ці функції можуть включати в себе можливість створення власних плейлистів, регулювання звуку та рівня гучності, а також можливість ділитися музикою з друзями та родичами.

Музичний плеєр може пропонувати різноманітний музичний контент, включаючи популярні хіти, новинки та старі класики в різних жанрах. Крім того, на деяких веб-плеєрах можна знайти радіостанції та плейлисти, створені користувачами.

При розробці музичного веб-плеєра, основною метою є забезпечення користувачам максимальної зручності та доступності. Музичний плеєр має бути легко зрозумілим та інтуїтивно зрозумілим для користувачів, щоб вони могли без зайвих зусиль знаходити необхідну музику та користуватися всіма його функціями. Особлива увага повинна бути приділена дизайну та інтерфейсу музичного плеєра, щоб зробити його привабливим та зручним для користувачів.

Музичні веб-плеєри стали невід'ємною частиною сучасної музичної індустрії, і вони продовжують зростати в популярності. Якщо вони

використовуються ефективно, то можуть стати вигідним та зручним варіантом для меломанів та підприємців.

1.5.2. Вимоги до інформаційної безпеки

Для використання музичного плеєра користувач повинен зареєструватись та авторизуватися, використовуючи свій логін та зашифрований пароль, який зберігається у базі даних для захисту від шахрайства. Крім того, на сайті музичного плеєра існують дві ролі - користувач та адміністратор, кожен з яких має доступ до своєї відповідної панелі. Адміністратору відкривається можливість додавати, редагувати та видаляти музичні композиції.

Музичний плеєр є електронним пристроєм або програмним забезпеченням, яке відтворює аудіофайли на веб-сайті та може бути доступний з будь-якого пристрою, який має доступ до Інтернету. Основними перевагами музичного плеєра є зручність та доступність. Користувач може відтворювати музику безпосередньо на веб-сторінці, не потребуючи встановлення додаткового програмного забезпечення на своєму пристрої. Крім того, музичний плеєр може мати додаткові функції, такі як можливість створення власних плейлистів, налаштування звуку та рівня гучності, а також функції, які дозволяють ділитися музикою з друзями та родичами.

Музичний веб-плеєр може пропонувати різноманітний музичний контент в різних жанрах, включаючи популярні хіти, новинки та старі класики, а також радіостанції та плейлисти, створені користувачами. Підприємці, які бажають створити свій власний музичний плеєр, можуть скористатися різноманітними готовими платформами, які дозволяють створити сайт без додаткових знань програмування. Крім того, вони можуть скористатися послугами спеціалізованих компаній, які займаються розробкою та просуванням музичних плеєрів.

Музичні веб-плеєри стали невід'ємною частиною сучасної музичної індустрії, і вони продовжують зростати в популярності. Якщо вони використовуються ефективно, то можуть стати вигідним та зручним варіантом.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для зручного використання веб-додатка технічний засіб користувача повинен відповідати таким технічним вимогам:

- мати операційну систему Windows 7-11, Linux або MacOS;
- мати не менше 2 ГБ оперативної пам'яті;
- мати не менше 5 ГБ вільного місця на жорсткому диску з системою.

1.5.4. Вимоги до інформаційної та програмної сумісності

Додаток був розроблений за допомогою мови програмування JavaScript. Середовищем розробки є інтегроване середовище розробки Visual Studio Code. Користувач повинен мати достатньо нову версію браузера, щоб забезпечити коректну роботу додатку. Додаток підтримує наступні версії браузерів:

- Windows: 7 версія та вище;
- Chrome: 113 версія та вище;
- Opera: 98 версія та вище;
- Edge: 112 версія та вище;
- Safari: 14.1 версія та вище;
- IE: 11 версія та вище;
- Android: 9 версія та вище;
- Firefox: 106 версія та вище;
- Opera Mobile: 54.3 версія та вище.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Результатом даної кваліфікаційної роботи має стати додаток-аудіоплеєр, написаний на мові програмування JavaScript з застосуванням додаткового фреймворку, бібліотек, баз даних та серверів. Кінцевий додаток націлений на користувачів, які бажають прослухати аудіофайли. Основне призначення додатку:

- програвання файлів формату аудіо;
- надання функціонального блоку навігації;
- сумісність з ОС Windows версій 7, 8, 10 та 11;
- підтримка найпоширеніших аудіоформатів.

2.2. Опис застосованих математичних методів

Під час розробки цього аудіоплеєра не використовувалися математичні методи або алгоритми.

2.3. Опис використаної архітектури та шаблонів проектування

Архітектура додатка - це структура і компоненти додатка, а також зв'язки між ними. У цьому розділі буде визначено архітектуру веб-додатка музичного плеєра на React з використанням TypeScript і сервера на Node Js.

Основними компонентами програми є:

- **Компонент для програвання музики** - відповідає за програвання музичних композицій. Включає в себе кнопки керування відтворенням, поле для відображення назви і тривалості поточної композиції, а також смугу прокрутки для перемотування треку. На цьому компоненті відображатимуться іконки

додавання композиції до плейлиста та перегляду інформації про трек. Компонент пов'язаний з компонентом пошуку композицій і компонентом для складання плейлистів.

- **Компонент пошуку композицій** - відповідає за пошук музичних композицій. Включає в себе поле для введення назви композиції або виконавця, а також кнопку для запуску пошуку. На цьому компоненті відображатимуться результати пошуку у вигляді списку композицій. Компонент пов'язаний з компонентом для складання плейлистів і компонентом для збереження інформації про прослухані треки і плейлисти.

- **Компонент для складання плейлистів** - відповідає за складання плейлистів. Містить список плейлистів і кнопки для створення, редагування та видалення плейлистів. На цьому компоненті відображатимуться композиції, додані в плейлист, а також кнопки керування відтворенням плейлиста. Компонент пов'язаний з компонентом для збереження інформації про прослухані треки і плейлисти.

- **Компонент для збереження інформації про прослухані треки та плейлисти**- відповідає за збереження інформації про прослухані треки та плейлисти. Використовує базу даних для зберігання інформації про композиції та плейлисти. Компонент пов'язаний з компонентом для складання плейлистів і компонентом пошуку композицій.

Зв'язки між компонентами будуть визначені таким чином:

- Компонент для програвання музики буде пов'язаний з компонентом пошуку композицій і компонентом для складання плейлистів. Під час вибору композиції з результатів пошуку або з плейлиста компонент для відтворення музики відображатиме інформацію про обрану композицію та відтворюватиме її.

- Компонент пошуку композицій буде пов'язаний з компонентом для складання плейлистів і компонентом для збереження інформації про прослухані треки та плейлисти. При виборі композиції з результатів пошуку, компонент для складання плейлистів буде додавати обрану композицію в поточний плейлист.

Під час вибору композиції з плейлиста, компонент для програвання музики відображатиме інформацію про обрану композицію та відтворюватиме її.
- Компонент для складання плейлистів буде пов'язаний з компонентом для збереження інформації про прослухані треки та плейлисти. При виборі композиції з результатів пошуку, компонент для складання плейлистів буде додавати обрану композицію в поточний плейлист.
Таким чином, архітектура додатка матиме таку структуру:
- Компонент для програвання музики
  - Компонент пошуку композицій
  - Компонент для складання плейлистів
- Компонент пошуку композицій
  - Компонент для складання плейлистів
  - Компонент для збереження інформації про прослухані треки та плейлисти
- Компонент для складання плейлистів
  - Компонент для збереження інформації про прослухані треки та плейлисти

Функціонал додатка музичного плеєра на React із прикладом використання:

Програвання музики - користувач може вибрати композицію з пошуку і програвати її. Для цього на сторінці додатка буде розміщено компонент для програвання музики, який містить кнопки керування відтворенням (play, pause, stop), поле для відображення назви та тривалості поточної композиції, а також смугу прокрутки для перемотування треку.
Приклад використання: користувач вводить у поле пошуку назву композиції "Bohemian Rhapsody" і запускає пошук. За результатами пошуку на сторінці відображається список композицій, зокрема шукана композиція. Користувач обирає цю композицію і натискає на кнопку "play". Композиція починає грати, у полі для відображення назви та тривалості поточної композиції з'являється інформація про обрану композицію.

