

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

**«Веб-додаток магазину вінілових платівок
«Musical Corner» працівники,
продукти та жанри»**

Виконав студент групи ПЗ-19-2
ДЕМЧЕНКО Ярослав Дмитрович

Керівник роботи:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
ЛОБАНЧИКОВА Надія Миколаївна

Житомир – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

В.о. зав. кафедри інженерії
програмного забезпечення
Андрій МОРОЗОВ
«14» лютого 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ДЕМЧЕНКО Ярослав Дмитрович**

Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**

Тема роботи: **«Веб-додаток магазину вінілових платівок «Musical Corner»
працівники продукти та жанри»**,

затверджена Наказом закладу вищої освіти від **«14» лютого 2023 р., №70с**

Вихідні дані для роботи: використання технологій, структур, засобів та методів
для розробки веб-додатку, методи та засоби створення веб-додатку з
використанням стеку MERN.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів,
що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Єфремов Ю.М.	16.02.2023р.	16.02.2023р.
2	Єфремов Ю.М.	16.02.2023р.	16.02.2023р.
3	Єфремов Ю.М.	16.02.2023р.	16.02.2023р.

Календарний план

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі. Пошук, огляд та аналіз аналогічних розробок. Формулювання технічного завдання.	5 січня 2023 – 20 січня 2023	
2	Проектування структури системи	15 січня 2023 – 23 січня 2023	
3	Написання програмного коду	23 січня 2023 – 15 квітня 2023	
4	Тестування	10 квітня 2023 – 25 квітня 2023	
5	Аналіз результатів	26 квітня 2023	
6	Попередній захист	26 квітня 2023	

РЕФЕРАТ

Кваліфікаційна робота складається з веб-додатку магазину вінілових платівок «Musical Corner» працівників, продуктів та жанрів та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи містить 62 сторінки, 50 ілюстрацій та 7 таблиць.

Метою роботи є розробка веб-додатку, за допомогою якого можна купувати вінілові платівки, читати останні новини музики та ділитися своїми враженнями про них.

В роботі визначено основні завдання на розробку системи, проаналізовано аналоги розробленої системи. Наведено сценарії роботи веб-додатку, його загальну структуру, опис алгоритмів взаємодії окремих модулів системи, опис об'єктної структури веб-додатку. Також наводиться результати тестування додатку.

КЛЮЧОВІ СЛОВА: ВЕБ-ДОДАТОК, ПЛАТІВКИ, NODE, REACT, MERN.

					ІПЗ.КР.Б – 121 – 23 – ПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	Веб-додаток магазину вінілових платівок «Musical Corner» працівники, продукти та жанри Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.		Я.Д. Демченко						
Керівник		Ю.М. Єфремов					4	84
Рецензент		Н.М. Лобанчикова				Житомирська політехніка, група ІПЗ-19-2		
Зав. каф.		А.В.Морозов						

ABSTRACT

The qualification work consists of a web application of a vinyl record store «Musical Corner» of employees, products and genres and an explanatory note. Documentation contains 62 pages, 50 illustrations and 7 tables.

The purpose of the work is to develop a web application that allows you to buy vinyl records, read the latest music news, and share your impressions about them.

The paper defines the main tasks for the development of the system, analyzes the analogues of the developed system. The paper presents the scenarios of the web application, its general structure, a description of the algorithms of interaction between individual modules of the system, and a description of the object structure of the web application. The results of testing the application are also presented.

KEYWORDS: WEB APPLICATION, RECORDS, NODE, REACT, MERN.

					ИПЗ.КР.Б – 121 – 23 – ИПЗ	
						5

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	8
ВСТУП.....	9
РОЗДІЛ 1. АНАЛІЗ ЗАДАЧІ ТА ОСОБЛИВОСТЕЙ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ВЕБ-ДОДАТКУ З ПРОДАЖУ ВІНІЛОВИХ ПЛАТІВОК.....	10
1.1. Постановка задачі	10
1.2. Аналіз аналогів програмного продукту.....	12
1.3. Вибір архітектури веб-додатку «MUSICAL CORNER»	16
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.....	18
Висновки до розділу 1.....	19
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ	20
2.1. Визначення варіантів використання та структури системи.....	20
2.2. Опис структури веб-додатку	23
2.3. Проєктування бази даних веб-додатку	26
2.4. Реалізація веб-додатку магазину вінілових платівок «MUSICAL CORNER» працівників, продуктів та жанрів	28
Висновки до розділу 2.....	44
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ДОДАТКОМ	45
3.1. Порядок встановлення та налаштування системи.	45
3.2. Структура інтерфейсу. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ- ДОДАТКОМ.....	45
3.3. Тестування роботи веб-додатку	58
Висновки до розділу 3.....	59
ВИСНОВКИ	60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	61

					ІПЗ.КР.Б – 121 – 23 – ПЗ	
						6

ДОДАТКИ.....	63
ДОДАТОК А.....	64
ДОДАТОК Б.....	69
ДОДАТОК В.....	79

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CRUD – Create, Read, Update, Delete (створення, читання, оновлення, видалення).

JSON – JavaScript Object Notation (формат збереження даних)

MERN – Mongo Express React Node (Це JavaScript-стек, що складається з 4 компонентів із відкритим кодом, які забезпечують комплексне середовище розробки)

ODM – Object Document Mapping (технологія, яка дозволяє взаємодіяти з нереляційними базами даних)

REST – Representational State Transfer (підхід до архітектури мережеских протоколів, які надають доступ до інформаційних ресурсів)

API - Application Programming Interface (Прикладний програмний інтерфейс)

БД – База даних

MUI – Material UI (це бібліотека React UI компонентів, яка реалізує Material Design від Google)

SPA – Single Page Application (патерн для створення веб-додатків)

ВСТУП

Протягом останніх декількох років було помітно відродження популярності вінілових платівок. Багато людей різного віку знову зацікавилися цим форматом музики. Більше компаній почали випускати платівки з саундтреками з власних фільмів, серіалів та відеоігор. Створення інтернет-магазину є актуальним оскільки клієнти зможуть замовляти вподобані платівки витрачаючи набагато менше часових ресурсів ніж при відвідуванні офлайн магазину.

Метою даної роботи є розробка веб-додатку, за допомогою якого можна буде переглядати вінілові платівки та купувати їх, а також розробка системи адміністрування даного веб-додатку, а саме частина платівок, жанрів та працівників.

Описана мета вимагає необхідність виконання наступних завдань:

- аналіз теоретичних та функціональних потреб додатку;
- аналіз конкурентів, визначення унікальності додатку;
- визначення структури та архітектури додатку;
- визначення засобів реалізації додатку;
- проектування та реалізація додатку.

Об'єктом дослідження є методи та засоби створення веб-додатку з використанням стеку MERN.

Предметом дослідження є використання технологій, структур, засобів та методів для розробки веб-додатку.

Розроблений веб-додаток можна буде використати як практичний сайт, для продажу вінілових платівок, з можливою подальшою розробкою додаткового функціоналу.

Кваліфікаційна робота складається з вступу, 3-х розділів, висновків, списку використаних джерел з 17 найменувань, 3 додатків, 7 таблиць та 50 ілюстрацій по тексту. Повний обсяг роботи становить 84 сторінок, з них 62 сторінок основного тексту

РОЗДІЛ 1. АНАЛІЗ ЗАДАЧІ ТА ОСОБЛИВОСТЕЙ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ВЕБ-ДОДАТКУ З ПРОДАЖУ ВІНІЛОВИХ ПЛАТІВОК

1.1. Постановка задачі

Основною метою провадження веб-додатку магазину вінілових платівок «Musical Corner» є: перегляд та покупка музикальних платівок. Впроваджений додаток буде корисний як і любителям прослуховування музики і колекціонерам різних платівок, так і людям, які тільки зараз намагаються включитися в цей світ.

У кваліфікаційній роботі необхідно розробити веб-додаток за допомогою якого можна буде переглядати та купувати платівки, який буде працювати як на комп'ютері так і на телефоні або планшеті.

Основними етапами реалізації цього завдання є:

- створення серверної частини додатку:
 - створення ядра серверу;
 - проєктування бази даних;
 - створення з'єднання з базою даних;
 - створення необхідних моделей бази даних з відповідними полями:
 - модель працівників;
 - модель жанрів;
 - модель платівок;
 - модель відгуків до платівок.
 - Створення необхідних контролерів для взаємодії з базою даних:
 - CRUD та фільтрація платівок;
 - CRUD коментарів до платівок;
 - CRUD та фільтрація жанрів;
 - CRUD, фільтрація та логін працівників;
 - Контролери для отримання статистики.
 - Створення необхідних шляхів для взаємодії з сервером:
 - шляхи для платівок;

- шляхи для коментарів до платівок;
- шляхи для жанрів;
- шляхи для працівників.
- Тестування серверної частини додатку.
- Розробка дизайну адміністративної частини додатку:
 - створення загальної концепції дизайну;
 - верстання елементів інтерфейсу адміністративної частини на основі

React Material UI:

- головна сторінка (статистика);
- сторінка входу;
- відображення всіх платівок, жанрів та працівників;
- відображення однієї платівки, жанру, працівника;
- форма для додавання і редагування платівок, жанрів та працівника.
- кабінет працівника;
- елементи фільтрації та сортування.
- Розробка адміністративної частини веб-додатку:
 - створення ядра адміністративної частини додатку;
 - створення API для взаємодії з серверною частиною додатку;
 - створення необхідних дій, для взаємодії додатку з API;
 - створення маршрутизації додатку;
 - створення необхідних компонентів та сторінок додатку;
 - підключення сторонніх бібліотек для покращення роботи додатку:
 - створення віджету Clouddinary для збереження зображень в хмарі;
 - створення графіків Chart.js для більш наочного відображення статистики;
 - підключення Email.js для відправлення повідомлень на електронну пошту.
 - Тестування адміністративної частини веб-додатку.
- Розробка дизайну клієнтської частини додатку:
 - створення загальної концепції дизайну;

○ верстання елементів інтерфейсу клієнтської частини на основі React Material UI:

- відображення платівок;
 - відображення відгуків до платівки;
 - відображення кошика;
 - елементи фільтрації та сортування.
- Розробка клієнтської частини веб-додатку:
 - створення ядра клієнтської частини додатку;
 - створення API для взаємодії з серверною частиною додатку;
 - створення необхідних дій, для взаємодії додатку з API;
 - створення маршрутизації додатку;
 - створення необхідних компонентів та сторінок додатку;
 - підключення сторонніх бібліотек для покращення роботи додатку:
 - створення віджету Spotify для прослуховування музики.
 - Тестування клієнтської частини веб-додатку.

Результатом реалізації поставленого завдання є частина серверу, частина адміністративної та серверної додатку, яким зможуть користуватися на різних девайсах. Працівник зможе увійти в адміністративну частину додатку, переглядати статистику, редагувати свій профіль, переглядати та редагувати інформацію про платівки та жанри, також якщо працівник матиме права адміністратора, то додатково зможе переглядати та редагувати інформацію про всіх працівників. Клієнт зможе переглядати платівки, залишати відгуки до них, додавати платівки в кошик. Також буде присутній пошук та фільтрація в обох частинах додатку.

1.2. Аналіз аналогів програмного продукту

Протягом останніх років популярність вінілових платівок сильно зросла. Хоча інтернет-магазин Rozetka досі є лідером по продажам, існує і багато інших

магазинів, які продають платівки. Варто зазначити, що дані додатки не мають настільки різноманітний функціонал, як в цілому нашому додатку. Це більше вмотивовує для створення даного продукту.

Розглянемо подібні, вже існуючі рішення.

Rozetka – найбільший інтернет магазин в Україні. В даному магазині можна придбати велике різноманіття товарів, серед них і платівки. Серед переваг можна зазначити одну з найкращих для використання, та одну з найскладніших систем фільтрації, яка регенерується після кожного обраного фільтру. Також має чудову систему відгуків про товар, можна завантажувати фото та відео. З недоліків, великої кількості платівок немає в наявності, більшість платівок, які продаються – платівки від сторонніх магазинів, до багатьох з цих платівок немає жодної інформації, деякі з платівок не нові, немає саундтреків до фільмів, серіалів та ігор. Не можна прослуховувати треки з платівок онлайн. На рис. 1.1 показано інтернет магазин Rozetka.

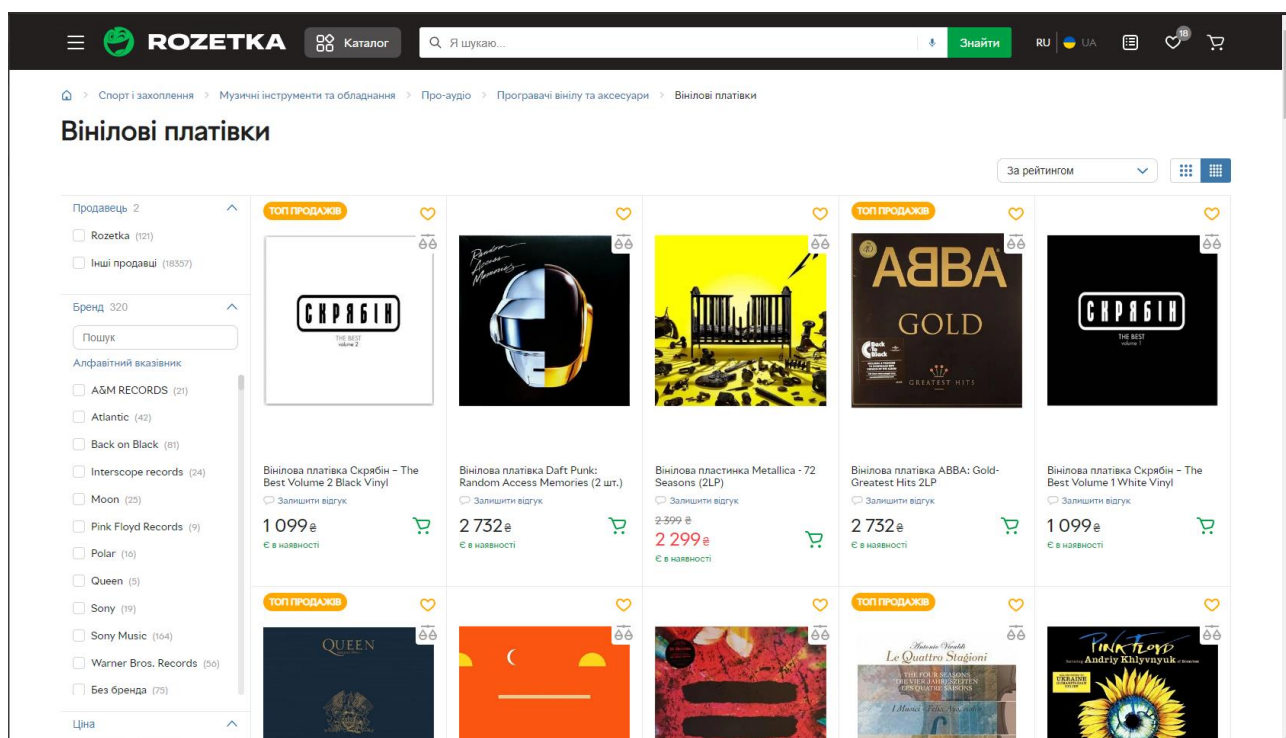


Рис. 1.1 – інтернет-магазин Rozetka

Vinyla – інтернет-магазин вінілу. Серед переваг можна зазначити, що в магазині продаються не тільки платівки, а й різне музичне обладнання та аксесуари, можна прослуховувати музику на сайті, присутні саундтреки з

фільмів та серіалів, та загалом велике різноманіття платівок різних жанрів. З мінусів можна зазначити неочевидний інтерфейс, частина фільтрації знаходиться в лівій частині екрану, частина посередині, пошук в іншому кутку, відсутність відгуків, велика кількість платівок або на стадії передзамовлення або їх немає в наявності. На рис. 1.2 показано інтернет магазин Vinyla.

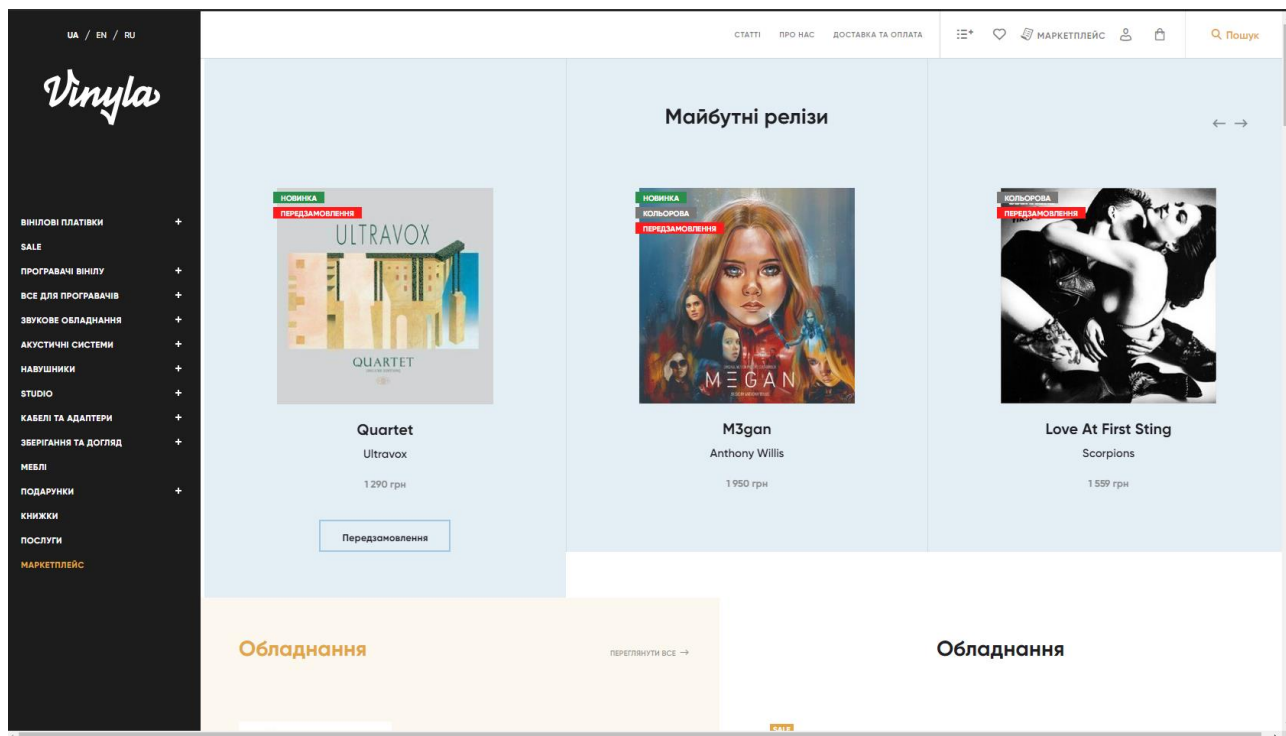


Рис. 1.2 – інтернет-магазин Vinyla

Playvinyl – інтернет магазин вінілу, має власний шоурум. Серед переваг можна зазначити приємний дизайн, в асортименті не тільки платівки а й обладнання та аксесуари. Можна дізнатися інформацію про виконавців. Велике різноманіття платівок, є платівки з саундтреками. З мінусів можна відзначити неточності в дизайні, відсутність власного кабінету, багато платівок немає в наявності, неповні фільтри. На рис. 1.3 показано інтернет магазин PlayVinyl.

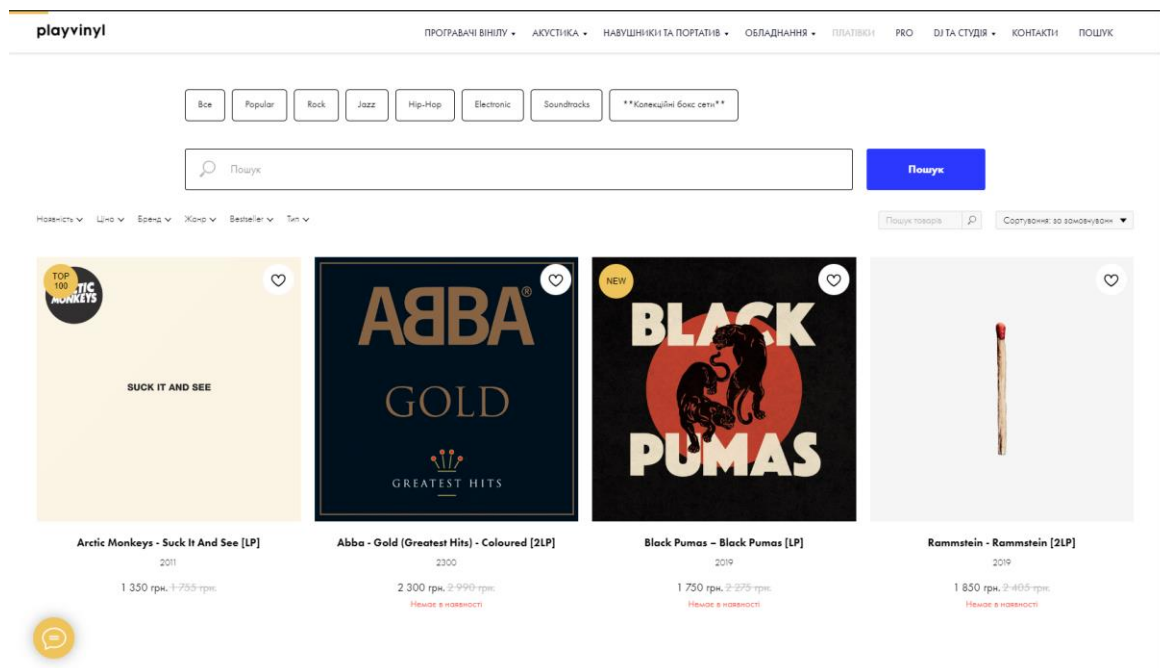


Рис. 1.3 – інтернет-магазин PlayVinyl

Отже порівняємо дані веб-додатки.

Таблиця 1.1

	Кабінет	Фільтрація	Саундтреки	Наявність обладнання та аксесуарів	Медіа на сторінці платівкою з
Rozetka	Так	Найкраща	Відсутні	Присутні	Відсутня
Vinyla	Так	Неочевидна	Присутні	Присутні	Присутня
PlayVinyl	Ні	Неповні	Присутні	Присутні	Присутня

Проаналізувавши аналоги створюваного веб-додатку, можна стверджувати, що в кожного додатку є свої переваги та недоліки, що дозволяє користувачу обирати, де здійснювати покупки.

Враховавши дану інформацію виникає потреба в розробці веб-додатку з більш вузьким призначенням, що дозволить сфокусуватися на головних особливостях продукту.

Отже головними рисами даного продукту будуть: простий дизайн, фільтрація, присутність саундтреків, присутність медіа на сторінці перегляду платівки.

1.3. Вибір архітектури веб-додатку «Musical Corner»

Для розробки веб-додатку можливо використати такі архітектурні шаблони програмного забезпечення:

- MVC;
- шаблон клієнт-сервер;
- шаблон слоїв;
- мікросервіси;
- сервісно-орієнтована архітектура.

Для інтернет магазину підходить декілька шаблонів, таких як MVC або клієнт-сервер, але оптимальним для реалізації додатку буде використання шаблону клієнт-сервер. На рис. 1.4 показано схему шаблону клієнт-сервер.

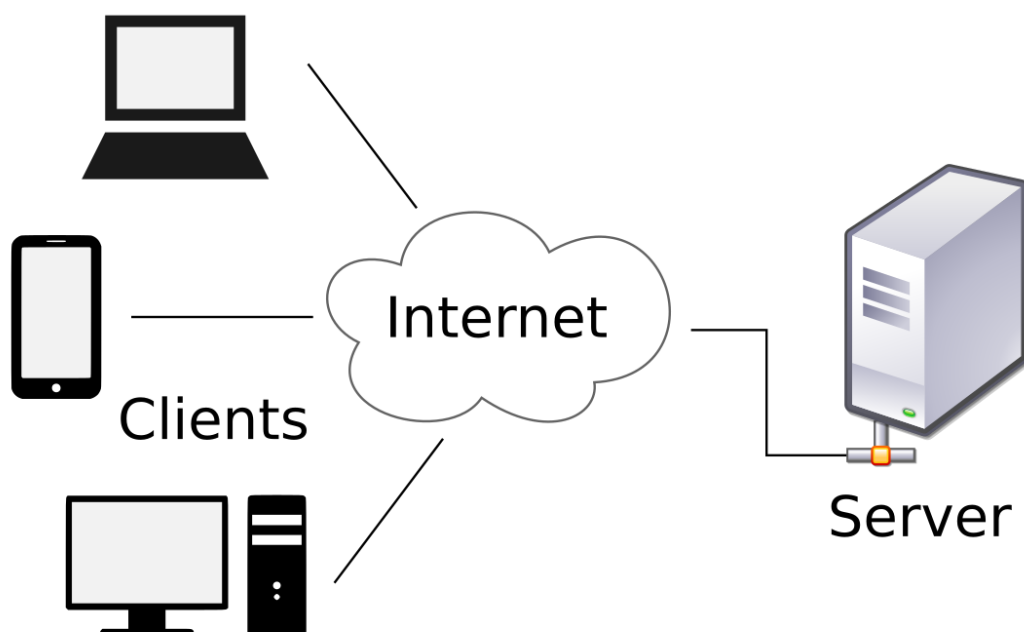


Рис. 1.4. Схема шаблону клієнт-сервер

Шаблон клієнт-сервер є одним з найпоширеніших архітектурних шаблонів у розробці програмного забезпечення. Він використовується для побудови систем, де взаємодія між користувачем і сервером є центральною [4].

У клієнт-серверній архітектурі система складається з двох основних компонентів:

– Клієнт: це програмна частина, яка взаємодіє з користувачем. Клієнтська частина може бути веб-браузером, мобільним додатком або настільним додатком. Клієнтська частина відповідає за відображення інтерфейсу користувача та збору введених даних. Вона надсилає запити до сервера і отримує відповіді для відображення результатів користувачу.

– Сервер: Це центральна частина системи, яка обробляє запити від клієнта і надає відповіді. Сервер зберігає бізнес-логіку, доступ до бази даних, обробку запитів та відповідей. Він може бути фізичним сервером або хмарним сервісом.

Шаблон клієнт-сервер є чудовим варіантом для інтернет-магазину з таких причин:

– Розділення обов'язків: клієнт-серверний шаблон дозволяє чітко розділити функціональні обов'язки між клієнтською і серверною частинами системи. Клієнт відповідає за відображення інтерфейсу користувача, взаємодію з користувачем та передачу запитів серверу. Сервер відповідає за обробку запитів, бізнес-логіку, збереження даних та надання відповідей клієнту. Це спрощує розробку, підтримку та масштабування системи.

– Забезпечення безпеки: клієнт-серверний шаблон дає можливість зосередитися на захисті серверної частини, яка зберігає конфіденційні дані та здійснює важливі обчислення. Це дозволяє використовувати різні механізми захисту, такі як шифрування даних, аутентифікація та авторизація користувачів, засоби захисту від атак та аналогічні.

– Розширюваність: завдяки розділенню на клієнтську і серверну частини, можна легко розширювати функціональність системи, додаючи нові функції або модулі. Це зменшує залежність між компонентами та спрощує розробку та підтримку.

Отже, можна зробити висновок, що даний архітектурний шаблон чудово підходить для побудови обраного додатку. Даний підхід забезпечить реалізацію проєкту, а також його подальшу модифікацію.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

При створенні веб-додатку потрібно обрати інструменти, які максимально спростять та пришвидшать розробку. Описана вимога зменшує кількість варіантів для обрання засобів розробки.

Оскільки на серверній частині додатку відбувається обробка запитів, бізнес-логік, збереження даних та надання відповідей клієнту, отже потрібно розумно обрати мову для написання серверу.

Для написання серверу було обрано фреймворк Express для Node.js, це один з найпопулярніших засобів для написання серверної частини додатку. До його переваг можна віднести:

- Простота у використанні: Express є доволі простим та інтуїтивно зрозумілим для розробки веб-додатків. Також через його мінімалістичну структуру можна доволі швидко створювати маршрути, обробники запитів та відповідей.

- Швидкодія: express є досить швидким фреймворком, оскільки він базується на платформі Node.js, яка використовує неблокуючий ввід/вивід (non-blocking I/O). Це дозволяє обробляти багато запитів паралельно та ефективно використовувати ресурси сервера.

- Інтеграція з різними засобами: express легко інтегрується з різними засобами для роботи з базами даних, що дає велику гнучкість при виборі технологій для проєкту.

При виборі бази даних для інтернет-магазину слід звертати увагу на такі фактори як типи даних, обсяг даних, швидкодія а також навички та досвід розробників у роботі з даною базою даних, майбутні потреби.

Для фреймворку Express для Node.js найкраще підходять такі бази даних як MongoDB, PostgreSQL, MySQL, SQLite та FirebaseDB.

Оскільки в майбутньому інтернет-магазин буде покращуватись для

продажу не тільки вінілових платівок а й аксесуарів, таких як програвачі або полицьки платівок, отже кращим вибором будуть нереляційні бази даних, оскільки в одній колекції можна буде мати документи з різними полями.

Для розробки інтернет-магазину краще використовувати базу даних MongoDB ніж FireBaseDB, оскільки MongoDB має кращу продуктивність, кращий обсяг зберігання даних, кращі запити для взаємодії з базою даних [5], а FireBaseDB краще підходить для невеликих додатків, або онлайн-ігор.

Отже під час розробки буде використовуватися база даних MongoDB.

Для фронтенд частини додатку порівняємо найпопулярніші бібліотеки та фреймворки, такі, як React.js, Vue.js, Angular.

З переваг React.js можна перелічити компонентний підхід, який дозволяє легко організувати код та перевикористовувати його, другою перевагою є його гнучкість, яка дає свободу вибору інших бібліотек [6].

До переваг Angular можна включити повноцінність цього фреймворку, він має вбудовані функції для маршрутизації, управління станами та тестування. Також перевагою можна назвати строгість даного фреймворку.

До переваг Vue.js можна включити його легкість, швидкодію а також гнучкість, можна обирати як використовувати Vue.js, як фреймворк чи бібліотеку.

Враховуючи навички та досвід розробників, те, що команда розробників складається з двох людей та гнучкість React.js, для написання клієнтської частини додатку було обрано React.js

Висновки до розділу 1

Після проведеного аналізу вже існуючих додатків було визначено їх переваги та недоліки, було виділено особливості, які мають бути присутні в майбутньому веб-додатку. У результаті аналізу засобів розробки було вирішено використати фреймворк Express.js, базу даних MongoDB та бібліотеку React.js для клієнтської частини.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ

2.1. Визначення варіантів використання та структури системи.

Робота є комплексною, кожен учасник команди виконував поставлені завдання для роботи. Об'єкти відповідальності було обрано з підстав особистих побажань. Розподіл роботи можна переглянути в таблиці 2.1.

Таблиця 2.1

Розподіл ролей

Учасник	Об'єкт відповідальності
Демченко Ярослав Дмитрович	Платівки, жанри та працівники
Лецишина Вікторія Олегівна	Новини, замовлення та користувачі

Веб-додаток магазину вінілових платівок «Musical Corner» створюється з метою перегляду, прослуховування і придбання вінілових платівок.

Вимоги користувачів для працівників, платівок і жанрів.

Усі користувачі клієнтської частини мають доступ до наступних функцій:

- перегляд всіх платівок;
- фільтрація, сортування та пошук платівок;
- додавання платівок в кошик;
- перегляд повної інформації про платівку;
- перегляд інформації про жанри;
- перегляд відгуків від інших користувачів;
- редагування власного кошику.

Авторизовані користувачі клієнтської частини мають доступ до додаткових функцій:

- написання відгуків;
- редагування та видалення власних відгуків.

Усі користувачі адміністративної частини (працівники) мають доступ до наступних функцій:

- перегляд та оновлення власного кабінету;

- створення нової платівки;
- фільтрація, сортування та пошук платівок;
- перегляд всіх платівок;
- перегляд повної інформації про платівку та її оновлення;
- налаштування видимості платівок для клієнтської частини;
- створення нового жанру;
- фільтрація, сортування та пошук жанрів;
- перегляд повної інформації про жанр та його оновлення;
- перегляд статистики.

Користувачі адміністративної частини (працівники) з правами адміністратора мають доступ до наступних функцій:

- створення нового працівника;
- фільтрація, сортування та пошук всіх працівників;
- перегляд всіх працівників;
- перегляд повної інформації про працівника, його оновлення та видалення.

Функціональні вимоги:

- адміністративна та клієнтська частина повинні мати можливості фільтрації, сортування та пошуку;
- відтворення частини музикальних доріжок (якщо доступно);
- адміністративна частина має містити автентифікацію;
- адміністративна частина має містити статистику;
- клієнтська частина повинна містити кошик.

Нефункціональні вимоги:

- інтерфейси клієнтської та адміністративної частини додатку мають бути зручними для використання для клієнта та для адміністратора відповідно;
- додаток має працювати в сучасних браузерях;
- легкість до масштабування.

Аналіз вимог користувачів та функціональних вимог дозволив виділити сутності, що забезпечать реалізацію програмного комплексу системи. Розглянемо яким чином звичайний працівник та адміністратор може взаємодіяти з адміністративною частиною веб-додатку (рис. 2.1).

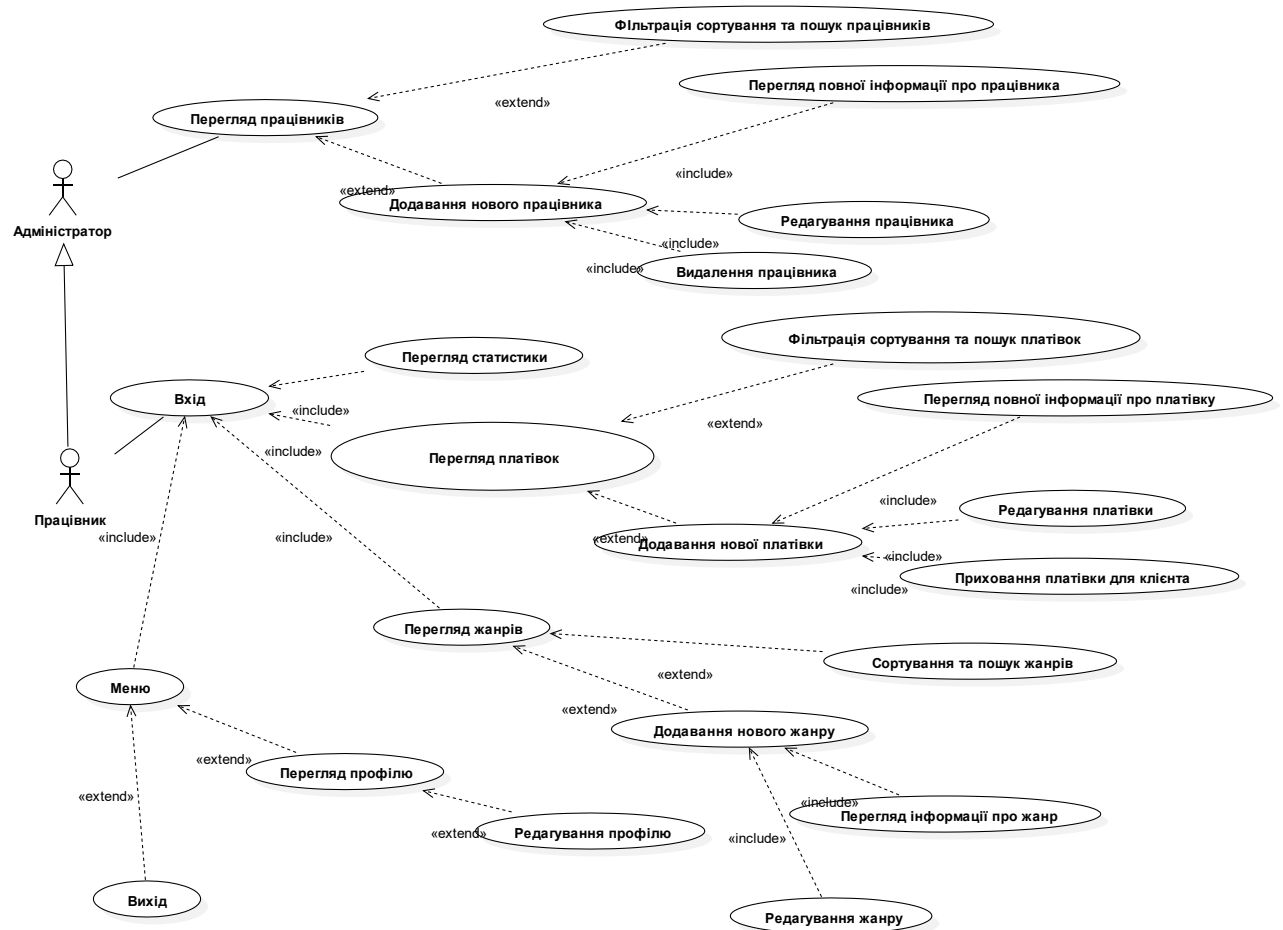


Рис.2.1. Діаграма варіантів використання адміністративної частини

Для доступу до адміністративної частини додатку будь-якому працівнику потрібно авторизуватись. Після авторизації працівник потрапляє на головну сторінку, де можна побачити статистику інтернет-магазину. Натиснувши на кнопку в правому, верхньому кутку екрану можа буде побачити меню, за допомогою якого можна буде вийти з профілю або перейти до особистого кабінету. В лівій частині екрану можна буде побачити кнопки, за допомогою яких можна буде перейти до перегляду всіх платівок, та жанрів. На кожній з даних сторінок працівник зможе переглядати інформацію, переглянути повну інформацію про платівку або жанр, перейти до створення нової платівки або жанру або переїти до їх редагування.