Пошук музики - користувач може шукати музичні композиції за назвою, виконавцем і жанром. Для цього на сторінці додатка буде розміщено компонент пошуку музики, який містить поле для введення назви композиції або виконавця, а також кнопку для запуску пошуку. Результати пошуку відображатимуться у вигляді списку композицій.

Приклад використання: користувач вводить у поле пошуку назву композиції "Bohemian Rhapsody" і запускає пошук. За результатами пошуку на сторінці відображається список композицій, зокрема шукана композиція. Користувач обирає цю композицію і додає її в плейлист.

Складання плейлистів - користувач може створювати плейлисти зі знайдених композицій. Для цього на сторінці додатка буде розміщено компонент для складання плейлистів, який містить у собі список плейлистів і кнопки для створення, редагування та видалення плейлистів. У плейлист можна додавати композиції, обрані з результатів пошуку.

Приклад використання: користувач створює новий плейлист і додає в нього кілька композицій, знайдених у результаті пошуку. Потім користувач вибирає цей плейлист і натискає на кнопку "play". Композиції починають грати по порядку, у полі для відображення назви та тривалості поточної композиції з'являється інформація про обрану композицію.

Збереження інформації про прослухані треки та плейлисти - додаток зберігає інформацію про прослухані треки та плейлисти в базі даних. Для цього використовується компонент для збереження інформації про прослухані треки та плейлисти. Цей компонент пов'язаний з компонентом для складання плейлистів і компонентом пошуку композицій.

Приклад використання: під час програвання композиції користувач може натиснути на кнопку "like" і додати композицію в обрані. Цю інформацію буде збережено в базі даних. Користувач також може зберегти поточний плейлист і

повернутися до нього пізніше. Збережені плейлисти будуть доступні в списку плейлистів під час наступного запуску програми.

Для реалізації проекту було обрано такі інструменти та технології:
- **React** - це JavaScript бібліотека для побудови користувацьких інтерфейсів. React дає змогу створювати компоненти, які можуть бути багаторазово використані в додатку, що робить код більш організованим і легко підтримуваним. Також, React використовує віртуальний DOM, що дає змогу оптимізувати продуктивність програми.
- **TypeScript** - це мова програмування, що являє собою надбудову над JavaScript. TypeScript додає статичну типізацію та інші нові можливості, що покращує якість коду і спрощує його підтримку.
- **NodeJs** - це середовище виконання JavaScript на сервері, яке дає змогу створювати серверні додатки на JavaScript. NodeJs має високу продуктивність і масштабованість, що робить його популярним вибором для створення серверної частини додатків.
- **Git** - це система контролю версій, яка дає змогу відстежувати зміни в коді та керувати ними. Git дає змогу зберегти історію змін, створювати гілки та зливати їх, працювати над проектом у команді та багато іншого.
- **VS Code** - це інтегроване середовище розробки, що дає змогу зручно розробляти додатки на різних мовах програмування, включно з JavaScript і TypeScript. У VSCode є безліч корисних функцій, таких як автодоповнення, налагодження коду, підтримка Git і багато іншого.
Крім того, були використані різні бібліотеки і пакети для полегшення розробки і вирішення конкретних завдань, такі як Axios для роботи з API, React Router для навігації, Redux для управління станом додатка і багато інших.

2.4. Опис використаних технологій та мов програмування

React був створений у 2011 році Джорданом Валке, розробником Facebook, у відповідь на проблеми, з якими стикаються розробники веб-додатків при створенні складних користувацьких інтерфейсів.

До появи React, розробники використовували традиційний підхід до створення користувацьких інтерфейсів, заснований на зміні HTML і CSS відповідно до дій користувача. Цей підхід ставав непрактичним під час роботи з великими і складними додатками, де користувацький інтерфейс часто змінюється і оновлюється.

React був розроблений як бібліотека для створення компонентів користувацького інтерфейсу, які можуть бути багаторазово використані та ефективно управляються. Він пропонує новий підхід до створення користувацьких інтерфейсів, заснований на декларативному програмуванні та компонентній архітектурі.

Основна ідея React полягає в тому, що користувальницький інтерфейс являє собою деревоподібну структуру компонентів, кожен з яких відповідає за окрему частину інтерфейсу. Ці компоненти можуть бути багаторазово використані й ефективно керовані, що робить створення і підтримку користувацького інтерфейсу набагато простішим і ефективнішим.

React швидко став популярним веб-фреймворком завдяки своїй простоті, модульності та високій продуктивності. Сьогодні React використовується багатьма великими компаніями і проектами, зокрема Facebook, Instagram, Netflix і Airbnb.

Плюси React:

- **Висока продуктивність завдяки віртуальному DOM.** React використовує віртуальний DOM, який дає змогу мінімізувати кількість операцій із реальним DOM і прискорити процес оновлення інтерфейсу. Це робить React одним із найшвидших фреймворків для створення користувацьких інтерфейсів.

- **Простота у використанні.** React має просту і зрозумілу структуру, що робить його легким у вивченні та використанні. Крім того, React використовує компонентний підхід, що дає змогу розбивати інтерфейс на окремі компоненти, що робить код більш читабельним і легко підтримуваним.
- **Відмінна документація та велике співтовариство розробників.** React має відмінну документацію та велике співтовариство розробників, що дає змогу швидко знаходити розв'язання проблем і отримувати допомогу в розробці.
- **Можливість використання на сервері.** React можна використовувати на стороні сервера, що дає змогу створювати універсальні додатки, які працюють як на клієнті, так і на сервері.

Мінуси React:

- **Відсутність стандартного підходу до управління станом.** React не надає стандартного підходу до управління станом, що може призвести до проблем під час розробки великих додатків.
- Для роботи з React необхідні навички роботи з JavaScript і JSX.** Для роботи з React необхідно мати певні навички роботи з JavaScript і JSX, що може бути проблемою для новачків у розробці.
- Необхідність використання сторонніх бібліотек для розв'язання деяких завдань.** Деякі завдання, як-от керування станом і маршрутизація, можуть вимагати використання сторонніх бібліотек, що може ускладнити процес розроблення та підтримки програми.

Приклад музичного плеєра на React з використанням TypeScript і Redux:

```
import React, { useState } from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { RootState } from './store/reducers';
import { playSong, pauseSong, setVolume } from './store/actions';
import './App.css';

function App() {
  const [isLoading, setIsLoading] = useState(true);
  const dispatch = useDispatch();
  const isPlaying = useSelector((state: RootState) => state.isPlaying);
  const volume = useSelector((state: RootState) => state.volume);

  const handlePlayPause = () => {
    if (isPlaying) {
      dispatch(pauseSong());
    } else {
      dispatch(playSong());
    }
  };

  const handleVolumeChange = (event: React.ChangeEvent<HTMLInputElement>) => {
    dispatch(setVolume(Number(event.target.value)));
  };

  const handleAudioLoad = () => {
    setIsLoading(false);
  };

  return (
    <div className="App">
      <h1>My Music Player</h1>
      <audio src="my_song.mp3" controls autoPlay={isPlaying} volume={volume/100} onLoadedData=
{handleAudioLoad} />
      {isLoading && <p>Loading...</p>}
      {!isLoading && (
        <>
          <button onClick={handlePlayPause}>{isPlaying ? 'Pause' : 'Play'}</button>
          <input type="range" min="0" max="100" value={volume} onChange={handleVolumeChange} />
        </>
      )}
    </div>
  );
}

export default App;
```

У цьому прикладі ми використовуємо Redux для управління станом програми. У стані ми зберігаємо інформацію про те, чи грає музика, і гучність звуку.
Коли користувач натискає на кнопку "Play/Pause", ми диспатчуємо відповідну дію, яка змінює значення стану `isPlaying` у Redux store. Коли користувач змінює значення повзунка гучності, ми також диспачуємо дію, яка змінює значення стану `volume`.

Ми також додали обробник події `onLoadedData`, щоб приховати текст "Loading..." після того, як аудіо-файл було завантажено.