Якщо працівник матиме роль адміністратора, то для нього будуть відкриті додаткові параметри, такі як перегляд працівників, їх створення, редагування та видалення.

Розглянемо яким чином користувач та зареєстрований користувач зможе взаємодіяти з клієнтською частиною веб-додатку (рис. 2.2).

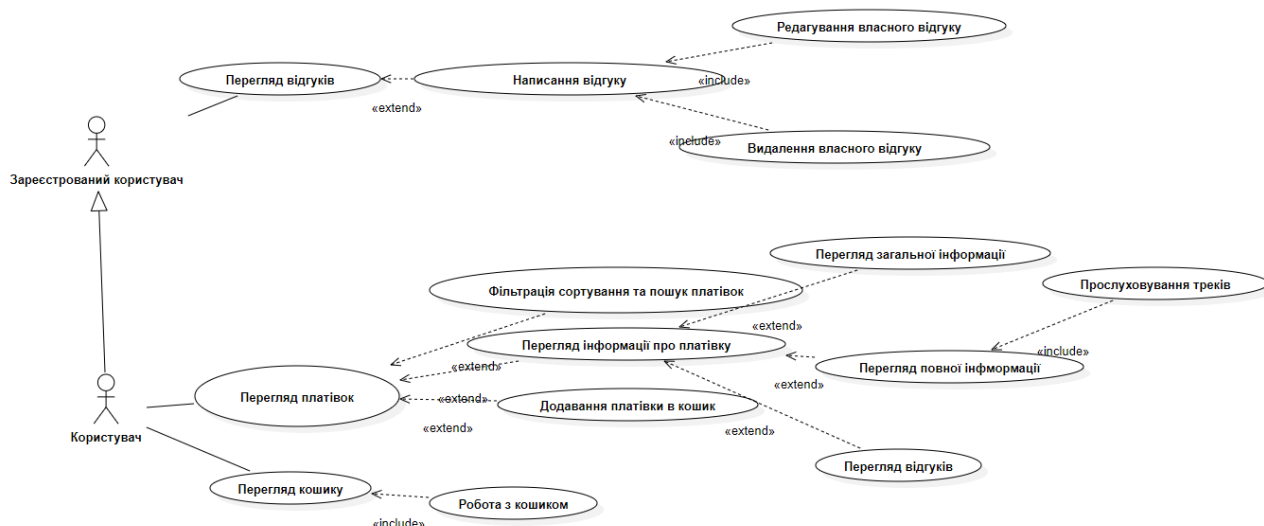


Рис.2.2. Діаграма варіантів використання клієнтської частини.

Звичайному користувачу не потрібно реєструватися для взаємодії з додатком, він може переглядати та фільтрувати платівки, переглядати відгуки до них, додавати їх до кошику, редагувати кошик. Але зареєстрований користувач може додавати свої відгуки та редагувати або видаляти їх.

2.2. Опис структури веб-додатку

Серверна частина буде використовувати модульний підхід організації коду. При використанні даного підходу сервер ділиться на окремі модулі, які далі підключаються в ядрі сервера на зразок бібліотек [7]. В файловій структурі проєкту будуть використані моделі, які використовуються для створення колекцій в базі даних, контролери – функції для роботи з базою даних, middleware – функції, які обробляють запити та відповіді в проміжку між отриманням запиту та відправленням відповіді сервером. Вони дозволяють виконувати додаткові операції на рівні сервера перед передачею запиту до наступного обробника або перед відправленням відповіді клієнту. Шляхи є

останнім типом модулів, за допомогою них визначається коли будуть виконуватися контролери. Файлову структуру проєкту можна побачити на рис. 2.3.

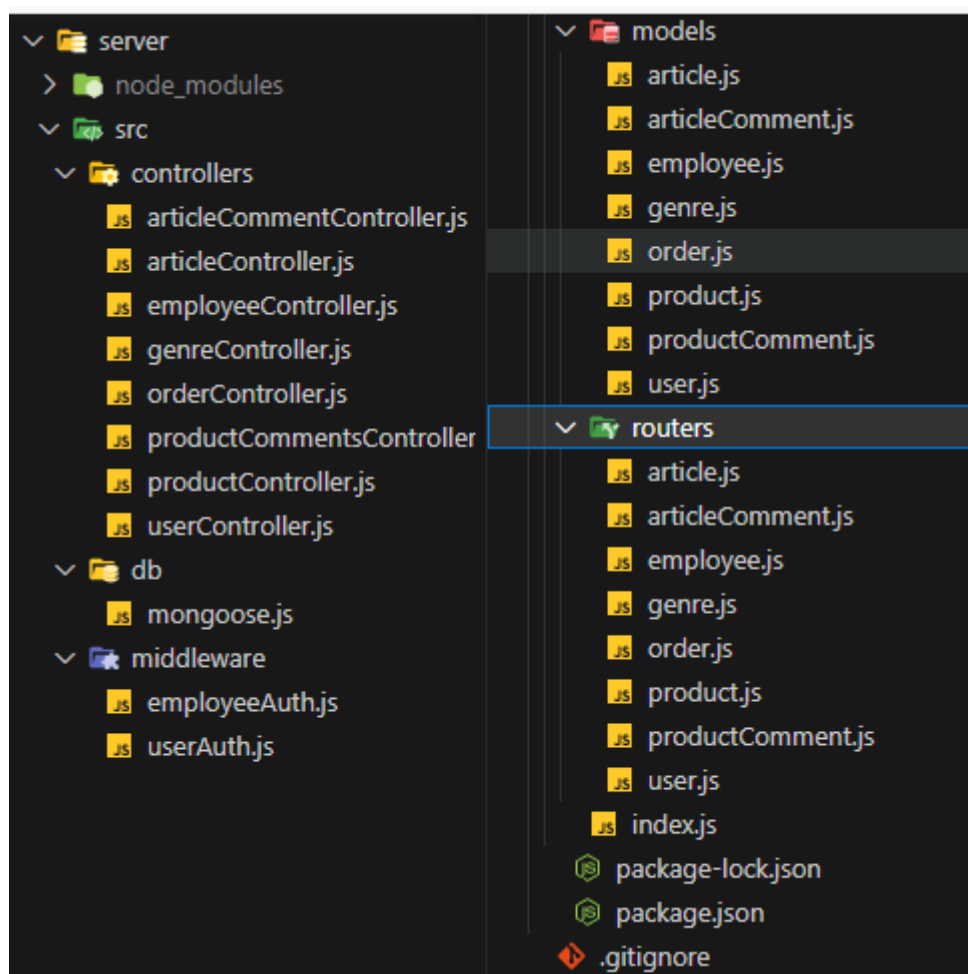


Рис. 2.3. Файлова структура серверної частини додатку

Файл `mongoose.js` слугує для підключення до бази даних, файл `index.js` в кореневому каталозі являється ядром серверу. Файли `package.json` та `package-lock.json` містять інформацію про даний додаток та пакети, які використані в ньому [8].

Клієнтська та адміністративні частини додатку буде розроблятися за допомогою `React.js`. Дані частини мають однакову файлову структуру, тому структуру веб-додатку на прикладі адміністративної частини можна побачити на рис. 2.4

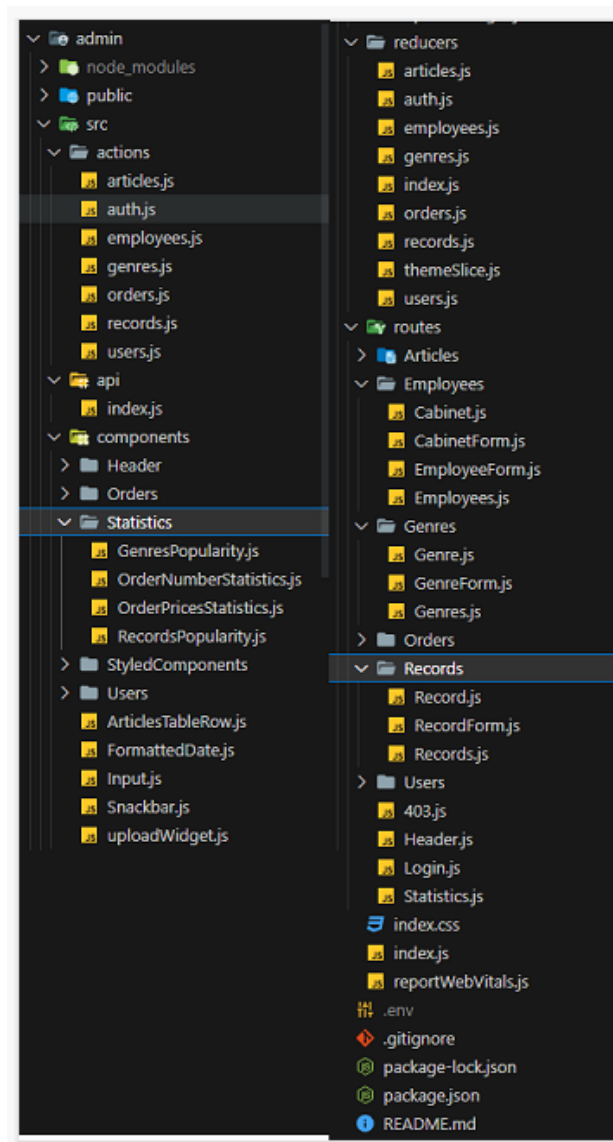


Рис. 2.4. Файлова структура адміністративної частини додатку

В папці `public` знаходиться оболочка проекту. В папці `src` знаходяться папки `actions`, в якій знаходяться функції для роботи з даними, які отримуються з серверної частини додатку, в папці `api` файл `index.js` містить шляхи, за допомогою яких адміністративна частина взаємодіє з серверною, папка `components` містить компоненти, компоненти є основним блоком для розробки інтерфейсу, вони дозволяють розбити інтерфейс на незалежні, повторно використовувані частини. Папка `reducers` містить функції для взаємодії з діями з папки `actions`. Папка `routes` містить всі сторінки, за допомогою яких буде будуватися інтерфейс веб-додатку. Файл `index.js` можна назвати ядром адміністративної частини додатку, `.env` файл – це файл конфігурації, який

використовується для зберігання конфіденційних даних додатку, такі як API-ключі, паролі та інші параметри.

2.3. Проектування бази даних веб-додатку

Спроектована частина бази даних містить такі колекції, як employee, в яких зберігаються дані про працівників, products, в якій зберігаються дані про самі продукти, genres, в якій зберігаються дані про всі наявні жанри та productcomments, в якій зберігаються відгуки до продуктів.

Колекція employees містить такі поля:

- fullName – об'єкт, який містить інформацію про повне ім'я працівника:
 - firstName – текстове поле, яке містить інформацію про ім'я працівника;
 - lastName – текстове поле, яке містить інформацію про прізвище працівника;
 - patronymic – текстове поле, яке містить інформацію про по-батькові працівника.
- email – текстове поле, яке містить інформацію про електронну пошту працівника;
- password – текстове поле, яке містить інформацію про пароль працівника, поле хешується;
- phoneNumber – текстове поле, яке містить інформацію про номер телефону працівника;
- isAdmin – поле типу boolean, яке містить інформацію про рівень доступу працівника;
- isWorking – об'єкт, який містить інформацію про те, чи працює працівник на даний момент, та причину, через яку працівник не працює на даний момент:
 - isWorking – поле типу boolean, яке містить інформацію про те, чи працює працівник зараз;
 - Reason – текстове поле, яке містить інформацію про причину, через яку працівник не працює на даний момент;
- address – об'єкт, який містить інформацію про адресу проживання

працівника:

- country – текстове поле, яке містить інформацію про країну проживання працівника;
- city – текстове поле, яке містить інформацію про місто проживання працівника;
- street – текстове поле, яке містить інформацію про вулицю, на якій проживає працівник;
- zipCode – текстове поле, яке містить інформацію про поштовий індекс працівника.

- tokens – масив об'єктів токенів, слугує для авторизації;
- системи симуляції.

Колекція genres містить такі поля:

- name – текстове поле, яке містить інформацію про назву жанру;
- description – текстове поле, яке містить інформацію про опис жанру;

Колекція products містить такі поля:

- name – текстове поле, яке містить інформацію про назву жанру;
- author – текстове поле, яке містить інформацію про автора платівки;
- description – текстове поле, яке містить детальний опис платівки;
- recordLength – текстове поле, яке містить інформацію про час програвання платівки;
- label – текстове поле, яке містить інформацію про лейбл, на якому була видана платівка;
- price – числове поле, яке містить інформацію про ціну продукту;
- images – масив текстових полів, в яких містяться посилання на зображення продукту;
- songList – масив текстових полів, в яких містяться назви треків платівки;
- releaseDate – поле дати, яка містить інформацію про дату виходу альбому;

- productionDate – поле дати, яка містить інформацію про дату виходу платівки;
- itemsAvailable – поле цілих чисел, в якому міститься інформація про кількість платівок на складі;
- genres – масив id, в якому містяться id жанрів даної платівки;
- country – текстове поле, яке містить інформацію про країну, з якої походить автор платівки;
- isColored – поле типу boolean, яке містить інформацію про те, чи є платівка кольоровою;
- recordsNumber – поле цілих чисел, яке містить інформацію про кількість платівок в комплекті;
- isHidden – поле типу boolean, яке дозволяє не показувати платівку клієнтам.

Колекція productcomment містить такі поля:

- userId – поле типу ObjectId, в якому міститься id користувача, який написав відгук;
- productid – поле типу ObjectId, в якому міститься id продукту, до якого написаний відгук;
- text – текстове поле, яке містить текст відгуку;
- creationDate – поле дати, яка містить інформацію про дату написання відгуку;
- editDate – поле дати, яка містить інформацію про дату редагування відгуку, якщо редагування відбулося.

2.4. Реалізація веб-додатку магазину вінілових платівок «Musical Corner» працівників, продуктів та жанрів

Серверна частина веб-додатку розробляється як RESTful API, це означає що додаток написаний з дотриманням принципів REST (Representational State Transfer) [9] та написаний на мові програмування JavaScript на платформі Node.js

за допомогою фреймворку Express.js. Сервер має ядро та додаткові модулі, які підключаються до нього, серед них моделі, контролери та шляхи. Оскільки клієнтська та адміністративна частини додатку будуть написати за допомогою мови програмування JavaScript, то для більшої зручності дані будуть передаватися у форматі JSON. Для взаємодії з базою даних буде використовуватись ODM Mongoose.js [10], який дозволяє працювати з даними бази даних у вигляді об'єктів.

Однією з головних колекцій бази даних є колекція products, яка містить ключові дані про продукт. На рис. 2.5 показано схему продукту, яка потім перетвориться в модель.

```
let productSchema = new mongoose.Schema({
  name: { type: String, required: [true, 'Поле "Назва" не може бути порожнім'], trim: true },
  author: { type: String, required: [true, 'Поле "Автор" не може бути порожнім'], trim: true },
  description: { type: String, required: [true, 'Поле "Опис" не може бути порожнім'], trim: true },
  recordLength: { type: String, required: [true, 'Поле "Час програвання платівки" не може бути порожнім'] },
  label: { type: String, required: [true, 'Поле "Лейбл" не може бути порожнім'], trim: true },
  price: { type: Number, required: [true, 'Поле "Ціна" не може бути порожнім'],
    validate(value) {
      if (value <= 0) {
        throw new Error("Ціна повинна бути більше 0")
      }
    }
  },
  images: { type: [String], default: ['https://grandmart.biz/Products/default.jpg'] },
  songlist: { type: [String],
    validate(value) {
      if (value.length < 1) {
        throw new Error("Платівка має містити хоча 1 один трек")
      }
    }
  },
  releaseDate: { type: Date, default: new Date().toISOString().split('T')[0] },
  productionDate: { type: Date, default: new Date().toISOString().split('T')[0] },
  itemsAvailable: { type: Int32, default: 1, trim: true },
  genres: { type: [mongoose.Schema.Types.ObjectId], ref: "Genre",
    validate(value) {
      if (value.length < 1) {
        throw new Error("Платівка має містити хоча 1 один жанр")
      }
    }
  },
  country: { type: String, required: [true, 'Поле "Країна" не може бути порожнім'], trim: true },
  isColored: { type: Boolean, default: false },
  recordsNumber: { type: Int32, default: 1, trim: true },
  isHidden: { type: Boolean, default: true }
});
```

Рис. 2.5. Схема продукту

Важливим middleware є функція employeeAuth, яка перевіряє чи користувач є автентифікованим, вона слугує для того, щоб до деяких функцій

мав доступ тільки автентифікований працівник. На рис. 2.6 показано функцію employeeAuth.

```
const employeeAuth = async (req, res, next) => {
  try {
    const token = req.header('Authorization').replace("Bearer ", "");
    const decoded = jwt.verify(token, 'kdweueksdsjkor');
    const employee = await Employee.findOne({_id: decoded._id, 'tokens.token': token});
    if (!employee) {
      throw new Error();
    }
    req.employee = employee;
    req.token = token;
    next();
  } catch (e) {
    res.status(401).send("Будь ласка, аутентифікуйтесь як працівник");
  }
};
```

Рис. 2.6. Функція EmployeeAuth

Важливими контролерами є контролери продуктів, оскільки це головна суть інтернет-магазину. Під час розробки контролерів потрібно звернути увагу на те, що вони повинні бути асинхронними. На рис. 2.7 показано контролер products_list, який відповідає за отримання всіх продуктів з бази даних.

```
// Display list of all Products.
exports.products_list = async (req, res) => {
  try {
    const records = await Product.find().sort({$natural: -1}).populate("genres");
    res.status(200).send({ records: records });
  } catch (error) {
    res.status(500).send(error.message);
  }
};
```

Рис. 2.7. Контролер products_list

Даний контролер отримує всі продукти, сортує їх за датою додавання а далі виконується оператор populate [11], який замінює всі id в масиві жанрів, документами з колекції жанрів. Далі даний масив продуктів повертається у відповідь, якщо це зробити не вдається, повертається повідомлення про помилку.

На рис. 2.8 показано контролер filtered_products_list, який відповідає за фільтрацію, сортування та пошук продуктів з бази даних.

```

exports.filtered_products_list = async (req, res) => {
  let filter = {},
      sorting = {};
  const { search, genres, availability, prices, sort, isColored, client } = req.query;
  if (
    (typeof search === "string" || search instanceof String) && search.length > 0
  ) {
    filter.$text = { $search: search };
  }
  if ((typeof client === "string" || client instanceof String) && client.length > 0 && client === "true")
    filter.isHidden = { $eq: false };
  if (genres.length > 0 && genres !== "none")
    filter.genres = { $in: genres.split(",") };
  if ((typeof availability === "string" || availability instanceof String) && availability.length > 0 && availability !== "none") {
    if (availability === "true") filter.itemsAvailable = { $gt: 0 };
    else if (availability === "false") filter.itemsAvailable = { $eq: 0 };
  }
  if ((typeof isColored === "string" || isColored instanceof String) && isColored.length > 0 && isColored !== "none") {
    if (isColored === "true") filter.isColored = { $eq: true };
    else if (isColored === "false") filter.isColored = { $eq: false };
  }
  if (prices.length > 0 && prices !== "none")
    filter.price = { $gte: prices.split(",")[0], $lte: prices.split(",")[1] };
  if ((typeof sort === "string" || sort instanceof String) && sort.length > 0 && sort !== "none") {
    if (Number(sort[sort.length - 1]) === 1)
      sorting[sort.slice(0, sort.length - 1)] = 1;
    else sorting[sort.slice(0, sort.length - 1)] = -1;
  }
  try {
    const records = await Product.find(filter)
      .collation({ locale: "en" })
      .sort(sorting)
      .populate("genres");
    res.status(200).send({ records: records });
  } catch (error) {
    res.status(500).send(error.message);
  }
};

```

Рис. 2.8. Контролер products_list

Даний контролер отримує дані запиту, якщо дані існують, то до об'єкту filter додаються нові поля, по яким буде здійснюватися фільтрація, якщо в запиті було сортування, то в об'єкт sort записується спосіб сортування. Потім відбувається пошук продуктів бази даних з використанням вказаних фільтрів, сортування за вказаними полями, далі аналогічно виконується оператор populate, який замінює id жанрів на їх документи, далі повертається у відповідь з відфільтрованими продуктами, якщо це зробити не вдається повертається повідомлення про помилку.

На рис. 2.9 показано контролер product_create, який відповідає за фільтрацію, сортування та пошук продуктів з бази даних.

```

//Create new Product
exports.product_create = async (req, res) => {
  try {
    const record = new Product({
      ...req.body,
    });
    await record.save();
    res.status(201).send({ success: true, record: record });
  } catch (error) {
    let errM = "";
    for (const [key, value] of Object.entries(error.errors)) {
      if (Object.entries(error.errors).length !== 1) errM += value + ", ";
      else errM = value.properties.message;
    }
    return res.status(400).send(errM);
  }
};

```

Рис. 2.9. Контролер products_create

Даний контролер створює новий продукт з даними, які отримає з тіла запиту, далі зберігає даний продукт в БД і відправляє у відповідь на запит. При помилці відправляється комплексний парсинг помилок і додається в змінну errM, яка відправляється у відповідь.

Для розроблення URL-адрес був створений новий файл, в якому викликаються необхідні контролери при запиті на конкретну URL-адресу. На рис. 2.10 показано файл роутингу продуктів.

```

er > src > routers > product.js > ...
const express = require("express");
const productController = require('../controllers/productController');
const router = new express.Router();
const employeeAuth = require('../middleware/employeeAuth')

router.get("/products", employeeAuth, productController.products_list);
router.get("/products/main", productController.products_main);
router.get("/products-client", productController.products_list_client);
router.get("/products/search", productController.filtered_products_list);
router.post("/products/cart", productController.cart_products_list);
router.get("/products/:id", productController.product_detail);
router.post("/products/create", employeeAuth, productController.product_create);
router.patch("/products/update/:id", employeeAuth, productController.product_update);
router.patch("/products/hide/:id", employeeAuth, productController.product_hide);

module.exports = router;

```

Рис. 2.10. Файл роутингу продуктів

У деяких роутингах була вказана middleware функція employeeAuth, яка потрібна для надання доступу працівникам до тих функцій, які потребують автентифікації.

У табл. 2.2 представлено основні кінцеві точки для продуктів та дії, які вони виконують.

Таблиця 2.2

Кінцеві точки для продуктів

Кінцева точка	HTTP-метод	Потреба в автентифікації	Опис
/products	Get	Так	Повертає список всіх продуктів.
/products-client	Get	Ні	Повертає список всіх видимих продуктів.
/products/search	Get	Ні	Повертає список продуктів, які підпали під фільтрацію, пошук, сортування.
/products/cart	Post	Ні	Повертає список всіх продуктів кошика.
/products/:id	Get	Ні	Повертає інформацію про конкретний продукт.
/products/create	Post	Так	Повертає інформацію про створений продукт.
/products/update/:id	Patch	Так	Повертає інформацію про оновлений продукт.
/products/hide/:id	Patch	Так	Ховає продукт з відображення для користувачів.

Важливими контролерами для працівника є контролери які слугують для входу в профіль або виходу з нього. На рис. 2.11 показано контролери входу та виходу.

```

// Login for employee
exports.employee_login = async (req, res) => {
  try {
    const employee = await Employee.findOneByCredentials(
      req.body.email,
      req.body.password
    );
    const token = await employee.generateAuthToken();
    res.send({ employee, token });
  } catch (error) {
    res.status(400).send(error.message);
  }
};

// Logout for employee
exports.employee_logout = async (req, res) => {
  try {
    req.employee.tokens = req.employee.tokens.filter((token) => {
      return token.token !== req.token;
    });
    await req.employee.save();
    res.send("Вихід успішний");
  } catch (error) {
    res.status(500).send(error.message);
  }
};

```

Рис. 2.11. Контролери employee_login та employee_logout

При вході спочатку викликається власна функція `findOneByCredentials` яка шукає працівника по його електронній пошті та пароллю. Далі створюється автентифікаційний токен, знайдений працівник та токен відправляються у відповідь, якщо виникає помилка, у відповідь відправляється повідомлення про помилку.

При виході з власного профілю з об'єкта працівника видаляється токен якого не було в реквесті, далі працівник зберігається та відправляється повідомлення про успішність чи неуспішність виходу.

У всіх роутингах працівників, крім входу, була вказана `middleware` функція `employeeAuth`, яка потрібна для надання доступу працівникам до потрібних функцій.

У табл. 2.3 представлено основні кінцеві точки для працівників та дії, які вони виконують.

					ІПЗ.КР.Б – 121 – 23 – ПЗ	
						34

Кінцеві точки для працівників

Кінцева точка	HTTP-метод	Потреба в автентифікації	Опис
/employees	Get	Так	Повертає список всіх працівників.
/employees/search	Get	Так	Повертає список працівників, які підпали під фільтрацію, пошук, сортування.
/employees/create	Post	Так	Повертає інформацію про створений продукт.
/employees/login	Post	Ні	Слугує для входу.
/employees/cabinet	Get	Так	Повертає інформацію про власний профіль.
/employees/logout	Post	Так	Слугує для виходу.
/employees/:id	Get	Так	Повертає інформацію про конкретного працівника.
/employees/update/:id	Patch	Так	Повертає інформацію про оновленого працівника.
/employees/cabinet/update	Patch	Так	Повертає інформацію про оновлений кабінет.
/employees/delete/:id	Delete	так	Видаляє працівника.

У деяких роутингах жанрів, була вказана middleware функція employeeAuth, яка потрібна для надання доступу працівникам до потрібних функцій.

У табл. 2.4 представлено основні кінцеві точки для жанрів та дії, які вони виконують.

Кінцеві точки для жанрів

Кінцева точка	HTTP-метод	Потреба в автентифікації	Опис
/genres	Get	Ні	Повертає список всіх жанрів.
/genres/search	Get	Ні	Повертає список жанрів, які підпали під, пошук, сортування.
/genres/create	Post	Так	Повертає інформацію про створений жанр.
/genres/:id	Get	Ні	Повертає інформацію про конкретний жанр.
/genres/update/:id	Patch	Так	Повертає інформацію про оновлений жанр.

У моделі жанрів відсутній контролер видалення, оскільки розгляд видалення жанру як операції є неправильним через те, що це може призвести до проблем з іншими функціями.

Оскільки створення відгуку доступне лише автентифікованим користувачам, в роутах, що стосуються відгуків, було використано middleware-функцію `userAuth`, розроблену Лецишиною Вікторією. Ця функція призначена для перевірки автентифікації користувачів та надання їм доступу до відповідних функцій.

У табл. 2.5 представлено основні кінцеві точки для відгуків та дії, які вони виконують.

					ІПЗ.КР.Б – 121 – 23 – ІЗ	
						36

Кінцеві точки для відгуків

Кінцева точка	HTTP-метод	Потреба в автентифікації	Опис
/product-comments/create/:productId	Post	Так	Повертає створений відгук.
/product-comments/update/:id	Patch	Так	Повертає оновлений відгук.
/product-comments/delete/:id	Delete	Так	Видаляє відгук.
/product-comments/:productId	Get	Ні	Повертає список всіх відгуків конкретного продукту.

Програмний код інших частин серверу наведено у додатку А.

Клієнтська та адміністративна частини веб-додатку розробляється за допомогою бібліотеки React.js. Вони мають основний файл, який підключає всі сторінки [1] та менеджер станів Redux [12] та middleware для нього – Redux-Thunk [13], API-функції відповідають за взаємодію з сервером шляхом надсилання запитів, а actions – це функції, які викликають API-функції та обробляють отримані відповіді з сервера, reducers – функції, які використовуються для зміни стану додатку. Всі перенаправлення виконуються за допомогою компоненту Link [2], це дозволяє використовувати патерн SPA [3]. Також присутні компоненти, які відповідають за відображення різних елементів користувацького інтерфейсу в фронтенді. Кожен компонент може мати свій власний файл, який містить JSX-код для створення структури та вигляду компонента, стилі, які використовуються для оформлення компонента, а також пов'язану з ним логіку. Компоненти можуть бути повторно використовуваними,

що дозволяє зручно використовувати їх у різних частинах додатку, і групуватися за функціональністю, що спрощує організацію та управління кодом.

Запити до серверної частини додатку відправляються за допомогою Axios, Axios - це бібліотека JavaScript, яка надає простий і зручний спосіб здійснювати HTTP-запити з браузера або з серверного середовища. Вона дозволяє виконувати різні типи запитів, такі як GET, POST, PATCH, DELETE та інші, і взаємодіяти з веб-серверами для отримання або передачі даних. На рис. 2.12 показано основні запити, пов'язані з продуктами.

```
export const fetchRecords = () => API.get(`/products`);
export const fetchFilteredRecords = (searchQuery) =>
  API.get(
    `/products/search?search=${searchQuery.search}&genres=${
      searchQuery.selectedGenres || "none"
    }&availability=${searchQuery.availability || "none"}&isColored=${
      searchQuery.isColored || "none"
    }&prices=${searchQuery.prices || "none"}&sort=${searchQuery.sort || "none"}`
  );
export const fetchRecord = (id) => API.get(`/products/${id}`);
export const createRecord = (newRecord) =>
  API.post(`/products/create`, newRecord);
export const updateRecord = (id, updatedRecord) =>
  API.patch(`/products/update/${id}`, updatedRecord);
export const hideRecord = (id, isHidden) => API.patch(`/products/hide/${id}`, isHidden);
```

Рис. 2.12. Запити, пов'язані з продуктами

API – об'єкт axios, створений з використанням основного URL серверу (рис.2.13).

```
const API_LINK = process.env.REACT_APP_SERVER_URL;

const API = axios.create({ baseURL: API_LINK });
```

Рис. 2.13. Створення об'єкту axios

Головний URL серверу знаходиться в файлі середовища .env. Цей файл зазвичай містить змінні оточення, які використовуються для налаштування та конфігурації додатків або середовищ виконання. Даний файл слугує для відокремлення конфігураційних даних від вихідного коду додатку. Це дає можливість змінювати змінні без необхідності перекомпіляції або змінювання вихідного коду додатку. Важливо зазначити, що цей файл містить конфіденційну

інформацію, таку як паролі, ключі API або секретні токени. Тому необхідно дотримуватися належних заходів безпеки та не допускати доступу до цього файлу неповноважним особам [14]. На рис. 2.14 показано файл середовища (всі ключі змінено).

```
admin@ / ? .env
1  REACT_APP_SERVER_URL='http://localhost:8000/'
2  REACT_APP_CLOUD_NAME='deasdsdasddj7asdh4'
3  REACT_APP_UPLOAD_PRESET='s8jasdqr7sasadou'
4  REACT_APP_EMAIL_TEMPLATE_ID="tasdasdasdy2b"
5  REACT_APP_EMAIL_PUBLIC_KEY="5CFf2asadasdadzaL"
6  REACT_APP_EMAIL_SERVICE_ID="sasdasdasd1f"
7  REACT_APP_TINY_EDITOR_API_KEY="tasdasdsarfwi0h0uaxvc6vg87o"
```

Рис. 2.14. Файл середовища

Одними з найважливіших функцій actions є функції getRecords та CreateRecord. На рис. 2.15 показано функцію getRecords.

```
export const getRecords = () => async (dispatch) => {
  try {
    const { data: { records } } = await api.fetchRecords();
    dispatch({ type: "FETCH_RECORDS", payload: records });
    dispatch({ type: "FETCH_ALL_RECORDS", payload: records });
  } catch (error) {
    console.log(error.message);
  }
};
```

Рис. 2.15. Функція getRecords

В даній функції викликається спочатку викликається API-функція fetchRecords, в результаті її виконання сервер повертає масив всіх продуктів, який записується в змінну records. Далі за допомогою dispatch викликається функція для зміни стану додатку, в неї передається отриманий масив records. Дві функції необхідні для правильного фільтрування. Якщо в результаті будь-якої з цих дій виникає помилка, то повідомлення про помилку виводиться в консоль.

На рис. 2.16 показано функцію createRecord.

```
export const createRecord = (record, navigate) => async () => {
  try {
    await api.createRecord(record);
    navigate("/admin/records");
  } catch (error) {
    return error.response.data;
  }
};
```

Рис. 2.16. Функція createRecord

Функція createRecord приймає дані про продукт та викликає API-функцію для створення нового продукту, передаючи ці дані, якщо додавання проходить успішно, то працівник перенаправляється на сторінку з усіма продуктами, інакше повертається повідомлення про помилку, яке буде використано для відображення причини помилки.

На рис. 2.17 показано функція reducer для продуктів.

```
const records = (state = {records: [], allRecords: []}, action) => {
  switch(action.type) {
    case 'FETCH_RECORDS':
      return {...state, records: action.payload};
    case 'FETCH_ALL_RECORDS':
      return {...state, allRecords: action.payload};
    case 'FETCH_FILTERED_RECORDS':
      return {...state, records: action.payload};
    case 'FETCH_RECORD':
      return {...state, record: action.payload};
    case 'HIDE_RECORD':
      return {...state, records: state.records.map((record) => (record._id === action.payload._id ? action.payload : record))};
    default:
      return state;
  }
}

export default records
```

Рис. 2.17. Функція reducer для продуктів

Ця функція змінює стан додатку в залежності від переданого типу, функція викликається з поточним станом та повертає вже змінений стан.

Це допомагає керувати станом додатку в одному місці та впроваджувати принцип однорідності (unidirectional data flow) у фронтенд-додатку, щоб забезпечити передбачуваність та простоту розширення додатку, сприяти перевикористанню коду та покращити його тестованість.

Далі всі reducer-функції комбінуються та підключаються в ядрі додатку.

Для роботи з станом додатку потрібно підключити useEffect, якщо треба отримати певні дані зі стану. useEffect – це хук, який містить callback-функцію,

яка виконується після рендерингу компонента, або при зміні одного елементу з відстежуваних, які знаходяться в масиві залежностей [15]. Також потрібно підключити хук `useSelector`. `useSelector` є одним з хуків бібліотеки `React Redux`, який використовується для вибору даних зі стану [16]. Він дозволяє компоненту підписатися на певну частину стану та отримувати оновлені значення цієї частини стану після змін. Хук `useSelector` отримує функцію вибору (`selector`), яка визначає, які дані зі стану потрібно вибрати. Функція вибору отримує весь стан і повертає вибрану частину цього стану для використання в компоненті. Коли дані, на які підписаний компонент, змінюються, відбувається повторний рендеринг компонента з оновленими значеннями. Також треба підключити `action`-функцію, яка буде використовуватися в даному компоненті. Робота зі станом жанрів для фільтрації показана на рис. 2.18.

```
52   import { getGenres } from "../../actions/genres";
85   const genres = useSelector((state) => state.genres.genres);
216   useEffect(() => {
217     dispatch(getGenres());
218   }, [dispatch]);
```

Рис. 2.18. Робота зі станом жанрів

У клієнтській частині додатку робота з серверною частиною та робота з станом додатку є аналогічними.

Програмний код адміністративної частини наведено у додатку Б.

У табл. 2.6 описано бібліотеки, які були використані для розробки адміністративної частини.

Таблиця 2.6

Використані бібліотеки

Бібліотеки	Опис
@mui/icons-material	Цей пакунок містить іконки матеріалів Google, перетворені у компоненти <code>SvgIcon</code> .
@mui/material	комплексна бібліотека компонентів, яка містить нашу реалізацію системи <code>Material Design</code> від Google.

@reduxjs/toolkit	Офіційний набір інструментів для ефективної розробки Redux.
axios	Клієнт HTTP на основі промісів для браузера та node.js.
chart.js	Прості, але гнучкі діаграми JavaScript для дизайнерів та розробників.
dotenv	Бібліотека для роботи з файлами середовища.
emailjs-com	Бібліотека для відправки повідомлень на електронну пошту.
generate-password-browser	Це (відносно) велика бібліотека для генерації випадкових та унікальних паролів у браузерах.
jwt-decode	Невелика браузерна бібліотека, яка допомагає декодувати токен JWTs

Також були використані інші бібліотеки, необхідні для React.js.

Одним з найважливіших етапів розробки є етап, під час якого розробляється кошик на клієнтський частині.

Оскільки в додатку є можливим оформляти замовлення без здійснення входу до власного акаунту, доцільним є зберігання інформації про обрані продукти в localStorage. На рис. 2.19 показано функцію, для додавання товару в кошик.

```
const handleClick = () => {
  let cart = JSON.parse(localStorage.getItem("cart"));
  if (cart) {
    cart.push({ count: 1, id: id });
    localStorage.setItem("cart", JSON.stringify(cart));
  } else {
    localStorage.setItem("cart", JSON.stringify([{ count: 1, id: id }]));
  }
  setDisabled(true);
};
```

Рис. 2.19. Функція для додавання продукту в кошик

Дана функція перевіряє чи кошик наявний, якщо так – додає новий елемент в кінець кошику, інакше створює кошик з id обраного продукту і кількістю 1.