TypeScript - це мова програмування з відкритим вихідним кодом, яка є розширенням JavaScript. Її розробила і підтримує компанія Microsoft, і її метою є спрощення процесу розроблення складних додатків і полегшення підтримки коду. TypeScript дає змогу програмістам використовувати сувору типізацію, класи, інтерфейси, модулі та інші функції, які не доступні в JavaScript.

TypeScript додає статичну типізацію до JavaScript, що полегшує розробку складних додатків і зменшує кількість помилок. Статична типізація дає змогу програмістам визначити типи даних, які використовуються в додатку, і виявляти помилки на етапі компіляції, а не на етапі виконання. TypeScript також має потужний механізм виведення типів, який дає змогу програмістам не вказувати типи явно в більшості випадків.

TypeScript також підтримує об'єктно-орієнтоване програмування, що дає змогу програмістам створювати більш складні та масштабовані додатки. Він дозволяє визначати класи, успадкування класів, абстрактні класи, інтерфейси та інші конструкції об'єктно-орієнтованого програмування. TypeScript також має функціональні можливості, які дають змогу програмістам використовувати функції вищого порядку і лямбда-вирази для спрощення коду.

TypeScript також має потужну систему модулів, яка дає змогу програмістам організовувати свій код в окремі модулі та імпортувати їх в інші модулі. Це полегшує підтримку коду і дає змогу програмістам розробляти додатки з великою кількістю файлів.

TypeScript підтримує безліч інструментів і функцій, які роблять його більш потужною мовою програмування, ніж JavaScript. Він також має чудову документацію та активну спільноту розробників, що полегшує процес вивчення та використання мови.

Крім того, TypeScript дає змогу легко інтегрувати наявний JavaScript-код у проекти, що робить його зручнішим для використання в наявних додатках. Це дає змогу програмістам поступово переходити на TypeScript, не переписуючи весь наявний код.

Загалом TypeScript є потужною і зручною мовою програмування, яка може допомогти програмістам створювати більш складні та масштабовані додатки, а також спрощувати процес розробки та полегшувати підтримку коду.

Плюси Typescript:

- Покращена надійність коду та зменшення помилок у процесі розробки. TypeScript додає статичну типізацію до JavaScript, що дає змогу програмістам визначити типи даних, що використовуються в додатку, і виявляти помилки на етапі компіляції, а не на етапі виконання. Це підвищує надійність коду та зменшує кількість помилок у процесі розроблення, що може суттєво прискорити розроблення додатка.
- Поліпшення читабельності коду та зменшення часу на його налагодження. TypeScript дає змогу програмістам використовувати інтерфейси та інші конструкції, які дають змогу описувати структуру даних і функцій, що може суттєво покращити читабельність коду та зменшити час на його налагодження.
- Спрощення масштабування проекту та зменшення часу на його розробку. TypeScript дає змогу програмістам створювати більш складні та масштабовані додатки, що може суттєво спростити процес розроблення та зменшити час на його виконання.
- Підтримка нових можливостей мови JavaScript. TypeScript дає змогу програмістам використовувати нові можливості мови JavaScript, як-от асинхронне програмування та декоратори, що може суттєво покращити якість і функціональність застосунку.
- Широке використання в індустрії та велика кількість інструментів і бібліотек для роботи з TypeScript. TypeScript широко використовується в індустрії та має велику спільноту розробників. Це забезпечує доступ до великої кількості інструментів і бібліотек для роботи з TypeScript, що може суттєво спростити процес розроблення додатка.

Мінуси Typescript:

- Необхідність вивчення додаткових концепцій, таких як типи даних та інтерфейси. TypeScript вимагає знання додаткових концепцій, таких як типи даних та інтерфейси, що може потребувати додаткового часу на вивчення мови.
- Збільшення часу на розробку через необхідність написання більшої кількості коду. TypeScript вимагає написання більшої кількості коду, що може збільшити час на розробку програми.
- Обмеження в сумісності з деякими інструментами та бібліотеками, які не підтримують TypeScript. Деякі інструменти та бібліотеки можуть не підтримувати TypeScript, що може обмежити вибір інструментів і ускладнити процес розроблення програми.

Node.js - це середовище виконання JavaScript, яке дає змогу запускати JavaScript-код на сервері. Node.js заснований на рушії V8, який розробляється компанією Google і використовується в браузері Google Chrome. Однак, на відміну від браузера Chrome, Node.js надає доступ до файлової системи, мережевих ресурсів та інших системних ресурсів, як-от доступ до баз даних, що дає змогу використовувати JavaScript для розроблення серверних додатків.

Одна з ключових переваг Node.js - це його асинхронна модель роботи, яка дає змогу обробляти велику кількість запитів на сервері, не блокуючи роботу інших запитів. Це досягається за допомогою використання подієвої моделі та не блокуючих операцій введення-виведення. Це робить Node.js ідеальним для розробки високонавантажених веб-додатків і API.

Node.js також дозволяє використовувати модулі, які дають змогу організовувати код в окремі компоненти та перевикористовувати його. Наприклад, за допомогою модулів можна розділити код на серверну та клієнтську частини, або на логіку застосунку та логіку бази даних.

Node.js має велику спільноту розробників, що забезпечує доступ до безлічі інструментів і бібліотек для розробки додатків. Це дає змогу швидко створювати та розвивати додатки на Node.js.

Крім того, Node.js підтримує пакетний менеджер npm, який дає змогу керувати залежностями застосунку і встановлювати додаткові модулі та бібліотеки. Це спрощує процес розробки і скорочує час на написання коду.

Загалом, Node.js є потужним і зручним середовищем виконання JavaScript для розробки серверних додатків. Його асинхронна модель роботи, підтримка модулів і велике співтовариство розробників роблять Node.js популярним вибором для розробки сучасних веб-додатків і API.

Плюси Node.js:

- Однопоточність, що робить процес розроблення простішим і зрозумілішим. Node.js працює в одному потоці, що полегшує процес розробки та зменшує ймовірність помилок. Також це дає змогу легко масштабувати додаток, не ускладнюючи код.
- Код на JavaScript, що полегшує роботу з фронтендом і сервером. Node.js дає змогу використовувати JavaScript на стороні сервера, що спрощує розробку і дає змогу легко перевикористовувати код між серверною та клієнтською частинами застосунку.
- Масштабованість завдяки асинхронній роботі. Node.js використовує асинхронну модель роботи, що дає змогу обробляти велику кількість запитів на сервері, не блокуючи роботу інших запитів. Це робить Node.js ідеальним для розробки високонавантажених веб-додатків та API.
- Велика кількість доступних бібліотек і модулів. Node.js має велику спільноту розробників, що забезпечує доступ до безлічі інструментів і бібліотек для розробки додатків. Це дає змогу швидко створювати та розвивати додатки на Node.js.

Мінуси Node.js:

- Не підходить для обчислювально складних завдань. Node.js не є найкращим вибором для обчислювально складних завдань, оскільки він працює в одному потоці.

- Немає підтримки багатопоточності. Node.js не підтримує багатопоточність, що може обмежити його використання в деяких випадках.
- Можливі проблеми з блокуванням введення-виведення. Іноді Node.js може зіткнутися з проблемами блокування введення-виведення, що може призвести до сповільнення роботи програми. Однак, існують інструменти для вирішення цієї проблеми.

MongoDB - це документоорієнтована NoSQL база даних, яка дає змогу зберігати дані у форматі документів BSON (Binary JSON). На відміну від реляційних баз даних, MongoDB не вимагає жорсткої схеми даних, що робить її гнучкішою і зручнішою у використанні для зберігання неструктурованих даних. MongoDB також має високу продуктивність і масштабованість, що дозволяє їй обробляти великі обсяги даних.
Приклад використання MongoDB у додатку музичного плеєра на React:
Додаток плеєра може використовувати MongoDB для зберігання інформації про композиції, плейлисти і прослухані треки. Наприклад, під час збереження плейлиста застосунок може створити документ у колекції "Плейлисти" зі списком композицій у форматі BSON. Під час відтворення композиції, додаток може завантажити інформацію про композицію з бази даних MongoDB і відобразити її на сторінці плеєра.

Переваги MongoDB:
- **Висока швидкість роботи**: MongoDB зберігає дані у форматі документів, що дає змогу швидко й ефективно обробляти запити. Вона також дає змогу розподіляти дані по декількох серверах для забезпечення більш високої продуктивності.
- **Гібкість у роботі з даними**: MongoDB дає змогу зберігати дані різних типів в одному документі та змінювати структуру документа в будь-який момент без необхідності зміни схеми бази даних. Це робить її гнучкішою й адаптивнішою до мінливих вимог проєкту.

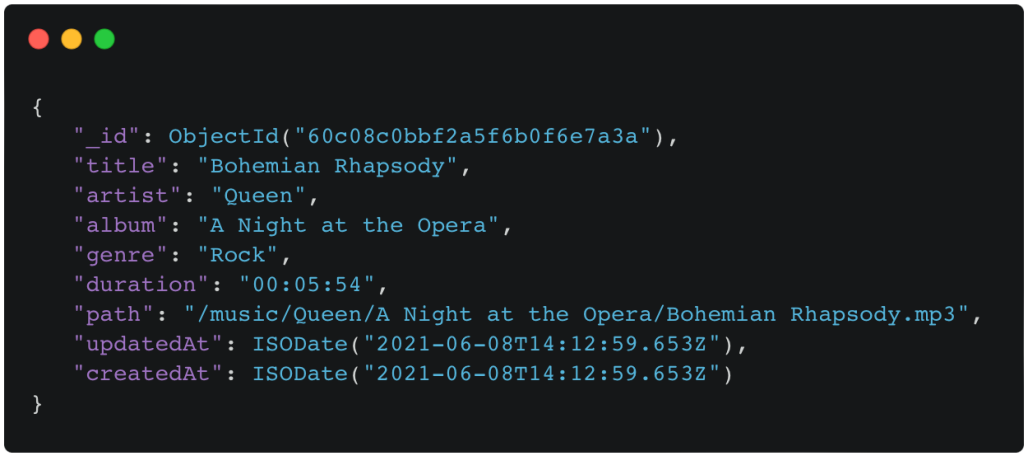
- **Легкість масштабування**: MongoDB дає змогу горизонтальне масштабування бази даних, що дає змогу розподіляти дані на кількох серверах для забезпечення більш високої продуктивності. Це також дає змогу легко збільшувати обсяг даних у базі даних за потреби.

Недоліки MongoDB:

- **Відсутність транзакцій**: MongoDB не підтримує транзакції, що може призвести до втрати даних під час збоїв. Це означає, що якщо станеться збій під час виконання операції, то деякі зміни можуть бути збережені, а інші - ні.
- **Потрібне додаткове програмне забезпечення для забезпечення безпеки даних**: MongoDB не надає вбудованих механізмів безпеки, таких як шифрування даних або аутентифікація користувача. Це означає, що для забезпечення безпеки даних необхідно використовувати додаткове програмне забезпечення.
- **Низька ефективність під час роботи з великими обсягами даних**: MongoDB може зіткнутися з проблемами продуктивності під час роботи з великими обсягами даних. Вона не підходить для обробки транзакційних даних, таких як банківські операції, які вимагають високого ступеня надійності та точності.

Таким чином, під час вибору MongoDB для проєкту необхідно враховувати як її переваги, так і недоліки, і приймати рішення залежно від конкретних вимог проекту.

Приклад колекції MongoDB для плеєра може мати такий вигляд:



```
{
    "_id": ObjectId("60c08c0bbf2a5f6b0f6e7a3a"),
    "title": "Bohemian Rhapsody",
    "artist": "Queen",
    "album": "A Night at the Opera",
    "genre": "Rock",
    "duration": "00:05:54",
    "path": "/music/Queen/A Night at the Opera/Bohemian Rhapsody.mp3",
    "updatedAt": ISODate("2021-06-08T14:12:59.653Z"),
    "createdAt": ISODate("2021-06-08T14:12:59.653Z")
}
```

SASS (Syntactically Awesome Style Sheets) - це препроцесор CSS, який дає змогу використовувати додаткові функції та можливості, що не доступні в стандартному CSS.

SASS надає безліч можливостей, таких як:

- Змінні: SASS дає змогу визначати змінні, які можуть містити кольори, шрифти, розміри блоків та інші параметри стилю. Це дає змогу швидко змінювати стиль і оформлення веб-сторінки.
- Вкладені правила: SASS дає змогу вкладати правила CSS одне в одне, що спрощує написання та підтримку стилів.
- Міксини: SASS дає змогу створювати міксини, які можуть містити групи правил CSS. Міксини можна використовувати для повторного використання коду і спрощення написання стилів.
- Успадкування: SASS дає змогу використовувати успадкування CSS, яке дає змогу успадковувати стилі від інших елементів.
- Функції: SASS надає безліч функцій, які можуть використовуватися для обробки даних і створення стилів.

Для використання SASS необхідно встановити компілятор, який перетворює SASS-код на стандартний CSS. Це можна зробити за допомогою спеціальних програм або плагінів для редакторів коду.
SASS - це потужний інструмент для створення стилів веб-сторінок, який спрощує написання та підтримку CSS.

Серед переваг SASS можна виділити такі:
- **Можливість використання змінних**. Однією з головних переваг SASS є можливість використання змінних, що спрощує написання і підтримку CSS-коду. Наприклад, можна визначити колірну палітру у вигляді змінних і потім використовувати ці змінні для визначення кольорів елементів на сторінці.
- **Можливість створення міксинів**. SASS дає змогу створювати міксини - набори правил CSS, які можна багаторазово використовувати в різних місцях. Наприклад, можна створити міксин для визначення стилю кнопок і потім використовувати його для створення всіх кнопок на сайті.
- **Легкість підтримки та зміни коду**. Завдяки використанню змінних і міксинів, SASS спрощує підтримку і зміну CSS-коду. Якщо потрібно змінити колір або шрифт на сайті, досить змінити значення змінної, яка визначає цей колір або шрифт, і зміни автоматично застосуються до всіх елементів, що використовують цю змінну.
- **Зручний синтаксис**. SASS має зручний синтаксис, який спрощує читання і написання коду. Наприклад, можна використовувати вкладеність правил CSS, щоб поліпшити читабельність коду.

Однак, деякі мінуси SASS можуть включати такі:
- **Необхідність додаткового налаштування та встановлення**. Для використання SASS необхідно встановити додаткове програмне забезпечення та налаштувати його на роботу з проектом.
- **Можливість виникнення помилок**. Використання SASS може призвести до виникнення помилок, якщо функції та міксини використані неправильно. Це

може призвести до помилок у відображенні сторінки та ускладнити їхній пошук і виправлення.

- **Можливість ускладнення коду**. Використання SASS може призвести до збільшення обсягу CSS-коду, особливо якщо розробник не має достатнього досвіду роботи з препроцесорами.

Крім того, SASS може не підходити для проектів, які не вимагають великого обсягу CSS-стилів або для проектів, які вже використовують інший препроцесор CSS.

2.5. Опис структури програми та алгоритмів її функціонування

React працює за принципом односторінкового додатку, який називається Single-Page Application (SPA).

Це архітектурний підхід, при якому усі дані виводяться на одній сторінці, динамічно змінюючись, а сам додаток або сайт не перезавантажується. Це збільшує швидкість роботи та скорочує передаваємий об’єм даних між браузером та сервером.

Головний файл index.html відповідає за завантаження на сторінку div елементу з id=’root’ та встановлення деяких елементів, таких як назва сайту на вкладці браузера, підключення скриптів та інші.

Після цього підключаються основні бібліотеки React та ReactDOM. ReactDOM - це пакет, який надає методи для взаємодії з DOM (Document Object Model) у контексті React-додатків. Він дозволяє рендерити компоненти React у віртуальному DOM та вставляти їх у справжній DOM сторінки.

Усі елементи веб-додатку завантажуються з компоненту App.js, у якого також є свої дочірні компоненти.

2.6. Обгрунтування та організація вхідних та вихідних даних програми

В даній кваліфікаційній роботі оброблюються дані користувача, які він завантажує до додатку у форматі аудіофайлів. Оскільки додаток призначений тільки для відтворення файлів, ніяких маніпуляцій з ними не відбувається. Вихідними даними є відтворені додатком звукові хвилі.