При переході на сторінку Cart за допомогою action-функції `getCartRecords` отримуються повна інформація про продукти в кошику. Далі інформація про кожен окремий продукт передається до окремого компонента `cartRow`, за допомогою якого користувач може обирати кількість кожного продукту, яку він хоче замовити або прибрати продукт з корзини. На рис. 2.20 показано функції для збільшення та зменшення кількості продукту.

```
const handleClickPlus = () => {
  if (counter < record.itemsAvailable) {
    setCounter((prev) => prev + 1);
    changeTotalPrice(record.price);
    let cart = JSON.parse(localStorage.getItem("cart"));
    cart.forEach(element => {
      if (element.id === record._id)
        element.count = counter + 1
    });
    localStorage.setItem("cart", JSON.stringify(cart));
  }
};

const handleClickMinus = () => {
  if (counter > 1) {
    setCounter((prev) => prev - 1);
    changeTotalPrice(-record.price);
    let cart = JSON.parse(localStorage.getItem("cart"));
    cart.forEach(element => {
      if (element.id === record._id)
        element.count = counter - 1
    });
    localStorage.setItem("cart", JSON.stringify(cart));
  }
};
```

Рис. 2.20. Функції для збільшення та зменшення кількості продукту

В даних функціях перевіряється чи можливо змінити кількість обраного продукту, якщо так, то збільшується кількість обраного продукту та загальна ціна кошику, також кількість змінюється в кошику `localStorage`. При натисканні на кнопку смітника, продукт прибирається з кошика в `localStorage` та після цього знову відбувається отримання всіх продуктів з кошика в `localStorage` (рис. 2.21).

```
const handleDeleteClick = (id, price) => {
  let cart = JSON.parse(localStorage.getItem("cart"));
  localStorage.setItem(
    "cart",
    JSON.stringify(cart.filter((e) => e.id !== id))
  );
  changeTotalPrice(price);
  dispatch(
    getCartRecords(
      JSON.parse(localStorage.getItem("cart")).map(({ id }) => id)
    )
  );
};
```

Рис. 2.21. Функції для збільшення та зменшення кількості продукту

Програмний код адміністративної частини наведено у додатку В.

Висновки до розділу 2

Після ретельного аналізу різноманітних сценаріїв використання було встановлено низку вимог до проєкту. Оскільки робота виконувалась в групі, було здійснено розподілення обов'язків між учасниками групи. Серед власних вимог зокрема важливими є наступні: реалізація функціоналу прослуховування музики (якщо це технічно можливо), наявність статистики на адміністративному розділі системи. Процес визначення основних вимог до роботи системи дозволив виявити можливість надати неавторизованим користувачам можливість взаємодіяти з кошиком.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ДОДАТКОМ

3.1. Порядок встановлення та налаштування системи.

Оскільки додаток є клієнт-серверним, отже серверна, клієнтська та адміністративна частини додатку можуть розміщуватись окремо. Мінімальні вимоги для розгортання серверної частини додатку залежать від орієнтовного трафіку додатку, але краще розглянути вимоги з запасом, отже розгортання серверної частини додатку потрібно проводити на сервері з такими вимогами:

- двоядерний процесор з тактовою частотою 2 ГГц або вище;
- 2 ГБ ОЗУ;
- 1 ГБ вільного місця на жорсткому диску;

Для запуску серверної частини додатку потрібно виконати наступний ряд дій:

1. Встановити Node.js.
2. За допомогою команди `npm install` встановити залежності додатку, які знаходяться в файлі `package.json`.
3. Запустити сервер.

Для розгортання клієнтської та адміністративної частини додатку потрібно задовільнити та виконати аналогічні вимоги та дії.

Для взаємодії з додатком підійде будь-який сучасний веб-браузер такий як Google Chrome версія 45+, Mozilla Firefox версія 38+ та подібні.

3.2. Структура інтерфейсу. Інтерфейс та порядок роботи з веб-додатком.

При відкритті адміністративної частини додатку користувач побачить вікно для входу, оскільки для будь-якої взаємодії з додатком треба авторизуватись (рис. 3.1).

Вхід

Email Address *

Password *

Увійти

Рис. 3.1. Сторінка входу

Якщо було введено неправильну електронну пошту або пароль, буде відображено відповідну помилку (рис.3.2).

Вхід

Email Address *

sdasd@asdads.com

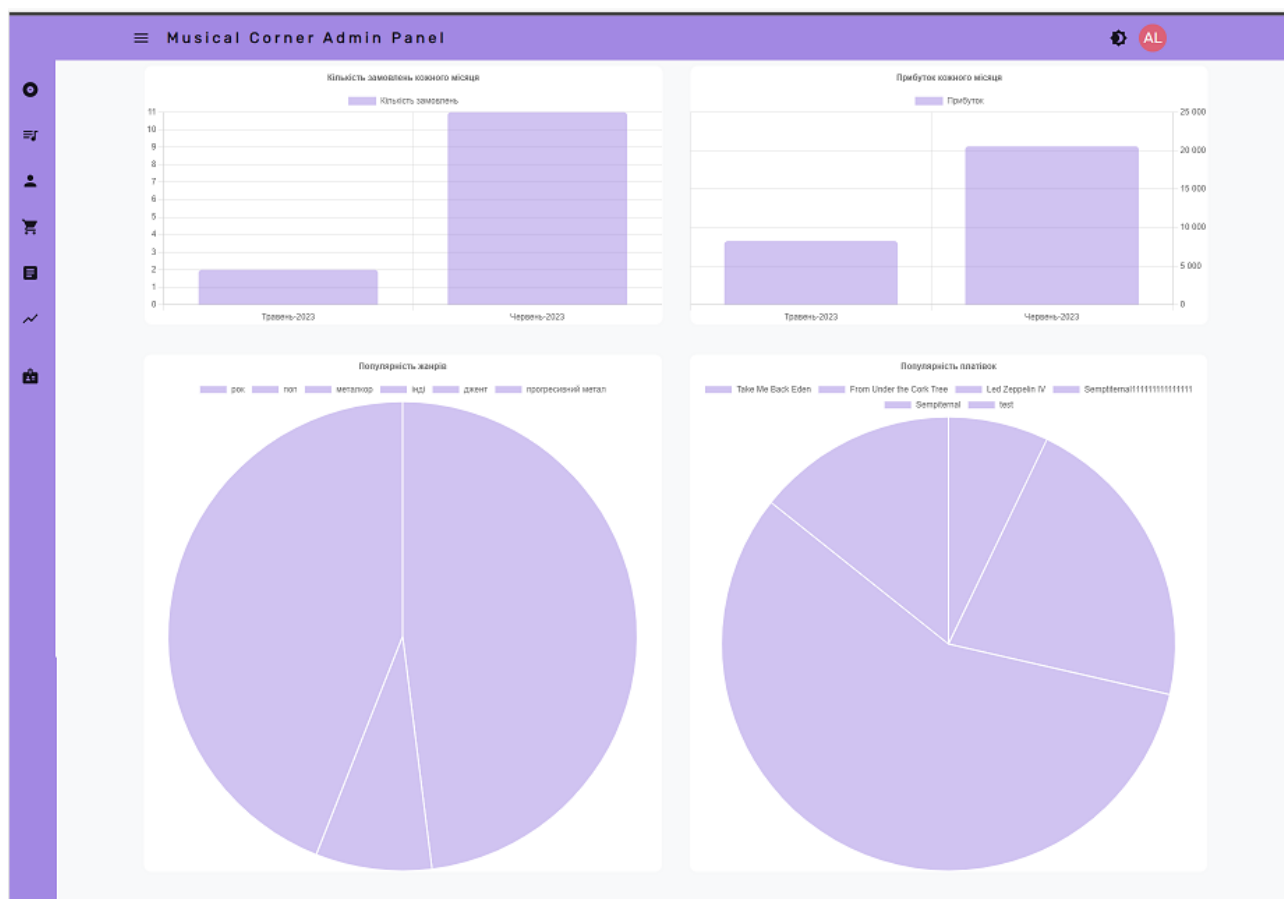
Password *

Увійти

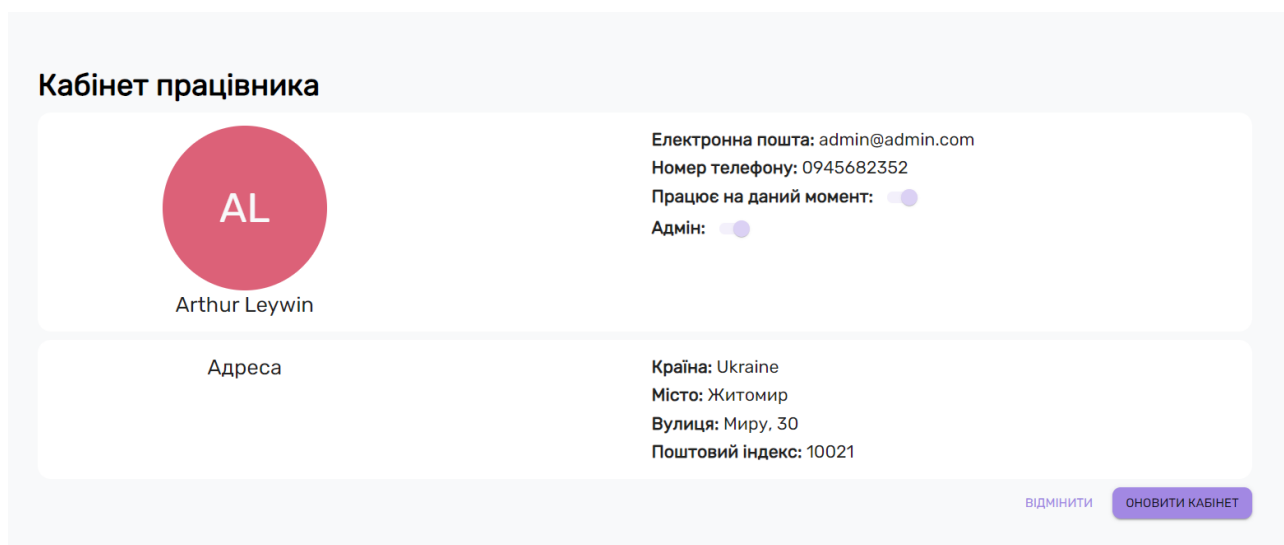
Невірний email

Рис. 3.2. Повідомлення про помилку

Після входу до власного профілю працівника перенаправить на головну сторінку додатку, на якому він може побачити статистику (рис. 3.3).



Працівник може натиснути на власний аватар, після цього відкриється меню, в якому є кнопка яка перенаправляє на сторінку власного кабінету, друга кнопка дозволяє вийти з власного акаунту. На рис. 3.4 показано кабінет працівника.



Працівник може оновити власний кабінет (рис. 3.5).

Оновити кабінет

Загальна інформація

Номер телефону *

0945682352

Старий пароль

Новий пароль

Адреса

Країна

Ukraine

Місто *

Житомир

Вулиця *

Миру, 30

Поштовий індекс *

10021

ВІДМІНИТИ

ОНОВИТИ ПРОФІЛЬ

Рис. 3.5. Оновлення кабінету

При натисканні на іконку платівки в лівій навігаційній панелі відкриється сторінка з списком усіх платівок (рис. 3.6).

Платівки

+ ДОДАТИ ПЛАТІВКУ

Шукати

Кольорова?

Сортування

Жанри

Наявність

Фільтрація по ціні

ПОШУК

X

Rows per page: 10 1-8 of 8

Назва платівки	Автор	Жанри	Ціна	Кількість на складі	Дії
DEVIL MAY CRY 5 (DELUXE DOUBLE VINYL)	Devil May Cry 5	саундтрек	15608	3	
Elden Ring - Official Vinyl	Elden Ring	саундтрек	16108	5	
Take Me Back Eden	Sleep Token	джент прогресивний метал	15608	15	
From Under the Cork Tree	Fall Out Boy	рок інді	15008	15	
Led Zeppelin IV	Led Zeppelin	рок	20008	9	
Semptiternal111111111111111111	Bring Me The Horizon	металкор рок	1008	5	

Рис. 3.6. Сторінка з списком платівок

Можна здійснити пошук платівки, фільтрацію по кольоровості платівки, по жанрам, наявності та по ціні, також можна здійснити сортування (рис. 3.7).

Платівки

+ ДОДАТИ ПЛАТІВКУ

Шукати

Q

Кольорова?

▼

Сортування

Від дорогих до дешевих ▼

Жанри

рок

▼

Наявність

▼

Фільтрація по ціні

▼

Q пошук

×

Rows per page: 10 1-4 of 4 < >

Назва платівки	Автор	Жанри	Ціна	Кількість на складі	Дії
Sempiternal	Bring Me The Horizon	металкор рок	20018	5	і ✎ 🗑
Led Zeppelin IV	Led Zeppelin	рок	20008	9	і ✎ 🗑
From Under the Cork Tree	Fall Out Boy	рок інді	15008	15	і ✎ 🗑
Sempiternal11111111111111111111	Bring Me The Horizon	металкор рок	1008	5	і ✎ 🗑

Рис. 3.7. Фільтрація та сортування платівок

Для скидання всіх фільтрів та сортування потрібно натиснути на кнопку з зображенням хреста, справа від кнопки «Пошук».

Можна переглянути повну інформацію про обрану платівку, для цього треба натиснути на іконку інформації (рис. 3.8 – 3.9).

Повна інформація

✎ ОНОВИТИ ПЛАТІВКУ

Назва	Take Me Back Eden
Виконавець	Sleep Token
Опис	Take Me Back to Eden - третій студійний альбом англійського рок-гурту Sleep Token. Продюсований фронтменом гурту Весселом та Карлом Бауном, він вийшов 19 травня 2023 року на лейблі Spinefarm Records.
Ціна	15608
Список пісень	1. Chokehold 2. The Summoning 3. Granite 4. Aqua Regia 5. Vore 6. Ascensionism 7. Are You Really Okay? 8. The Apparition 9. DYWTYLM 10. Rain 11. Take Me Back to Eden 12. Euclid

Рис. 3.8. Інформація про платівку (частина 1)

Жанри	1. джент 2. прогресивний метал
Кількість на складі	15 шт.
Дата виходу альбому	19 травня 2023 р.
Дата виходу платівки	19 травня 2023 р.
Час програвання платівки	63:26
Лейбл	Spinefarm
Кольорова?	Так
Кількість платівок в комплекті	2

Зображення



Рис. 3.9. Інформація про платівку (частина 2)

Також платівку можна приховати, в такому випадку вона не буде відображатись для клієнтів, для цього потрібно натиснути на іконку ока (рис. 3.10).

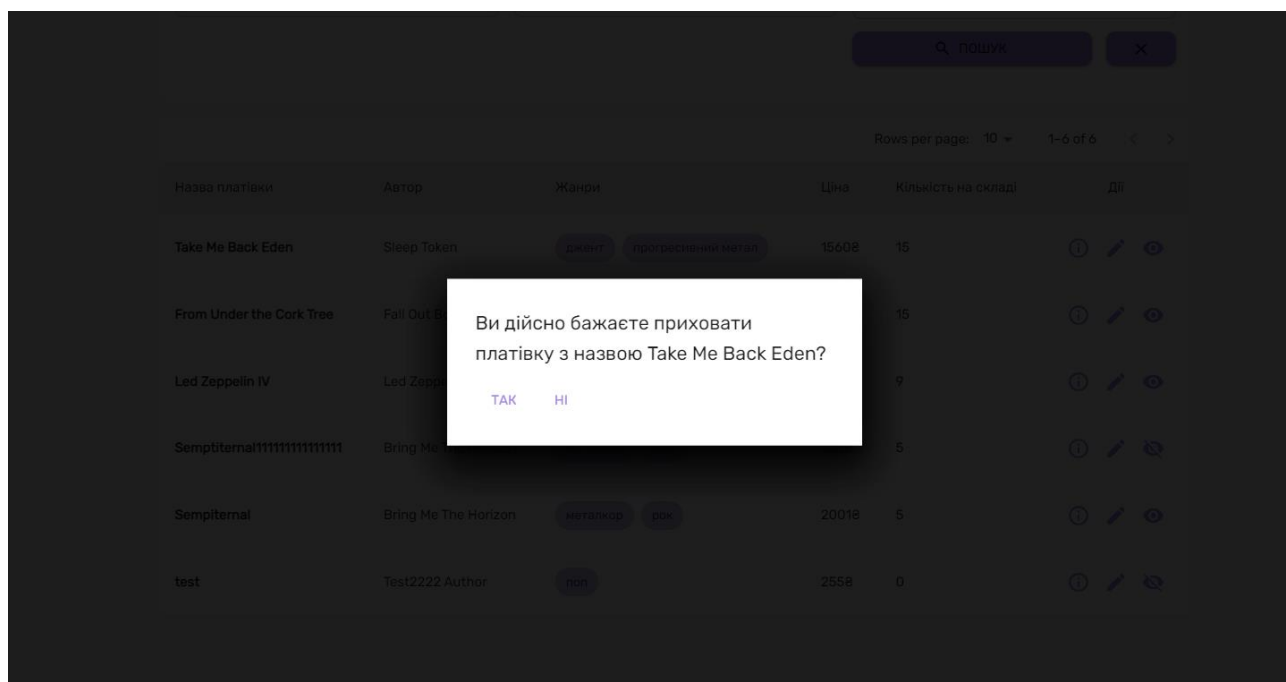


Рис. 3.10. Приховування платівки

Також будь-яку інформацію про платівку можна відредагувати, для цього потрібно натиснути на іконку з олівцем (рис. 3.11 – 3.12).

Оновити платівку

Загальна інформація

Назва платівки *

Take Me Back Eden

Автор платівки *

Sleep Token

Ціна *

1560

₺

Жанри

джент

прогресивний метал

Кількість на складі *

15

Додаткова інформація

Опис платівки *

Take Me Back to Eden – третій студійний альбом англійського рок-гурту Sleep Token. Продюсований фронтменом гурту Весселом та Карлом Бауном, він вийшов 19 травня 2023 року на лейблі Spinefarm Records.

Треки, що містяться на даній платівці(один трек в одному рядку) *

Chokehold
The Summoning
Granite
Aqua Regia
Vore

Рис. 3.11. Оновлення даних платівки (частина 1)

Час програвання платівки *

63:26

Лейбл *

Spinefarm

Кількість платівок у комплекті *

2

Дата випуску альбому *

19.05.2023

Дата випуску платівки *

19.05.2023

Країна

United Kingdom of Great Britain and Northern Ireland

☒ Кольорова?

ДОДАТИ ЗОБРАЖЕННЯ

ВІДМІНИТИ

ОНОВИТИ ПЛАТІВКУ

Рис. 3.12. Оновлення даних платівки (частина 2)

Також можна додавати нові платівки, форма для додавання виглядає аналогічно, після додавання платівки вона автоматично буде прихована.