2.7. Опис роботи розробленого програмного продукту

Для взаємодії з додатком, потрібен лише браузер та IDE, у моєму випадку Visual Studio Code. Завдяки тому, що React має в собі встановлений локальний сервер, аудіоплеєр після компіляції буде запускатися просто у новій вкладці браузеру.

2.7.1. Використані технічні засоби

Для забезпечення користувачу зручної роботи з сучасними браузерами, розробники рекомендують мати наступні мінімальні параметри комп'ютера:

1) Центральний процесор (ЦП): Pentium 4 або еквівалент.
2) Відеоадаптер: 3D адаптер від nVidia, Intel, AMD/ATI.
3) Відеопам'ять: мінімум 128 МБ відеопам'яті.
4) Накопичувач: щонайменше 150 ГБ вільного простору на жорсткому диску.
5) Оперативна пам'ять: мінімум 2 ГБ оперативної пам'яті.

Ці рекомендації допоможуть забезпечити плавну роботу браузера і відтворення веб-контенту на вашому комп'ютері.

2.7.2. Використані програмні засоби

При розробці цього проекту було використано наступні програмні засоби:

- Visual Studio Code;
- Node.js

Visual Studio Code (VS Code) – це інтегроване середовище розробки, створене компанією Microsoft. Воно призначене для роботи з програмним кодом у різних мовах програмування та надає широкий спектр функцій, що полегшують розробку програмного забезпечення.

Основні особливості Visual Studio Code:

1) Підтримка різних мов програмування: VS Code надає підтримку для багатьох популярних мов програмування, таких як JavaScript, Python, C#, Java, HTML, CSS і багато інших. Завдяки цьому, розробники можуть працювати над проектами різного типу в одному редакторі.
2) Розширення та налаштування: VS Code дозволяє розширити його функціональність за допомогою розширень, які розробляються спільнотою. Розширення дозволяють додавати нові можливості, підтримку для конкретних технологій, інструменти для розробки та інше. Крім того, користувачі можуть налаштовувати редактор за своїми потребами.
3) Інтеграція з Git: VS Code має вбудовану підтримку системи контролю версій Git. Це дозволяє розробникам здійснювати коміти, переглядати та порівнювати зміни, створювати та об'єднувати гілки, працювати з репозиторіями та багато іншого, все безпосередньо з редактора.
4) Розширені можливості пошуку та заміни: VS Code надає потужні інструменти для пошуку та заміни тексту в проекті. За допомогою регулярних виразів та різних параметрів пошуку, можна легко знаходити та замінювати текст у великих обсягах коду.

5) Відладка коду: редактор підтримує можливість відладки коду для різних мов програмування. За допомогою спеціальних розширень та налаштувань, розробники можуть ставити точки зупинки, переглядати значення змінних, виконувати кроки виконання коду та інше.

Він доступний на різних операційних системах, включаючи Windows, macOS та Linux.

Node.js - це середовище виконання JavaScript, яке дає змогу запускати JavaScript-код на сервері. Одна з ключових переваг Node.js - це його асинхронна модель роботи, яка дає змогу обробляти велику кількість запитів на сервері, не блокуючи роботу інших запитів. Це досягається за допомогою використання подієвої моделі та не блокуючих операцій введення-виведення. Це робить Node.js ідеальним для розробки високонавантажених веб-додатків і API. Node.js також дозволяє використовувати модулі, які дають змогу організовувати код в окремі компоненти та перевикористовувати його.

2.7.3. Виклик та завантаження програми

Розроблений веб-застосунок може бути доступний в мережі Інтернет за допомогою стандартизованої адреси (URL: http://localhost:3000) через сучасні браузери, що підтримуються на різних операційних системах, наприклад: Google Chrome, Opera, Mozilla Firefox тощо.

2.7.4. Тестування

Для проведення тестування додатка на різних рівнях будуть використовуватися такі інструменти:

1. **Модульне тестування** - тестування окремих модулів додатка за допомогою Jest. Це дасть змогу перевірити роботу кожного компонента та функції додатка в ізоляції від інших компонентів.

2. **Інтеграційне тестування** - тестування взаємодії компонентів і модулів між собою, а також із зовнішніми системами. Для цього будуть використовуватися як автоматизовані, так і ручні тести.
3. **Тестування користувальницького інтерфейсу** - тестування користувальницького інтерфейсу на відповідність вимогам і очікуванням користувачів. Для цього будуть використовуватися інструменти, такі як Enzyme і Selenium.
Крім того, будуть використовуватися методи функціонального тестування, які дадуть змогу перевірити роботу застосунку в умовах, максимально наближених до реальних.

Результати тестування документуватимуться у спеціально створеному звіті, який міститиме інформацію про проведені тести, виявлені помилки та недоліки, а також рекомендації щодо поліпшення програми.
Мета тестування - не тільки перевірити працездатність застосунку, а й поліпшити його якість і зручність використання для користувачів.

Після проведення модульного, інтеграційного та функціонального тестування застосунку, а також тестування призначеного для користувача інтерфейсу на різних пристроях і браузерах було виявлено кілька незначних помилок, пов'язаних із деякими функціями застосунку. Усі помилки були успішно виправлені, і повторне тестування не виявило жодних проблем із роботою програми.

Під час тестування додатка на продуктивність було зафіксовано, що він працює швидко і без затримок під час обробки великого обсягу даних. Було застосовано методи оптимізації запитів до бази даних і кешування даних для вирішення проблеми з обробкою великого обсягу даних. Також було проведено тестування на надійність і зручність використання програми. Користувачі високо оцінили зручність інтерфейсу і швидкий відгук програми на дії користувача.

У результаті, всі тести були успішно пройдені, і додаток готовий до використання. Звіт про тестування містить інформацію про проведені тести, виявлені помилки та недоліки, а також рекомендації щодо поліпшення програми.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Вихідні дані:

1. передбачуване число операторів програми – 1120;

2. коефіцієнт складності програми – 1,55;

3. коефіцієнт корекції програми в ході її розробки – 0,2;

4. годинна заробітна плата розробника – 150 грн/год

5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;

6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,1;

7. вартість машино-години ЕОМ – 18 грн/год (1 грн – е/е, 10 грн – ПЗ, 7 грн -амортизація).

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки. Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,}$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p),$$

де q – передбачуване число операторів (1120);

C – коефіцієнт складності програми (1,55);

p – коефіцієнт корекції програми в ході її розробки (0,2).

$$Q = 1120 \cdot 1{,}55 \cdot (1 + 0{,}2) = 2217{,}2;$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75...85) \cdot K}, \text{ людино-годин}$$

де В – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі (1,2);

К – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності (1,1);

$$t_u = \frac{2217{,}2 \cdot 1{,}2}{80 \cdot 1{,}1} = 38{,}25 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20...25) \cdot K};$$

$$t_a = \frac{2217{,}2}{20 \cdot 1{,}1} = 10{,}54 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)\cdot K};$$

$$t_n = \frac{2217,2}{25\cdot 1,1} = 80 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4...5)\cdot K};$$

$$t_{отл} = \frac{2217,2}{5\cdot 1,1} = 401 \text{ людино-годин.}$$

– за умови комплексного налагодження завдання:

$$t_{отл}^{к} = 1,2 \cdot t_{отл};$$

$$t_{отл}^{к} = 1,2 \cdot 401 = 481 \text{ людино-годин.}$$

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до};$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису;

$$t_д = \frac{Q}{(15...20)\cdot K};$$

$$t_{др} = \frac{2217.2}{20\cdot 1,1} = 100,78 \text{ людино-годин.}$$

де $t_{до}$ – трудомісткість редагування, печатки й оформлення документації;

$$t_{до} = 0{,}75 \cdot t_{др};$$

$t_{до}$ = 0,75 · 100,78 = 75,58 людино-годин.

$t_{д}$ = 100,78 + 75,58 = 176,36 людино-годин.

Отримаємо трудомісткість розробки програмного забезпечення:

t = 50 + 38,25 + 10,54 + 80 + 401 + 176,36 = 756,15 людино-годин.

У результаті ми розрахували, що в загальній складності необхідно 756,15 людино-годин для розробки даного програмного забезпечення.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ $Кпо$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$Кпо = З_{ЗП} + З_{МВ}, \text{ грн},$$

$З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн},$$

де t – загальна трудомісткість, людино-годин;

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година.

З урахуванням того, що середня годинна зарплата розробника становить 115 грн/год, то отримаємо:

$$З_{ЗП} 756,15 \cdot 150 = 113422,5 \text{ грн}$$

Вартість машинного часу $З_{МВ}$, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{МВ} = t_{отл} \cdot C_М \text{, грн,}$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год;

$C_М$ – вартість машино-години ЕОМ, грн/год.

$$З_{МВ} = 401 \cdot 18 = 7218 \text{ грн}$$

Звідси витрати на створення програмного продукту:

$$Кпо = 113422,5 + 7218 = 120640,5 \text{ грн}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p} \text{, мес.}$$

де B_k – число виконавців;

F_p – місячний фонд робочого часу (при 40 годинному робочому тижні F_p = 176 годин).

Витрати на створення програмного продукту:

$$T = \frac{756,15}{1 \cdot 176} = 4,29 \text{ міс.}$$

Вартість розробки інтернет-магазину становить 113422,5 грн. Час розробки очікується приблизно 4,29 місяці при стандартному 40-годинному робочому тижні і 176-годинному робочому місяці. Цей термін включає час, необхідний для

дослідження, розробки алгоритму, дизайну і документування. Загальна кількістю людино-годин, яку буде витрачено на розробку – 756,15.

ВИСНОВКИ

У результаті роботи було створено веб-додаток музичного плеєра на React з використанням TypeScript і сервера на Node Js. Під час розроблення було вивчено основні особливості React, TypeScript і NodeJs, обрано необхідні інструменти (наприклад, Redux) і технології для реалізації проекту. Було проведено роботу з проектування архітектури застосунку та опису його функціональності.

Основний функціонал застосунку полягає в можливості програвання музики, пошуку музичних композицій, складанні плейлистів і зберіганні інформації про прослухані треки та плейлисти. Для реалізації цього функціоналу було реалізовано відповідні компоненти та функції, а також було написано серверний код для обробки запитів від клієнта.

Важливим етапом роботи було тестування застосунку. Для цього було використано такі інструменти: модульне тестування (Jest), інтеграційне тестування (автоматизовані та ручні тести) і тестування користувацького інтерфейсу (Enzyme і Selenium). Також було проведено функціональні тести для перевірки роботи застосунку в умовах, максимально наближених до реальних.

У результаті тестування було виявлено незначні помилки, пов'язані з деякими функціями застосунку, які було успішно виправлено. Також були зафіксовані позитивні результати при тестуванні продуктивності, надійності та зручності використання додатка.

У висновку було описано результати роботи над проектом, а також можливості доопрацювання додатка та розширення його функціональності. Усі результати роботи були задокументовані в спеціально створеному звіті, який містить інформацію про проведені тести, виявлені помилки та недоліки, а також рекомендації щодо поліпшення програми.

Конкретніше, можна розглянути такі напрямки розвитку додатка:

- **Додавання можливості створення та редагування плейлистів:** це дасть змогу користувачам створювати персональні добірки музики, з урахуванням

їхніх особистих вподобань та настрою. Для цього можна додати відповідний функціонал у користувацький інтерфейс і написати відповідний серверний код.

- **Інтеграція з сервісами потокового передавання музики:** це дасть змогу користувачам слухати музику з різних джерел, таких як Spotify, Apple Music, Google Play тощо. Для цього потрібно інтегрувати відповідні API та розробити відповідний функціонал у користувацькому інтерфейсі.

- **Автоматичне оновлення інформації про музичні треки та виконавців:** це дасть змогу застосунку автоматично оновлювати інформацію про нові музичні треки та виконавців, що забезпечить користувачів завжди свіжим контентом. Для цього можна використовувати відповідні API та написати відповідний серверний код.

- **Покращення користувацького інтерфейсу:** можна продовжувати роботу над поліпшенням користувацького інтерфейсу, з урахуванням зворотного зв'язку від користувачів. Наприклад, можна поліпшити навігацію, дизайн і візуальне оформлення, а також додати нові функції для зручності використання.

- **Оптимізація застосунку:** можна продовжувати роботу над оптимізацією застосунку для поліпшення продуктивності та зручності використання. Наприклад, можна оптимізувати запити до бази даних, використовувати кешування даних і покращувати алгоритми обробки даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ReactJS. URL: https://reactjs.org/
2. JavaScript. URL: https://uk.javascript.info
3. Eric Elliott. Programming JavaScript Applications: Robust Web Architecture with Node, HTML5, and Modern JS Libraries. — O'Reilly Media, 2014. — 254 c.
4. Robin Wieruch. The Road to React: Your journey to master plain yet pragmatic React.js. — Independently published, 2018. — 394 c.
5. Eric Freeman, Elisabeth Robson. Head First JavaScript Programming: A Brain-Friendly Guide. — O'Reilly Media, 2014. — 704 c.
6. Marijn Haverbeke. Eloquent JavaScript: A Modern Introduction to Programming. — No Starch Press, 2018. — 472 c.
7. David Flanagan. JavaScript: The Definitive Guide. — O'Reilly Media, 2020. — 706 c.
8. MySQL. URL: https://www.mysql.com
9. Jon Duckett. HTML & CSS: Design and Build Websites. — Wiley, 2011. — 512 c.
10. Rachel Andrew. The New CSS Layout. — Smashing Magazine, 2017. — 218 c.
11. Dan Cederholm. CSS3 For Web Designers. — A Book Apart, 2010. — 112 c.

ДОДАТОК А

КОД ПРОГРАМИ

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Player</title>
</head>
<body>
  <div id="root"></div>
</body>
</html>
```

```
import * as React from 'react'
import * as ReactDOM from 'react-dom'
import { combineReducers, createStore, applyMiddleware } from 'redux'
import { Provider } from 'react-redux'
import { App } from './components'
import { socket, music } from './reducers'
import './main.scss'

const rootReducer = combineReducers({
  socket,
  music,
})

const STORE = createStore(rootReducer)

ReactDOM.render(
  <Provider store={STORE}>
    <App />
  </Provider>,
  document.getElementById('root')
)
```

```
import * as React from 'react'
import { Player } from '../components'
import { connect } from 'react-redux'
import { bindActionCreators } from 'redux'
import { NEW_MUSIC, GET_MUSIC, SEND_MUSIC } from '../../server/constants/music'
import { saveMusic } from '../actions'
import { MusicData } from '../intefaces/music'

interface IProps {
  socket: SocketIOClient.Socket;
  musicData: MusicData;
  saveMusic: Function;
}

export class AppComponent extends React.PureComponent<IProps, any> {

  render() {
    const { musicData } = this.props
    return (
      <div>
        <Player
          musicData={musicData}
        />
      </div>
    )
  }

  // LIFE CYCLE METHODS

  componentDidMount() {
    const { socket } = this.props
    socket.on(NEW_MUSIC, this.handleServerHasNewMusic)
    socket.on(SEND_MUSIC, this.handleServerSendMusic)
  }

  // HANDLE METHODS
  handleServerHasNewMusic = () => {
    const { socket } = this.props
    socket.emit(GET_MUSIC)
  }

  handleServerSendMusic = (newMusic: any) => {
    const { saveMusic } = this.props
    saveMusic(newMusic)
  }
}


function mapStateToProps(state: any) {
  return {
```

```
    socket: state.socket,
    musicData: state.music,
  }
}

function mapDispatcheToProps(dispatch: any) {
  return {
    saveMusic: bindActionCreators<any>(saveMusic, dispatch),
  }
}

export const App = connect(mapStateToProps, mapDispatcheToProps)(AppComponent)
```