					ІПЗ.КР.Б – 121 – 23 – ІПЗ	
						51

Працівник може переглянути список всіх жанрів, якщо натисне на іконку жанрів в лівій навігаційній панелі (рис. 3.13).

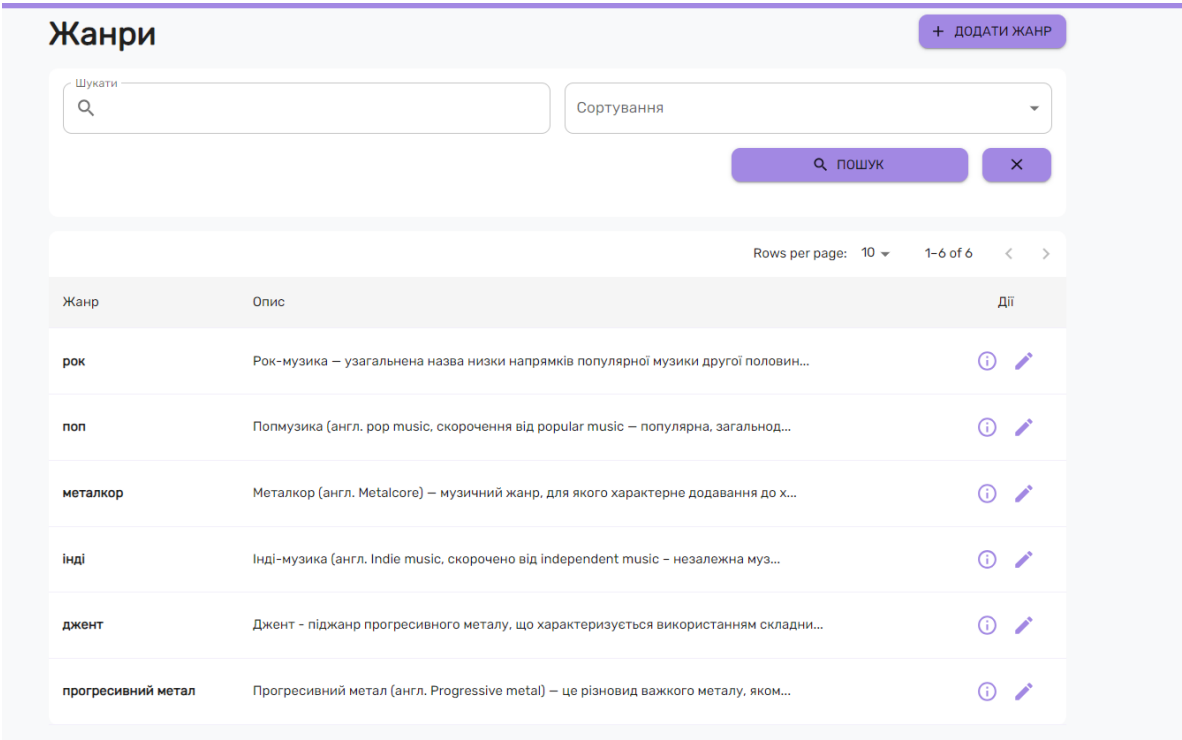


Рис. 3.13. Сторінка з списком жанрів

Аналогічно з платівками, список жанрів можна відсортувати а також здійснити пошук, можна переглянути детальну інформацію про жанр, яка складається з назви та його опису, також можна оновити інформацію про жанр. Також є можливим додавання нового жанру (рис. 3.14).

Рис. 3.14. Додавання жанру

Якщо працівник має права адміністратора він може переглянути список всіх працівників (рис. 3.15).

Працівники

+ ДОДАТИ ПРАЦІВНИКА

Шукати

Сортування

Адмін?

Працює?

ПОШУК

×

Rows per page: 10

1-5 of 5

<

>

Ім'я	Email	Номер телефону	Працює?	Адмін?	Дії
Leywin Arthur	admin@admin.com	0945682352	✓	✓	
Паркер Пітер	employeeest@employeeest.com	0933456566	✓	✗	
Лецишина Вікторія	svn.junipero@gmail.com	0945682351	✗	✓	
Королева Мара	employee@employee.com	0933456567	✓	✗	
Демченко Ярослав	yaroslavflash@gmail.com	0945671456	✗	✓	

Рис. 3.15. Сторінка з списком працівників

Аналогічно можна здійснити пошук працівників, сортування та фільтрацію їх списку. Також можна оновити інформацію про будь-якого працівника, для оновлення доступно більше інформації, ніж при оновленні особистого кабінету (рис. 3.16).

Загальна інформація

Ім'я *

Пітер

Прізвище *

Паркер

По-батькові

Електронна пошта *

employeeest@employeeest.com

Номер телефону *

0933456566

Адреса

Країна

Ukraine

Місто *

Zhytomyr

Вулиця *

2

Поштовий індекс *

10020

Додаткова інформація

☐ Є адміном?

☒ Працює на даний момент?

Рис. 3.16. Оновлення інформації про працівника

Також присутня можливість видалення інших працівників (рис. 3.17).

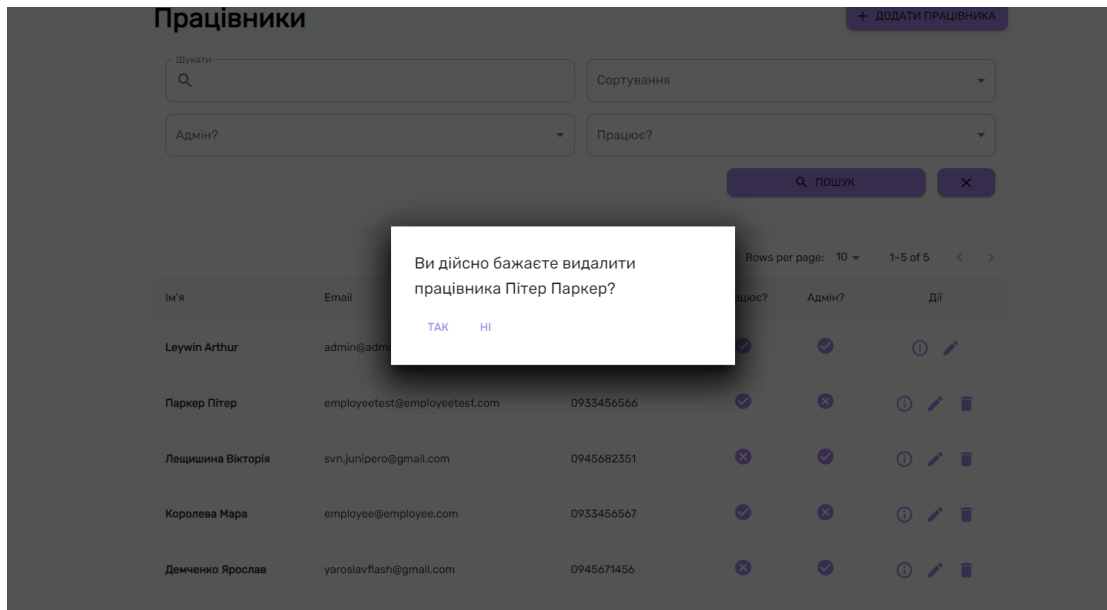


Рис. 3.17. Видалення працівника

Також є можливість додати нового працівника, при додаванні пароль генерується автоматично та повідомлення з даними для входу надсилається працівнику на електронну пошту (рис. 3.18).

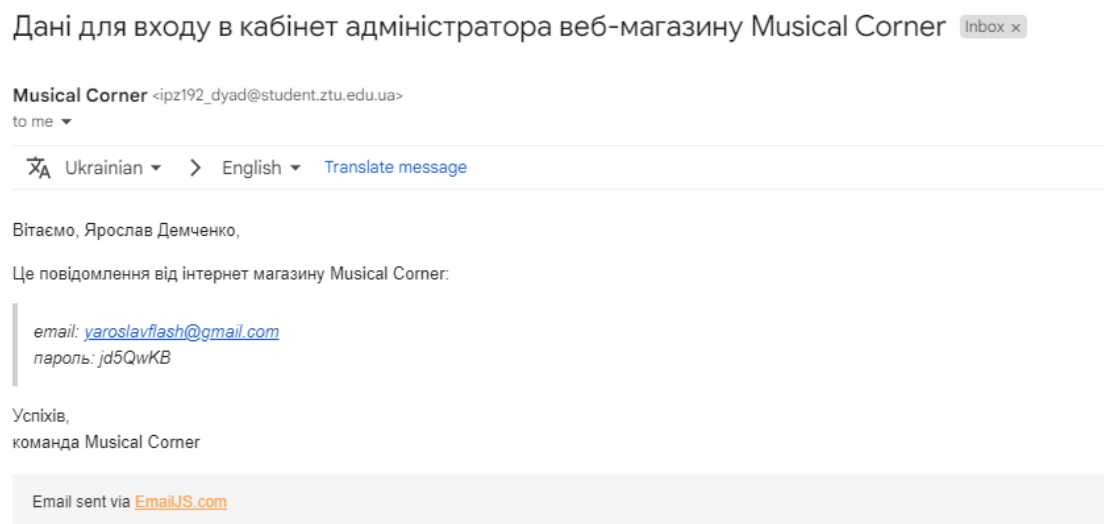


Рис. 3.18. Приклад електронного листа

При відкритті клієнтської частини додатку користувач побачить головну сторінку, яка була розроблена Лещишиною Вікторією, при натисканні на кнопку «Платівки» у верхній навігаційній панелі користувач попадає на сторінку зі списком платівок (рис. 3.19).

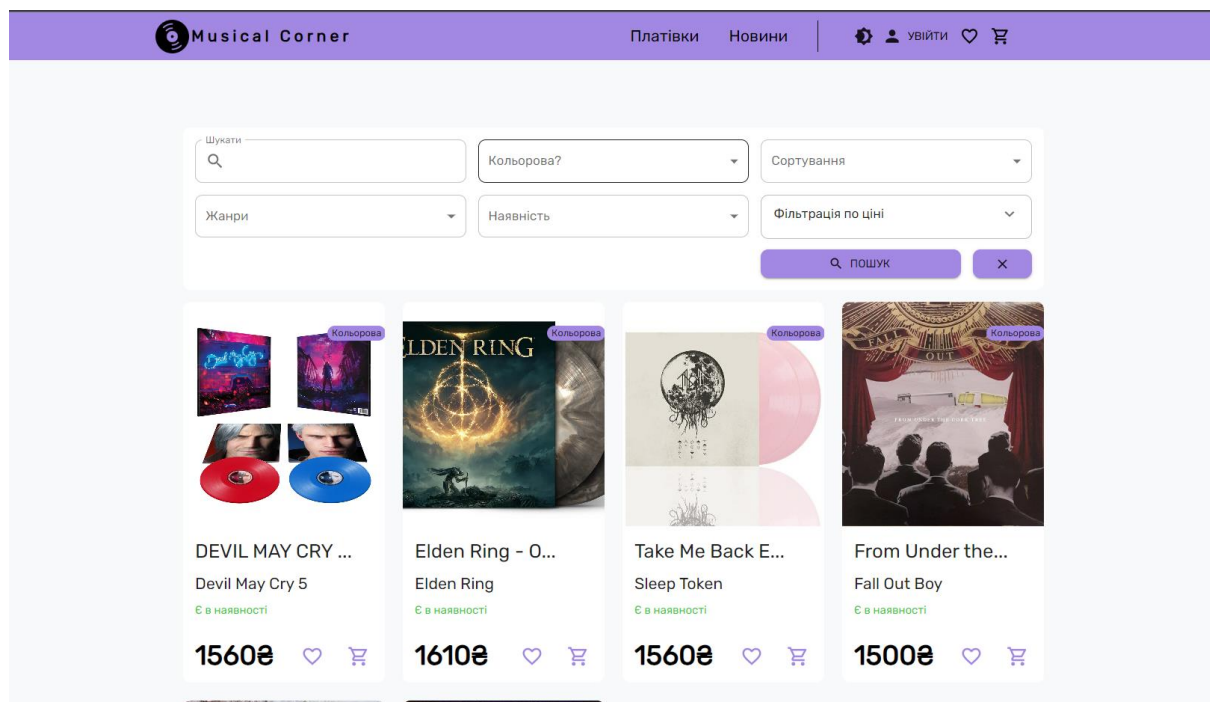


Рис. 3.19. Сторінка з списком платівок

Можна здійснити пошук платівки, фільтрацію по кольоровості платівки, по жанрам, наявності та по ціні, також можна здійснити сортування, аналогічно як у адміністративній частині.

При натисканні на будь-яку платівку відкриється вікно з головною інформацією про платівку (рис. 3.20).

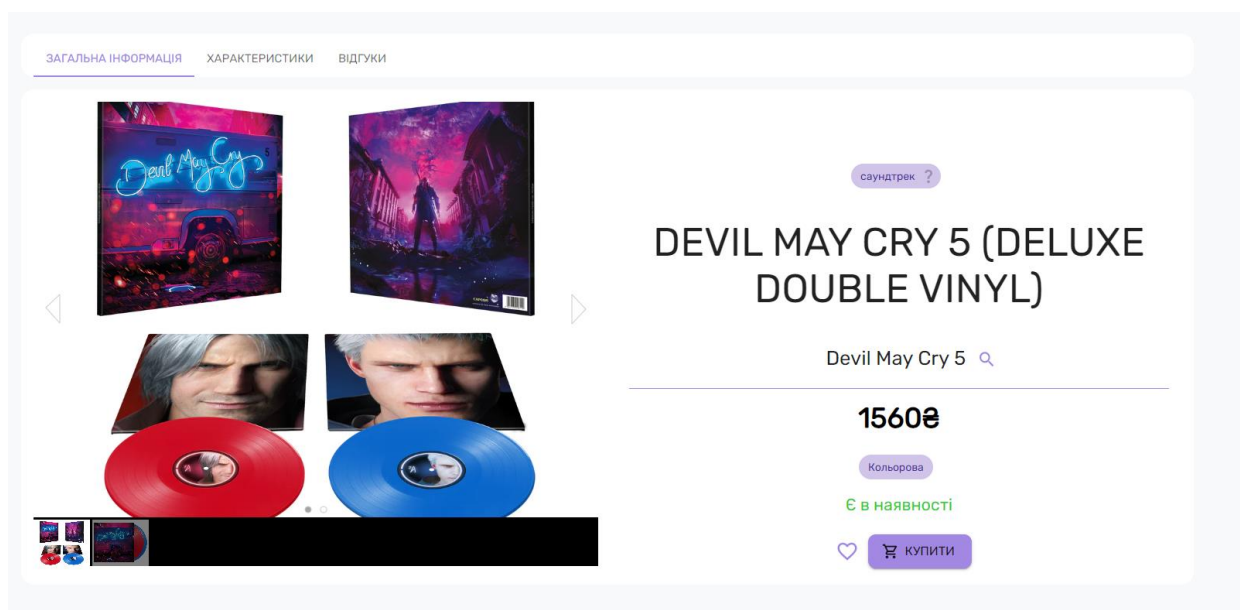


Рис. 3.20. Сторінка з відображенням загальної інформації про платівку

При наведенні на жанр, відкривається підказка з інформацією про даний жанр, при натисканні на значок збільшувального скла відкривається сторінка з списком платівок від даного автора. Також на даній сторінці при натисканні на кнопку «Купити» можна додати платівку в кошик. Якщо вибрати вкладку «Характеристики» в списку вкладок, відкриється сторінка з повними характеристиками платівки (рис. 3.21).

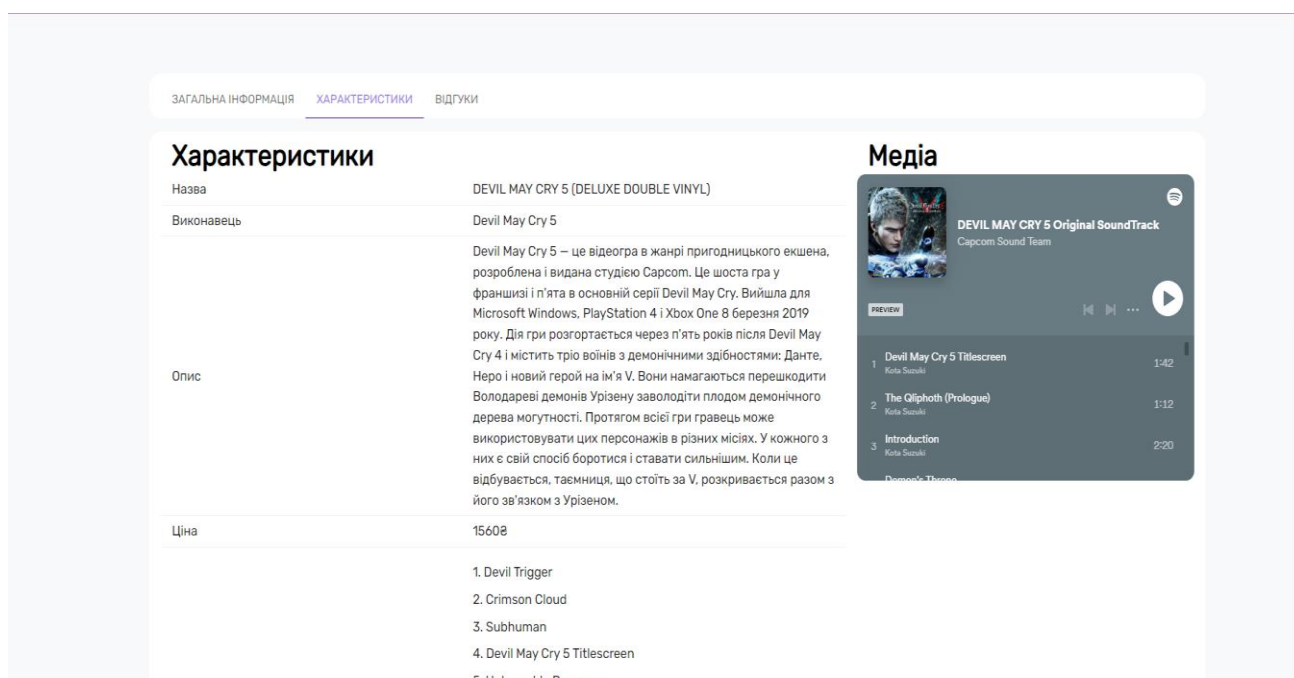


Рис. 3.21. Сторінка з відображенням повних характеристик про платівку

На даній сторінці також можна прослухати треки даної платівки (якщо доступно).

Якщо вибрати вкладку «Відгуки» відкриється сторінка з відгуками до даної платівки. Якщо користувач не автентифікувався, він не може писати відгуки, також він не може редагувати їх (рис. 3.22).