```
import * as React from 'react'
import { MusicData, INewTrack, IMusicMeta } from '../../intefaces/music'
import './Player.scss'

interface IProps {
  musicData: MusicData;
}

interface IState {
  filterValue: string;
  playbackButtonState: string;
  activeTrack: INewTrack;
  localMusic: Array<any>;
  currentTime: number;
}

const PLAYBACK_STATUS = {
  play: 'play',
  stop: 'stop',
  waiting: 'waiting',
}

const SIDES = {
  prev: 'prev',
  next: 'next',
}

const DEFAULT_ACTIVE_TRACK = {
  artist: '',
  name: '',
  fullname: '',
```

```
  duration: 0,
}

/**
 * @desc Компонент кастомного плеера с плейлистом
 */
export class Player extends React.Component<IProps, IState> {
  audioNode: HTMLAudioElement;
  meta: IMusicMeta;

  constructor(props: IProps) {
    super(props)
    this.state = {
      filterValue: '', // Состояние поля фильтра
      playbackButtonState: PLAYBACK_STATUS.play, // Состояние кнопки ПУСК/ПАУЗА
      activeTrack: DEFAULT_ACTIVE_TRACK, // Данные об активной песне (исполнитель,
название, длительность)
      localMusic: [], // При получении музыки извне, компонент добавляет разнные данные
к песням и хранит у себя в состоянии
      currentTime: 0, // Время песни (каком месте остановился плеер)
    }
  }

  render() {
    const {
      props,
      state,
      renderMusicList,
      filterMusic,
      handleChangeFilterInput,
    } = this
    const {
      filterValue,
      playbackButtonState,
      localMusic,
      activeTrack,
      currentTime,
    } = state

    return (
      <div className='player'>
        <audio
          ref={this.saveAudioRefAndAddListeners}
          src={this.createFullPathToSong(activeTrack.fullname)}
        />
        <div className='player__controls'>
          <div
            className='player__prev'
            onClick={this.handleClickPrev}
          >
```

```
            {'<'}
          </div>
          <div
            className={`player__playback-button player__playback-button--
${playbackButtonState}`}
            onClick={this.handleClickPlaybackButton}
          >
              {playbackButtonState}
          </div>
          <div
            className='player__next'
            onClick={this.handleClickNext}
          >
            {'>'}
          </div>
          <div className='player__progress-bar-wrapper'>
            <div
              className='player__progress-bar'
              onClick={this.handleClickProgressBar}
            >
              <div
                className='player__progress-bar--progress'
                style={this.getProgressStyle(currentTime, activeTrack.duration)}
              ></div>
            </div>
          </div>
          <div className='player__time'>
            <div className='player__current-
time'>{this.formatSongTime(currentTime)}</div>
            <div className='player__time-separator'>/</div>
            <div className='player__total-
time'>{this.formatSongTime(activeTrack.duration)}</div>
          </div>
          <div className='player__volume'>
            <input
              className='player__volume-range'
              type='range'
              min='0.01'
              max='1'
              step='0.01'
              defaultValue='0.5'
              onChange={this.handleChangeVolume}
            />
          </div>
        </div>
        <input
          className='player__filter-input'
          placeholder='Search for artists or tracks'
          type="text"
          value={filterValue}
```

```
          onChange={handleChangeFilterInput}
        />
        { renderMusicList(filterMusic(filterValue, localMusic)) }
      </div>
    )
  }

  // LIFE CYCLE METHODS

  componentWillReceiveProps(newProps: any): void {

    // Сохраняем различные метаданные, добавляем к полученным песням данные о их
длительности
    this.meta = newProps.musicData.meta
    this.addDurationToLocalMusic(newProps.musicData.music)

  }

  componentDidUpdate() {

    // Добавляем первый в списке трек в состояние воспоизведения
    this.setDefaultTrackToState()
  }


  // HANDLE METHODS

  handleChangeVolume = (e: any): void => {
    this.audioNode.volume = e.target.value
  }

  handleClickProgressBar = ({nativeEvent, target}: any): void => {
    const { activeTrack: {duration}, playbackButtonState } = this.state

    // Вычисляем, на каком расстоянии нажал пользователь и меняем время песни
    this.audioNode.currentTime = nativeEvent.offsetX / target.parentElement.clientWidth
* duration

    // Если плеер на паузе, то перевести в состояние воспроизведения
    if (playbackButtonState === PLAYBACK_STATUS.play) this.play()
  }

  handleClickPrev = (): void => {
    this.putTrackBySide(SIDES.prev)
  }

  handleClickNext = (): void => {
    this.putTrackBySide(SIDES.next)
  }
```

```
  handleEndedAudio = (): void => {
    this.putTrackBySide(SIDES.next)
  }

  handleChangeFilterInput = (e: any): void => {
    this.setState({
      filterValue: e.target.value,
    })
  }

  handleClickPlaybackButton = (): void => {
    this.togglePlaybackButton()
  }

  handleClickOnTrack = (track: INewTrack) => (): void => {
    const {activeTrack} = this.state
    if (activeTrack.fullname === track.fullname) {
      this.togglePlaybackButton()
      return;
    }
    this.setDefaultButtonState()
    this.playTrack(track)
  }

  handleTimeUpdate = (): boolean => {
    const { currentTime } = this.state
    const audioTime: number = Math.floor(this.audioNode.currentTime)
    if (currentTime === audioTime) return false;
    this.setState({
      currentTime: audioTime,
    })
  }


  // RENDER METHODS

  //TODO: Перевести обработчик клика на список

  /**
   * @desc Создает jsx-элемент песни в плейлисте
   * @param {INewTrack} songData Данные о песне
   * @return {JSX.Element}
   */
  renderTrackItem = (songData: INewTrack): JSX.Element => {
    const { activeTrack } = this.state
    const itemClass: string = `player__item ${activeTrack.fullname ===
songData.fullname ? 'player__item--active': ''}`
    return (
      <li
        className={itemClass}
```

```
        key={songData.fullname}
        onClick={this.handleClickOnTrack(songData)}
      >
        <div className='player__item-song'>
          { this.getSongName(songData) }
        </div>
        <div className='player__item-duration'>
          { this.formatSongTime(songData.duration) }
        </div>

      </li>
    )
  }

  renderMusicList = (music: Array<INewTrack> = []): JSX.Element => {
    const { filterValue } = this.state
    const classList: string = `player__list ${music.length === 0 && filterValue === ''
? 'player__list--loader' : ''}`
    return (
      <ul className={classList}>
        {
          music.map((track: any) => this.renderTrackItem(track))
        }
      </ul>
    )
  }



  // CUSTOM METHODS

  /**
   * @desc Вычесляем ширину полосы воспроизведения песни
   * @param {number} currentTime Текущее время песни в секундах
   * @param {duration} currentTime Общее время песни в секундах
   * @return {Object}
   */
  getProgressStyle = (currentTime: number, duration: number) => {
    return {
      width: currentTime / duration * 100 + '%',
    }
  }

  //TODO: Сделать более декларативным

  /**
   * @desc В зависимости от выбранной стороны, воспроизвести песню слева или справа
   * @param {string} side С какой стороны воспроизвести песню
   */
  putTrackBySide = (side: string) => {
```

```
    const { activeTrack, localMusic } = this.state

    // Узнаем под каким индеком в массиве всех песен находится активная
    const indexActiveSong: number = localMusic.findIndex(songData => songData.fullname
=== activeTrack.fullname)

    if (indexActiveSong < 0 || localMusic.length === 0) {
      console.error('Не найдена песня или их негде искать')
      return false
    }

    let newSong: INewTrack;
    switch(side) {
      case SIDES.next:
        newSong = localMusic[indexActiveSong + 1] || localMusic[0]
        break
      case SIDES.prev:
        newSong = localMusic[indexActiveSong - 1] || localMusic[localMusic.length - 1]
        break
    }
    this.setState({
      activeTrack: newSong,
    }, this.play)
  }

  /**
   * @desc Сохраняет ссылку на dom-узел тега <audio/> и подписывается на события
   */
  saveAudioRefAndAddListeners = (ref: any) => {
    this.audioNode = ref;
    this.addListeners()
  }

  addListeners = () => {
    if (!this.audioNode) return

    // Когда песня закончилась
    this.audioNode.onended = this.handleEndedAudio

    // Кодга время песни изменилось
    this.audioNode.ontimeupdate = this.handleTimeUpdate
  }