Рис. 3.22. Сторінка відгуків

Якщо автентифікуватись та перейти на сторінку з відгуками, то відображається форма для написання відгуку, та кнопки редагування та видалення власних відгуків (рис. 3.23).

Рис. 3.23. Сторінка відгуків, якщо користувач автентифікований

Якщо оновити власний відгук, то буде відображатися інформація про дату оновлення відгуку.

Якщо на сторінці з усіма платівками або на сторінці з повною інформацією про платівку натиснути на кнопку з зображенням кошика, ця платівка додається в кошик в кількості 1 штука. При переході на сторінку кошика відображається обрані платівки та їх загальна ціна (рис. 3.24).

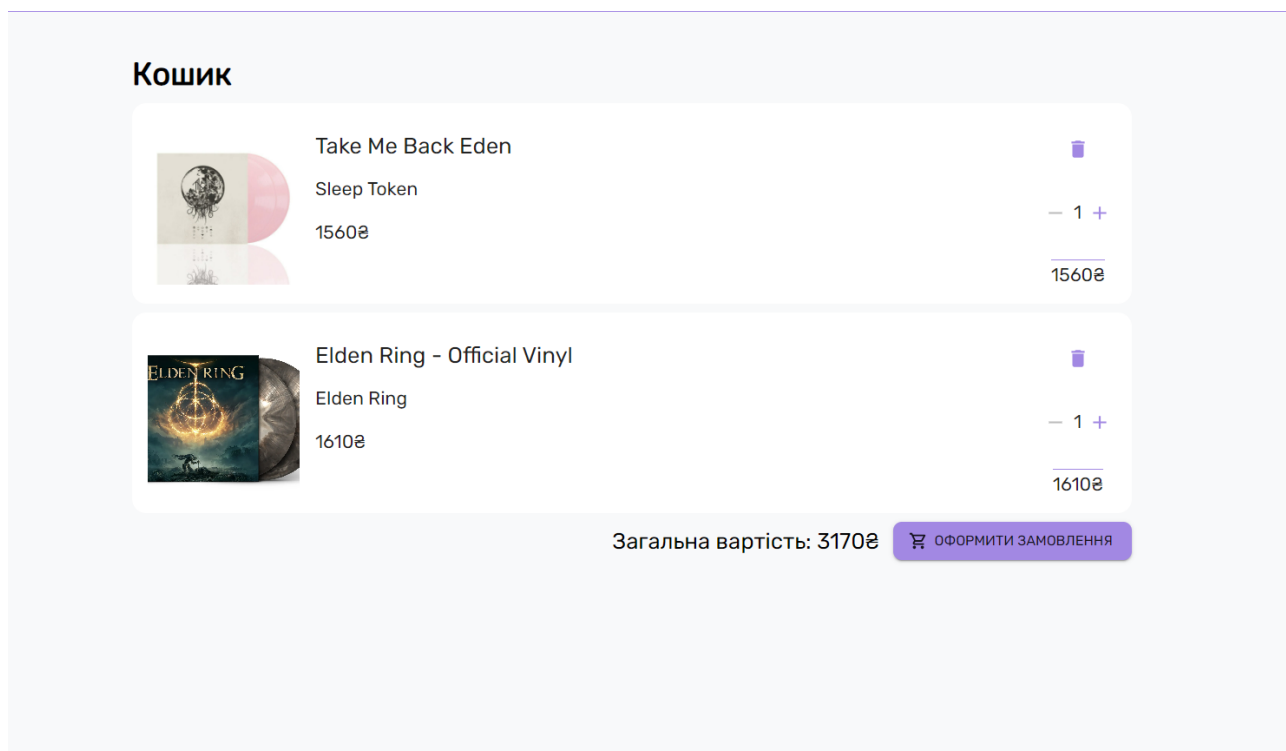


Рис. 3.24. Сторінка кошику

При зміні кількості продуктів або видаленні продукту з кошику, загальна ціна також змінюється.

3.3. Тестування роботи веб-додатку

Після завершення розробки веб-застосунку, проводилося тестування, щоб переконатися, що він відповідає вимогам, поставленим у даній роботі. Тестування функціональних можливостей та роботи елементів клієнтської та адміністративної частини було виконано шляхом ручного тестування. В результаті проведення даного типу тестування було виявлено та виправлено деякі помилки.

Тестування серверної частини додатку було проведено за допомогою програми Postman. Postman - це популярний інструмент для розробників API, який надає можливість створювати, тестувати та взаємодіяти з API. Він дозволяє легко відправляти HTTP-запити, перевіряти відповіді сервера, налаштовувати параметри запитів і аналізувати результати.

Postman надає зручний інтерфейс, в якому можна створювати різні типи запитів, такі як GET, POST, PUT, DELETE і багато інших. Він також дозволяє налаштовувати заголовки, параметри, тіло запиту та автентифікацію. За допомогою Postman можна відправляти запити на реальний сервер або локальний розробничий сервер для тестування та налагодження API [17].

На рис. 3.25 наведено приклад тестування за допомогою програми Postman.

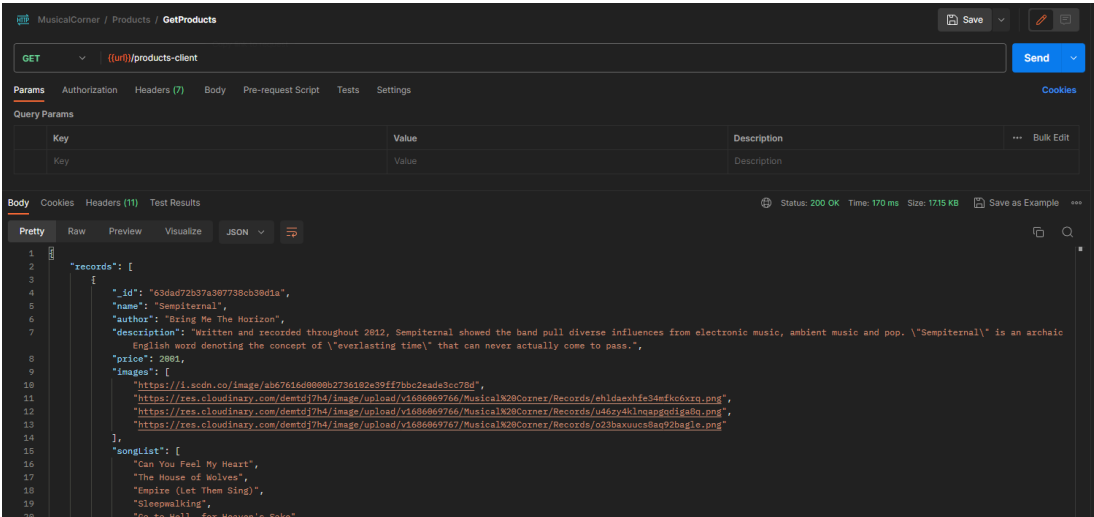


Рис. 3.25. Приклад тестування в програмі Postman

Висновки до розділу 3

В даному розділі було наведено порядок встановлення веб-додатку, та визначено мінімальну конфігурацію системи для успішного розгортання додатку.

Оскільки були дотримані усі вимоги до розробки додатку, тому були уникнуті проблеми під час розробки та використання додатку.

Також було проведено тестування усіх частин додатку, в результаті виконаного тестування було знайдено та виправлено декілька помилок.

ВИСНОВКИ

В процесі розробки кваліфікаційної роботи бакалавра було спроектовано веб-додаток магазину вінілових платівок «Musical Corner». Результатів було досягнуто, оскільки були виконані дані кроки:

1. Виконано аналіз предметної області та існуючих аналогів з метою виявлення потреб користувачів і визначення основних особливостей веб-додатку. Було обрано базу даних MongoDB та стек MERN як метод розробки додатку.

2. Встановлено технічне завдання, визначено потік роботи, оскільки робота виконувалась в групі, було здійснено розподілення обов'язків, описано структуру веб-додатку, спроектовано базу даних та розроблено весь необхідний функціонал додатку

3. Наведено характеристики системи для оптимального розгортання серверної, клієнтської та адміністративної частини додатку, описано характеристики для запуску додатку для отримання найкращого досвіду. Ретельно описано порядок взаємодії клієнта та працівника з веб-додатком. Проведено тестування системи, було виявлено і усунено помилки.

В результаті проведеної роботи було розроблено елементи веб-додатку магазину вінілових платівок, а саме продукти, жанри та працівники, які відповідають усім зазначеним вище вимогам та потребам додатку. Було забезпечено високу якість додатку. Отриманий додаток готовий для використання потенційним покупцем або для використання при відкритті власного бізнесу. В майбутньому планується покращення та розширення веб-додатку, що зумовлено можливостями обраних технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. createBrowserRouter [Електронний ресурс] – Режим доступу до ресурсу: <https://reactrouter.com/en/main/routers/create-browser-router>
2. Link [Електронний ресурс] – Режим доступу до ресурсу: <https://reactrouter.com/en/main/components/link>
3. Mary Boyd. Single Page Applications: A Powerful Design Pattern for Modern Web Apps – Medium [Електронний ресурс] / Mary Boyd – 9 березня 2018 р. – Режим доступу до ресурсу: <https://medium.com/a-lady-dev/single-page-applications-a-powerful-design-pattern-for-modern-web-apps-ec3590bb7e7a>
4. What is Client Server Architecture? [Електронний ресурс] – Режим доступу до ресурсу: <https://intellipaat.com/blog/what-is-client-server-architecture/?US>
5. Advantages of MongoDB [Електронний ресурс] – Режим доступу до ресурсу: <https://www.mongodb.com/advantages-of-mongodb>
6. Pros and Cons of ReactJS [Електронний ресурс] – Режим доступу до ресурсу: <https://www.javatpoint.com/pros-and-cons-of-react>
7. Node.js Modules [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/nodejs/nodejs_modules.asp
8. Geshan Manandhar. Organizing your Express.js project structure for better productivity – LogRocket [Електронний ресурс] / Geshan Manandhar – 10 січня 2022 р. – Режим доступу до ресурсу: <https://blog.logrocket.com/organizing-express-js-project-structure-better-productivity/>
9. What Is A RESTful API? [Електронний ресурс] – Режим доступу до ресурсу: <https://aws.amazon.com/what-is/restful-api/>
10. Mongoose getting Started [Електронний ресурс] – Режим доступу до ресурсу: <https://mongoosejs.com/docs/index.html>
11. Populate [Електронний ресурс] – Режим доступу до ресурсу: <https://mongoosejs.com/docs/populate.html>
12. React Redux Quick Start [Електронний ресурс] – Режим доступу до ресурсу: <https://react-redux.js.org/tutorials/quick-start>

					ІІЗ.КР.Б – 121 – 23 – ІІЗ	
						61

13. Redux-thunk [Електронний ресурс] – Режим доступу до ресурсу:
<https://github.com/reduxjs/redux-thunk>
14. Sam Pakvis. Using environment variables in React – Medium [Електронний ресурс] / Sam Pakvis – 25 травня 2018 р. – Режим доступу до ресурсу:
<https://trekinbami.medium.com/using-environment-variables-in-react-6b0a99d83cf5>
15. useEffect [Електронний ресурс] – Режим доступу до ресурсу:
<https://react.dev/reference/react/useEffect>
16. Reinald Reynoso. React-Redux Hooks: useSelector and useDispatch – GitConnected [Електронний ресурс] / Reinald Reynoso – 2 грудня 2019 р. – Режим доступу до ресурсу: <https://levelup.gitconnected.com/react-redux-hooks-useselector-and-usedispatch-f7d8c7f75cdd>
17. Postman overview [Електронний ресурс] – Режим доступу до ресурсу:
<https://learning.postman.com/docs/introduction/overview/>

ДОДАТКИ

					ПЗ.КР.Б – 121 – 23 – ПЗ	
						63

Модель продукту

```

const mongoose = require("mongoose");
const Int32 = require("mongoose-int32").loadType(mongoose);

let productSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Поле "Назва" не може бути порожнім'],
    trim: true,
  },
  author: {
    type: String,
    required: [true, 'Поле "Автор" не може бути порожнім'],
    trim: true,
  },
  description: {
    type: String,
    required: [true, 'Поле "Опис" не може бути порожнім'],
    trim: true,
  },
  recordLength: {
    type: String,
    required: [true, 'Поле "Час програвання платівки" не може бути порожнім'],
  },
  label: {
    type: String,
    required: [true, 'Поле "Лейбл" не може бути порожнім'],
    trim: true
  },
  price: {
    type: Number,
    required: [true, 'Поле "Ціна" не може бути порожнім'],
    validate(value) {
      if (value <= 0) {
        throw new Error("Ціна повинна бути більше 0")
      }
    }
  },
  images: {
    type: [String],
    default: ['https://grandmart.biz/Products/default.jpg'] //замінити
  },
  songList: {
    type: [String],
    validate(value) {
      if (value.length < 1) {
        throw new Error("Платівка має містити хоча б один трек")
      }
    }
  }
});

```

```

    },
    releaseDate: {
      type: Date,
      default: new Date().toISOString().split('T')[0]
    },
    productionDate: {
      type: Date,
      default: new Date().toISOString().split('T')[0]
    },
    itemsAvailable: {
      type: Int32,
      default: 1,
      trim: true,
    },
    genres: {
      type: [mongoose.Schema.Types.ObjectId],
      ref: "Genre",
      validate(value) {
        if (value.length < 1) {
          throw new Error("Платівка має містити хоча б один жанр")
        }
      }
    },
    country: {
      type: String,
      required: [true, 'Поле "Країна" не може бути порожнім'],
      trim: true,
    },
    isColored: {
      type: Boolean,
      default: false
    },
    recordsNumber: {
      type: Int32,
      default: 1,
      trim: true,
    },
    isHidden: {
      type: Boolean,
      default: true
    }
  }
});

productSchema.virtual('users', {
  ref: "User",
  localField: '_id',
  foreignField: 'favourites'
});

```

```

productSchema.virtual('orders', {
  ref: "Order",
  localField: '_id',
  foreignField: 'productId'
});

productSchema.virtual('productComments', {
  ref: "ProductComment",
  localField: '_id',
  foreignField: 'productId'
});

productSchema.index({'$*': 'text'})

const Product = mongoose.model('Product', productSchema)

module.exports = Product;

```

Контролери продукту

```

const Product = require("../models/product");

// Display list of all Products.
exports.products_list = async (req, res) => {
  try {
    const records = await Product.find().sort({$natural: -1}).populate("genres");
    res.status(200).send({ records: records });
  } catch (error) {
    res.status(500).send(error.message);
  }
};

exports.products_list_client = async (req, res) => {
  try {
    const records = await Product.find({isHidden: false}).sort({$natural: -1}).populate("genres");
    res.status(200).send({ records: records });
  } catch (error) {
    res.status(500).send(error.message);
  }
};

// Display list of filtered Products.
exports.filtered_products_list = async (req, res) => {
  let filter = {},
    sorting = {};
  const { search, genres, availability, prices, sort, isColored, client } = req.query;
  if (
    (typeof search === "string" || search instanceof String) && search.length > 0
  ) {

```

```

    filter.$text = { $search: search };
  }
  if ((typeof client === "string" || client instanceof String) && client.length > 0
  && client === "true")
    filter.isHidden = { $eq: false };
  if (genres.length > 0 && genres !== "none")
    filter.genres = { $in: genres.split(",") };
  if ((typeof availability === "string" || availability instanceof String) &&
  availability.length > 0 && availability !== "none") {
    if (availability === "true") filter.itemsAvailable = { $gt: 0 };
    else if (availability === "false") filter.itemsAvailable = { $eq: 0 };
  }
  if ((typeof isColored === "string" || isColored instanceof String) &&
  isColored.length > 0 && isColored !== "none") {
    if (isColored === "true") filter.isColored = { $eq: true };
    else if (isColored === "false") filter.isColored = { $eq: false };
  }
  if (prices.length > 0 && prices !== "none")
    filter.price = { $gte: prices.split(",")[0], $lte: prices.split(",")[1] };
  if ((typeof sort === "string" || sort instanceof String) && sort.length > 0 &&
  sort !== "none") {
    if (Number(sort[sort.length - 1]) === 1)
      sorting[sort.slice(0, sort.length - 1)] = 1;
    else sorting[sort.slice(0, sort.length - 1)] = -1;
  }
  try {
    const records = await Product.find(filter)
      .collation({ locale: "en" })
      .sort(sorting)
      .populate("genres");
    res.status(200).send({ records: records });
  } catch (error) {
    res.status(500).send(error.message);
  }
};

// Display list of Cart Products.
exports.cart_products_list = async (req, res) => {
  let filter = {}
  const { ids } = req.body;
  if (ids && ids.length > 0)
    filter._id = { $in: ids };
  else
    return res.status(200).send({ records: [] });
  try {
    const records = await Product.find(filter)
      .status(200).send({ records: records });
  } catch (error) {
    res.status(500).send(error.message);
  }
}

```

```

};

//Display details about one Product
exports.product_detail = async (req, res) => {
  try {
    const record = await Product.findById(req.params.id).populate("genres");
    if (!record) {
      return res.status(404).send();
    }
    res.status(200).send(record);
  } catch (error) {
    res.status(500).send(error.message);
  }
};

//Create new Product
exports.product_create = async (req, res) => {
  try {
    const record = new Product({
      ...req.body,
    });
    await record.save();
    res.status(201).send({ success: true, record: record });
  } catch (error) {
    let errM = "";
    for (const [key, value] of Object.entries(error.errors)) {
      if (Object.entries(error.errors).length !== 1) errM += value + ", ";
      else errM = value.properties.message;
    }
    return res.status(400).send(errM);
  }
};

//Update existing Product
exports.product_update = async (req, res) => {
  try {
    const updatedRecord = await Product.findByIdAndUpdate(
      req.params.id,
      req.body,
      {
        new: true,
      }
    );
    if (!updatedRecord) {
      return res.status(404).send();
    }
    res.status(200).send({ success: true, record: updatedRecord });
  } catch (error) {
    let errM = "";
    for (const [key, value] of Object.entries(error.errors)) {

```

```

        if (Object.entries(error.errors).length !== 1) errM += value + ", ";
        else errM = value.properties.message;
    }
    return res.status(400).send(errM);
}
};

//Hide product
exports.product_hide = async (req, res) => {
    try {
        const product = await Product.findById(req.params.id);
        if (!product) {
            return res.status(404).send();
        }
        if (typeof req.body.isHidden === "boolean") {
            product.isHidden = req.body.isHidden;
        }
        await product.save();
        res.status(200).send(await product.populate('genres'));
    } catch (error) {
        res.status(500).send(error.message);
    }
}
};