  /**
   * @desc В зависимости от состояния, делать кнопку воспроизведения либо "ПАУЗА", либо
"ПУСК".
   * В соответствии с этим запускать песню или ставить на паузу
   */
  togglePlaybackButton = () => {
    switch(this.state.playbackButtonState) {
```

```
    case PLAYBACK_STATUS.play:
      this.play()
      break;
    case PLAYBACK_STATUS.stop:
      this.pause()
      break;
  }
}

setDefaultButtonState = () => {
  this.setState({
    playbackButtonState: PLAYBACK_STATUS.play,
  })
}

playTrack = (track: INewTrack) => {
  this.setState({
    activeTrack: track,
  }, this.togglePlaybackButton)
}

play = () => {
  const { activeTrack } = this.state
  if (activeTrack !== DEFAULT_ACTIVE_TRACK) {
    this.audioNode.play()
    this.setState({
      playbackButtonState: PLAYBACK_STATUS.stop,
    })
  }
}

pause = () => {
  this.audioNode.pause()
  this.setState({
    playbackButtonState: PLAYBACK_STATUS.play,
  })
}

/**
 * @desc Получить название песни для плейлиста из общих данных о ней
 */
getSongName = ({ artist, name, fullname}: INewTrack): string  => {
  return (artist && name)
  ? this.getBeautySongString(artist, name)
  : this.getSongWithoutFormat(fullname)
}

/**
 * @desc Получить из полного названия песни все, кроме формата.
 * Например: Alfi_Goggy-start.mp3 -> Alfi_Goggy-start
```

```
   */
  getSongWithoutFormat = (songName: string): string => {
    return songName.slice(0, songName.indexOf('.'))
  }

  /**
   * @desc Делает из исполнителя и названия более красивую строку. Например: Prodigy -
God
   */
  getBeautySongString = (artist: string, name: string): string  => {
    return artist + ' - ' + name
  }

  filterMusic = (filterValue: string = '', music: Array<INewTrack> = []): Array<any> =>
{
    return music.filter(
      (songData: INewTrack) => this
        .getSongName(songData)
        .toLowerCase()
        .includes(filterValue.toLowerCase().trim())
    )
  }

  setDefaultTrackToState = () => {
    const song: INewTrack = this.getFisrtTrack()
    if (this.state.activeTrack === DEFAULT_ACTIVE_TRACK && song) {
      this.setState({
        activeTrack: song,
      })
    }
  }

  getFisrtTrack = (): INewTrack => {
    const { localMusic } = this.state

    if (localMusic.length === 0) return null

    return localMusic[0]
  }

  /**
   * @desc Составляет путь к файлу, чтобы браузер мог его получить в виде статики
   */
  createFullPathToSong = (songName: string = ''): string => {
    if (!songName || typeof this.meta.path !== 'string') return ''

    return this.meta.path + songName
  }

  /**
```

```
   * @desc Добавить к песням в плейлисте их длительность
   */
  addDurationToLocalMusic = (newMusic: Array<INewTrack>): void => {
    Promise
      .all(
        newMusic.map(
          async song => Object.assign({}, song,
            {
              duration: await this.getSongDuration(song.fullname),
            }
          )
        )
      )
      .then(musicWithDuration => {
        this.setState({
          localMusic: musicWithDuration,
        })
      })
  }

  /**
   * @desc Получаем длительность песни на основе его метаданных
   */
  getSongDuration(songName: string = ''): Promise<number> {
    const path: string = this.createFullPathToSong(songName)

    if (path === '') return Promise.resolve(0)

    return new Promise((resolve, reject) => {
      const audio = new Audio(path)
      audio.onloadedmetadata = () => {
        resolve(audio.duration)
      }
    })
  }

  /**
   * @desc Форматирует время в секундах в более человеческий вид. Например: 123 ->
02:03
   */
  formatSongTime = (time: number): string => {
    const separator: string = ':'
    const minutes: number = Math.floor(time / 60)
    const seconds: number = Math.floor(time % 60)
    return `${minutes < 10 ? '0' + minutes : minutes}` + separator + `${seconds < 10 ?
'0' + seconds : seconds}`
  }
}
```

```
const app = new (require('koa'))()
const server = require('http').createServer(app.callback())
const io = require('socket.io')(server)
const { handleServerIsStarted } = require('./helpers')
const CONFIG = require('./config')
const serverIO = new (require('./ServerIO'))(io)

server.listen(CONFIG.port, handleServerIsStarted)
```

```
// TODO: Заменить NodeID3 на что-то другое. Падает при большом количестве треков
const NodeID3 = require('node-id3')

const path = require('path')
const { isEqualArrays, getFileNamesInDir } = require('./helpers')
const { NEW_MUSIC, GET_MUSIC, SEND_MUSIC } = require('./constants/music')
const CONFIG = require('./config')

module.exports = class ServerIO {
  constructor(io) {
    this.socket = null
    this.currentMusicNames = []

    this.init = this.init.bind(this)
    this.handleIOConnection = this.handleIOConnection.bind(this)
    this.checkMusicFolderWithInterval = this.checkMusicFolderWithInterval.bind(this)
    this.checkOnChangeMusicFolder = this.checkOnChangeMusicFolder.bind(this)
    this.emit = this.emit.bind(this)
    this.formatMusicNames = this.formatMusicNames.bind(this)
    this.handleGetMusic = this.handleGetMusic.bind(this)


   io.on('connection', this.handleIOConnection)
  }

  init() {
    this.currentMusicNames = []
    this.checkMusicFolderWithInterval()
    this.socket.on(GET_MUSIC, this.handleGetMusic)
```

```
  }

  /**
   * @desc Когда клиент просит список песен, то эта функция отправляет песни в нужном
виде
   */
  async handleGetMusic() {
    const formatedMusic = await this.formatMusicNames(this.currentMusicNames)
    this.socket.emit(SEND_MUSIC, formatedMusic)
  }

  handleIOConnection(socket) {
    this.socket = socket
    this.init()
  }

  /**
  * @desc Запускает слежку за папкой с музыкой
  * @param { number } interval in ms. Default value equals 5s
  * @return { number } Returns interval id
  */
  checkMusicFolderWithInterval(interval = 5e3) {
    return setInterval(this.checkOnChangeMusicFolder, interval)
  }

  /**
   * @desc Сравнивает новые данные папки с музыкой, если есть различия - сказать
клиенту об этом
   */
  async checkOnChangeMusicFolder() {
    const newMusicNames = await getFileNamesInDir(CONFIG.paths.musicFullPath)
    if (isEqualArrays(newMusicNames, this.currentMusicNames) === false) {
      this.currentMusicNames = newMusicNames;
      this.emit(NEW_MUSIC)
    }
  }

  /**
   * @desc Emit action to SocketIO
   * @param { string } action
   * @param { any } payload
   */
  emit(action, payload = null) {
    this.socket.emit(action, payload)
  }

  /**
   * @desc Приводит названия песен в обьекты с метаданными и отдает
   * @param { Array<string> } musicNames
   * @return { Array<Object> }
```

```
   */
  async formatMusicNames(musicNames = []) {
    //TODO: перевести с императивного стиля в более декларативный

    // FIXME: Обернуть async в try/catch
    const formatedMusic = await Promise.all(
      musicNames.map( trackName => {
        return new Promise((resolve, reject) => {
          const fullFilePath = path.join(CONFIG.paths.musicFullPath, trackName)
          NodeID3.read(fullFilePath, (err, tags) => {
            if (err) {
              console.log(err)
              reject(err)
            }

            resolve({
              artist: tags.artist || '',
              name: tags.title || '',
              fullname: trackName,
            })
          })
        })
      })
    )
    return {
      meta: {
        path: CONFIG.paths.musicDirName + '/', // Example: 'music/'
      },
      music: formatedMusic,
    }
  }

}
. . .
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом_ Колесник.docx	Пояснювальна записка до дипломного проекту. Документ Word.
Диплом_ Колесник.pdf	Пояснювальна записка до дипломного проекту в форматі PDF
Програма	
Program.zip	Архів. Містить коди програми і відкомпільовану програму
Презентація	
Презентація_Колесник.pptx	Презентація дипломного проекту