```

ДОДАТОК Б

API продуктів

```

export const fetchRecords = () => API.get(`/products`);
export const fetchFilteredRecords = (searchQuery) =>
    API.get(
        `/products/search?search=${searchQuery.search}&genres=${
            searchQuery.selectedGenres || "none"
        }&availability=${searchQuery.availability || "none"}&isColored=${
            searchQuery.isColored || "none"
        }&prices=${searchQuery.prices || "none"}&sort=${searchQuery.sort || "none"}`
    );
export const fetchRecord = (id) => API.get(`/products/${id}`);
export const createRecord = (newRecord) =>
    API.post(`/products/create`, newRecord);
export const updateRecord = (id, updatedRecord) =>
    API.patch(`/products/update/${id}`, updatedRecord);
export const hideRecord = (id, isHidden) => API.patch(`/products/hide/${id}`,
isHidden);

```

Actions-функції продуктів

```

import * as api from "../api";

export const getRecords = () => async (dispatch) => {
    try {
        const { data: {records} } = await api.fetchRecords();
    }
}

```

```

    dispatch({ type: "FETCH_RECORDS", payload: records });
    dispatch({ type: "FETCH_ALL_RECORDS", payload: records });
  } catch (error) {
    console.log(error.message);
  }
};

export const getFilteredRecords = (searchQuery) => async (dispatch) => {
  try {
    const { data: {records} } = await api.fetchFilteredRecords(searchQuery);
    const { data } = await api.fetchRecords();

    dispatch({ type: "FETCH_FILTERED_RECORDS", payload: records });
    dispatch({ type: "FETCH_ALL_RECORDS", payload: data.records });
  } catch (error) {
    console.log(error.message);
  }
};

export const getRecord = (id) => async (dispatch) => {
  try {
    const { data } = await api.fetchRecord(id);
    dispatch({ type: "FETCH_RECORD", payload: data });
  } catch (error) {
    console.log(error.message);
  }
};

export const createRecord = (record, navigate) => async () => {
  try {
    await api.createRecord(record);
    navigate("/admin/records");
  } catch (error) {
    return error.response.data;
  }
};

export const updateRecord = (id, record, navigate) => async () => {
  try {
    await api.updateRecord(id, record);
    navigate("/admin/records");
  } catch (error) {
    return error.response.data;
  }
};

export const hideProduct = (id, isHidden) => async (dispatch) => {
  try {
    const { data } = await api.hideRecord(id, {isHidden});
    dispatch({ type: "HIDE_RECORD", payload: data });
  }

```

```

} catch (error) {
  console.log(error.message);
}
};

```

Форма для створення та редагування продуктів

```

import { useState, useEffect } from "react";
import { useDispatch, useSelector } from "react-redux";
import { useParams, useNavigate, Link } from "react-router-dom";
import {
  Button,
  Typography,
  Paper,
  Container,
  MenuItem,
  Select,
  FormControl,
  InputLabel,
  Box,
  Chip,
  Grid,
  OutlinedInput,
  Stack,
  ImageList,
  ImageListItem,
  ImageListItemBar,
  IconButton,
  FormControlLabel,
  Switch,
} from "@mui/material";

import ClearIcon from "@mui/icons-material/Clear";
import { deepPurple } from "@mui/material/colors";

import Input from "../../components/Input";
import UploadWidget from "../../components/uploadWidget";
import SnackbarError from "../../components/Snackbar";

import { createRecord, updateRecord, getRecord } from "../../actions/records";
import { getGenres } from "../../actions/genres";

const ITEM_HEIGHT = 48;
const ITEM_PADDING_TOP = 8;

const MenuProps = {
  PaperProps: {
    style: {
      maxHeight: ITEM_HEIGHT * 4.5 + ITEM_PADDING_TOP,
      width: 250,
    },
  },
};

```

```

    },
  };

const initialState = {
  name: "",
  author: "",
  description: "",
  price: 0,
  recordLength: 1,
  label: "",
  images: [],
  songList: [],
  itemsAvailable: 1,
  recordsNumber: 1,
  isColored: false,
  releaseDate: new Date().toISOString().split("T")[0],
  productionDate: new Date().toISOString().split("T")[0],
  genres: [],
  country: "",
};

const RecordForm = () => {
  const [formData, setFormData] = useState(initialState);
  const [imagesList, setImagesList] = useState([]);
  const dispatch = useDispatch();
  const navigate = useNavigate();
  const { id } = useParams();
  const { genres } = useSelector((state) => state.genres);
  const record = useSelector((state) => state.records.record);
  const [error, setError] = useState({ open: false, message: "" });
  const [checked1, setChecked1] = useState(formData.isColored);

  const [countriesList, setCountriesList] = useState([]);
  const countriesURL = "https://restcountries.com/v3.1/all?fields=name,flags";

  useEffect(() => {
    fetch(countriesURL)
      .then((response) => response.json())
      .then((responseJson) =>
        setCountriesList(
          responseJson.sort((a, b) =>
            a.name.official.localeCompare(b.name.official)
          )
        )
      )
      .catch((error) => {
        console.error(error);
      });
  }, [countriesURL]);

```

```

useEffect(() => {
  if (id) dispatch(getRecord(id));
}, [id]);

useEffect(() => {
  dispatch(getGenres());
}, [dispatch]);

useEffect(() => {
  if (record) {
    setFormData({
      ...record,
      songList: record.songList.join("\n"),
      genres: record.genres.map((genre) => genre._id),
    });
    setImagesList(record.images);
    setChecked1(record.isColored);
  }
}, [record]);

const handleSubmit = async (e) => {
  e.preventDefault();
  formData.songList = formData.songList.split("\n");
  formData.isColored = checked1;
  if (imagesList.length > 0) formData.images = imagesList;
  if (id) {
    setError({
      message: await dispatch(updateRecord(id, formData, navigate)),
      open: true,
    });
  } else {
    setError({
      message: await dispatch(createRecord(formData, navigate)),
      open: true,
    });
  }
};

const handleChange = (e) => {
  setFormData({ ...formData, [e.target.name]: e.target.value });
};

const handleOnUpload = (error, result, widget) => {
  if (error) {
    console.log(error);
    return;
  } else {
    setImagesList((prevArray) => [...prevArray, result?.info?.secure_url]);
  }
};

```

```

const handleIsColoredChange = () => {
  setChecked1((prevState) => !prevState);
};

return (
  <Container sx={{ marginTop: "10px" }} maxWidth="xl">
    <Typography variant="h4">
      {!id ? "Додати платівку" : "Оновити платівку"}
    </Typography>
    <form autoComplete="off" onSubmit={handleSubmit}>
      <SnackbarError
        open={error.open}
        message={error.message}
        onClose={() => {
          setError({ open: false, message: "" });
        }}
      />
      <Paper
        elevation={3}
        sx={{
          p: 2,
          textAlign: "left",
          marginTop: "10px",
          marginBottom: "10px",
          borderRadius: "16px",
        }}
      >
        <Grid container spacing={2}>
          <Grid item xs={12} sm={4} md={4}>
            <Typography variant="h5">Загальна інформація</Typography>
          </Grid>
          <Grid item xs={12} sm={8} md={8}>
            <Input
              name="name"
              label="Назва платівки"
              handleChange={handleChange}
              type="text"
              value={formData.name}
            />
            <Input
              name="author"
              label="Автор платівки"
              handleChange={handleChange}
              type="text"
              value={formData.author}
            />
            <Input
              name="price"
              label="Ціна"
              handleChange={handleChange}

```

```

type="text"
      value={formData.price}
    />
    <FormControl fullWidth sx={{ mb: 2 }}>
      <InputLabel>Жанри</InputLabel>
      <Select
        name="genres"
        onChange={handleChange}
        multiple
        value={formData.genres}
        required
        input={
          <OutlinedInput id="select-multiple-chip" label="Жанри" />
        }
        renderValue={(selected) => (
          <Box sx={{ display: "flex", flexWrap: "wrap", gap: 0.5 }}>
            {selected.map((value) => (
              <Chip
                key={value}
                sx={{
                  mr: 1,
                  bgcolor: deepPurple[100],
                  color: deepPurple[800],
                }}
                label={
                  genres.find((item) => item._id === value)?.name
                }
              />
            ))}
          </Box>
        )}
        MenuProps={MenuProps}
      >
        {genres.map((genre) => (
          <MenuItem value={genre._id} key={genre._id}>
            {genre.name}
          </MenuItem>
        ))}
      </Select>
    </FormControl>

    <Input
      name="itemsAvailable"
      label="Кількість на складі"
      handleChange={handleChange}
      type="text"
      value={formData.itemsAvailable}
    />

  </Grid>
</Grid>

```

```

</Paper>
<Paper
  elevation={3}
  sx={{
    p: 2,
    textAlign: "left",
    marginTop: "10px",
    marginBottom: "10px",
    borderRadius: "16px",
  }}
>
  <Grid container spacing={2}>
    <Grid item xs={12} sm={4} md={4}>
      <Typography variant="h5">Додаткова інформація</Typography>
    </Grid>
    <Grid item xs={12} sm={8} md={8}>
      <Input
        name="description"
        label="Опис платівки"
        handleChange={handleChange}
        type="text"
        multiline={true}
        rows={5}
        value={formData.description}
      />
      <Input
        name="songList"
        label="Треки, що містяться на даній платівці(один трек в одному
рядку)"
        handleChange={handleChange}
        type="text"
        multiline={true}
        rows={5}
        value={formData.songList}
      />
      <Input
        name="recordLength"
        label="Час програвання платівки"
        handleChange={handleChange}
        type="text"
        value={formData.recordLength}
      />
      <Input
        name="label"
        label="Лейбл"
        handleChange={handleChange}
        type="text"
        value={formData.label}
      />
      <Input

```

```

name="recordsNumber"
  label="Кількість платівок у комплекті"
  handleChange={handleChange}
  type="text"
  value={formData.recordsNumber}
/>
<Input
  name="releaseDate"
  label="Дата випуску альбому"
  handleChange={handleChange}
  type="date"
  value={
    new Date(formData.releaseDate).toISOString().split("T")[0]
  }
/>
<Input
  name="productionDate"
  label="Дата випуску платівки"
  handleChange={handleChange}
  type="date"
  value={
    new Date(formData.productionDate).toISOString().split("T")[0]
  }
/>

<FormControl fullWidth sx={{ mb: 2 }}>
  <InputLabel>Країна</InputLabel>
  <Select
    name="country"
    onChange={handleChange}
    value={formData.country}
    required
    input={
      <OutlinedInput id="select-multiple-chip" label="Країна" />
    }
  >
    {countriesList.map((country) => (
      <MenuItem
        value={country.name.official}
        key={country.name.official}
      >
        <img
          src={country.flags.png}
          alt={country.name.official}
          width={30}
        />
        <Typography variant="span" sx={{ ml: 1 }}>
          {country.name.official}
        </Typography>
      </MenuItem>
    ))}
  </Select>
</FormControl>

```

```

    )))
      </Select>
    </FormControl>
    <FormControllabel
      sx={{ mb: 2 }}
      control={
        <Switch checked={checked1} onChange={handleisColoredChange} />
      }
      label="Кольорова?"
    />
    {imagesList.length > 0 && (
      <ImageList
        sx={{ width: 700, height: 180, mx: "auto" }}
        cols={4}
        rowHeight={179}
      >
        {imagesList.map((image, i) => (
          <ImageListItem key={image}>
            <img
              src={`_${image}_`}
              srcSet={`_${image}_`}
              alt={"Зображення " + i}
              loading="lazy"
            />
            <ImageListItemBar
              sx={{
                background:
                  "linear-gradient(to bottom, rgba(0,0,0,0.7) 0%, " +
                  "rgba(0,0,0,0.3) 70%, rgba(0,0,0,0) 100%)",
              }}
              position="top"
              actionIcon={
                <IconButton
                  sx={{ color: "white" }}
                  onClick={() =>
                    setImagesList((prev) =>
                      prev.filter((item) => item !== image)
                    )
                  }
                />
              }
              <ClearIcon />
              </IconButton>
            }
            actionPosition="right"
          />
        </ImageListItem>
      )
    )
  )
}
</ImageList>
))
</div>

```

```

<UploadWidget onUpload={handleOnUpload} folder={"Records"}>
  ({ open }) => {
    function handleClick(e) {
      e.preventDefault();
      open();
    }
    return (
      <Button
        variant="contained"
        fullWidth
        onClick={handleOnClick}
      >
        Додати зображення
      </Button>
    );
  }
</UploadWidget>
</div>
</Grid>
</Grid>
</Paper>
<Stack spacing={2} direction="row" justifyContent="flex-end">
  <Link to="/admin/records">
    <Button variant="text" color="primary">
      Відмінити
    </Button>
  </Link>
  <Button type="submit" variant="contained" color="primary">
    {!id ? "Додати платівку" : "Оновити платівку"}
  </Button>
</Stack>
</form>
</Container>
);
};

export default RecordForm;

```

ДОДАТОК В

Сторінка кошику

```

import { useState, useEffect } from "react";
import { useDispatch, useSelector } from "react-redux";
import { Link } from "react-router-dom";

import { Container, Stack, Button, Typography } from "@mui/material";
import ShoppingCartOutlinedIcon from "@mui/icons-material/ShoppingCartOutlined";

import CartRow from "../components/CartRow";

```

					ІПЗ.КР.Б – 121 – 23 – ІІЗ	79

```

import { getCartRecords } from "../actions/records";

const Cart = () => {
  const dispatch = useDispatch();
  let cartRecords = useSelector((state) => state.records.cartRecords);
  const [totalPrice, setTotalPrice] = useState(0);

  const handleDeleteClick = (id, price) => {
    let cart = JSON.parse(localStorage.getItem("cart"));
    localStorage.setItem(
      "cart",
      JSON.stringify(cart.filter((e) => e.id !== id))
    );
    changeTotalPrice(price);
    dispatch(
      getCartRecords(
        JSON.parse(localStorage.getItem("cart")).map(({ id }) => id)
      )
    );
  };

  const changeTotalPrice = (price) => {
    setTotalPrice((prev) => prev + price);
  };

  useEffect(() => {
    if (JSON.parse(localStorage.getItem("cart")))
      dispatch(
        getCartRecords(
          JSON.parse(localStorage.getItem("cart")).map(({ id }) => id)
        )
      );
  }, [dispatch]);

  return (
    <Container sx={{ height: "100vh", marginTop: "10px" }}>
      <Typography variant="h4">Кошик</Typography>
      {cartRecords && cartRecords.length === 0 ? (
        <Typography variant="h5">Кошик порожній</Typography>
      ) : (
        cartRecords &&
        cartRecords.map((record) => (
          <CartRow
            key={record._id}
            counterP={
              JSON.parse(localStorage.getItem("cart")).find(
                (e) => e.id === record._id
              )
            }
          />
        ))
      )
    }
  );

```

```

(e) => e.id === record._id
      ).count
      : 1
    }
    record={record}
    handleDeleteClick={handleDeleteClick}
    changeTotalPrice={changeTotalPrice}
  />
))
}}
{cartRecords && cartRecords.length !== 0 && (
  <Stack
    direction="row"
    spacing={2}
    justifyContent="flex-end"
    alignItems="center"
  >
    <Typography variant="h5">Загальна вартість: {totalPrice}</Typography>
    <Link
      to="/order"
      state={{
        totalPrice: totalPrice,
        records: JSON.parse(localStorage.getItem("cart")),
      }}
    >
      <Button
        size="large"
        variant="contained"
        startIcon={<ShoppingCartOutlinedIcon />}
      >
        Оформити замовлення
      </Button>
    </Link>
  </Stack>
)}
</Container>
);
};

export default Cart;

```

Елемент кошику

```

import { useEffect, useState } from "react";

import { Stack, Typography, Paper, IconButton, Divider } from "@mui/material";
import DeleteIcon from "@mui/icons-material/Delete";
import AddIcon from "@mui/icons-material/Add";
import RemoveIcon from "@mui/icons-material/Remove";

import { ButtonStyle } from "../StyledComponents/CompStyles";

```

					ІІЗ.КР.Б – 121 – 23 – ІІЗ	81

```

const CartRow = (props) => {
  const { record, handleDeleteClick, counterP, changeTotalPrice } = props;
  const [counter, setCounter] = useState(counterP);

  const handleClickPlus = () => {
    if (counter < record.itemsAvailable) {
      setCounter((prev) => prev + 1);
      changeTotalPrice(record.price);
      let cart = JSON.parse(localStorage.getItem("cart"));
      cart.forEach(element => {
        if (element.id === record._id)
          element.count = counter + 1
      });
      localStorage.setItem("cart", JSON.stringify(cart));
    }
  };

  const handleClickMinus = () => {
    if (counter > 1) {
      setCounter((prev) => prev - 1);
      changeTotalPrice(-record.price);
      let cart = JSON.parse(localStorage.getItem("cart"));
      cart.forEach(element => {
        if (element.id === record._id)
          element.count = counter - 1
      });
      localStorage.setItem("cart", JSON.stringify(cart));
    }
  };

  useEffect(() => {
    changeTotalPrice((counter * record.price)/2);
  }, []);

  return (
    <>
      <Paper
        elevation={2}
        sx={{
          p: 2,
          textAlign: "left",
          marginTop: "10px",
          marginBottom: "10px",
          borderRadius: "16px",
        }}
      >
        <Stack
          direction="row"
          justifyContent="space-between"
          spacing={3}

```

```

sx={{ mt: 2 }}
>
<Stack
  direction="row"
  justifyContent="space-between"
  alignItems="flex-start"
  spacing={2}
>
  <div style={{ padding: 2 }}>
    <img
      height={175}
      width={175}
      style={{ objectFit: "cover" }}
      src={record.images[0]}
      alt={record.name + "зображення"}
    />
  </div>
  <Stack direction="column" spacing={2}>
    <Typography variant="h5">{record.name}</Typography>
    <Typography variant="h6">{record.author}</Typography>
    <Typography variant="h6">{record.price}</Typography>
  </Stack>
</Stack>
<Stack
  direction="column"
  justifyContent="space-between"
  alignItems="center"
  spacing={2}
>
  <IconButton
    sx={ButtonStyle}
    onClick={() => handleDeleteClick(record._id, -
(record.price*counter))}
  >
    <DeleteIcon />
  </IconButton>
  <Stack
    direction="row"
    alignItems="center"
    justifyContent="space-between"
  >
    <IconButton
      sx={ButtonStyle}
      onClick={handleClickMinus}
      disabled={counter === 1 ? true : false}
    >
      <RemoveIcon />
    </IconButton>
    <Typography variant="h6">{counter}</Typography>
    <IconButton

```

```

sx={ButtonStyle}
    onClick={handleClickPlus}
    disabled={counter === record.itemsAvailable ? true : false}
  >
    <AddIcon />
  </IconButton>
</Stack>
<Typography variant="h6">
  <Divider />
  {record.price * counter}₺
</Typography>
</Stack>
</Stack>
</Paper>
</>
);
};

export default CartRow;

```