

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» — користувачі, замовлення та
новини»**

Виконала студентка групи ІПЗ-19-2
ЛЕЩИШИНА Вікторія Олегівна

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: **ЛЕЩИШИНА Вікторія Олегівна**

Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**

Тема роботи: **«Веб-додаток магазину вінілових платівок «Musical Corner» — користувачі, замовлення та новини»,**

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

Вихідні дані для роботи: розробка веб-додатку, який дозволить переглядати вінілові платівки, здійснювати їх купівлю і преглядати останні новини світу музики, а також розробка системи адміністрування даного веб-додатку, а саме частини користувачів, замовлень та новин; методи та засоби, які використовуються для 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 – 15 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» користувачів, замовлень та новин та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи містить 86 сторінок, 62 ілюстрації та 2 таблиці.

Метою роботи є розробка веб-додатку, завдяки якому можна купувати вінілові платівки, переглядати останні новини світу музики та ділитися своїми думками щодо 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, а також опис об'єктної структури веб-додатку. Крім того, наведено 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» користувачі, замовлення та новини Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.		В. О. Лещишина						
Керівник		Ю.М. Єфремов					4	86
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 users, orders and news, and an explanatory note. The explanatory note to the qualification work contains 86 pages, 62 illustrations and 2 tables.

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

The paper defines the main tasks for the development of the system, analyzes analogs of the developed system. The paper presents the scenarios of the web application, its general structure, a description of the algorithms of interaction of individual modules of the system, as well as a description of the object structure of the web application. In addition, the results of testing the application are presented.

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

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

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	7
ВСТУП.....	8
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.	13
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.	20
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ	21
2.1. ВИЗНАЧЕННЯ ВАРІАНТІВ ВИКОРИСТАННЯ ТА СТРУКТУРИ СИСТЕМИ.	21
2.2. РОЗРОБКА СТРУКТУРИ ФАЙЛІВ СИСТЕМИ.	25
2.3. РОЗРОБКА СТРУКТУРИ БАЗИ ДАНИХ.....	27
2.4. РЕАЛІЗАЦІЯ ФУНКЦІОНАЛУ ВЕБ-ДОДАТКУ.	30
Висновки до розділу 2.	50
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ДОДАТКОМ	51
3.1. ПОРЯДОК ВСТАНОВЛЕННЯ ТА НАЛАШТУВАННЯ СИСТЕМИ.....	51
3.2. СТРУКТУРА ІНТЕРФЕЙСУ. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ.	51
3.3. ТЕСТУВАННЯ РОБОТИ СИСТЕМИ.	66
Висновки до розділу 3.	67
ВИСНОВКИ.....	68
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	70
ДОДАТКИ	72
ДОДАТОК А	73
ДОДАТОК Б	77

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CRUD – create, read, update, delete (створення, читання, оновлення, видалення).

JSON – JavaScript Object Notation (формат збереження даних).

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

ODM – Object Document Mapping.

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 (бібліотека React UI компонентів, яка реалізує Material Design від Google).

ВСТУП

Протягом останніх років спостерігається значне зростання популярності вінілових платівок серед музичних ентузіастів і колекціонерів [1]. Багато людей, особливо меломанів, цінують автентичний звук, який надає вініл, що додає особливого характеру музиці. Крім того, такі платівки вважаються більш довговічними і мають більший розмір пакування, що допомагає побачити більше деталей на обкладинці альбому. Зі зростанням попиту на вінілові платівки різні компанії почали випускати не лише альбоми на вінілі різних музикальних гуртів, а й саундтреки до фільмів, серіалів і відеоігор. Тому, створення інтернет-магазину вінілових платівок є актуальним, оскільки це полегшить клієнтам процес вибору та купівлі бажаних вінілових платівок, а додаткове надання можливості перегляду останніх новин музики принесе задоволення не лише меломанам, а й початківцям у світі музики.

Метою даної роботи є розробка веб-додатку, який дозволить переглядати вінілові платівки, здійснювати їх купівлю і переглядати останні новини світу музики, а також розробка системи адміністрування даного веб-додатку, а саме частини користувачів, замовлень та новин.

Для досягнення описаної мети, необхідно виконати наступні завдання:

- провести аналіз теоретичних та функціональних потреб додатку;
- вивчити конкурентів та визначити унікальність додатку;
- визначити структуру та архітектуру додатку;
- обрати необхідні засоби для реалізації додатку;
- здійснити проектування та реалізацію додатку.

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

Предметом дослідження є застосування 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 13 найменувань, 2 додатків, 2 таблиць та 62 ілюстрацій по тексту. Повний обсяг роботи становить 86 сторінок, з них 71 сторінок основного тексту.

					ІПЗ.КР.Б – 121 – 23 – ПЗ	
						9

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

1.1. Постановка задачі.

Головною метою розробки веб-додатку магазину вінілових платівок «Musical Corner» є перегляд та придбання музикальних платівок. Цей додаток принесе користь від використання для широкого спектру користувачів, включаючи як любителів музики і колекціонерів, так і тих, хто тільки починає відкривати для себе цей світ.

У кваліфікаційній роботі необхідно розробити веб-додаток, який дозволить переглядати та купувати платівки і матиме адаптивний дизайн.

Основними етапами реалізації цього завдання є:

- створення серверної частини додатку:
 - проєктування бази даних;
 - створення необхідних моделей бази даних з відповідними схемами:
 - модель користувача;
 - модель замовлень;
 - модель новин;
 - модель коментарів до новин.
 - створення необхідних контролерів для взаємодії з базою даних:
 - CRUD, фільтрація, сортування, пошук користувачів;
 - CRUD, фільтрація, сортування, пошук замовлень;
 - CRUD, фільтрація, сортування, пошук новин;
 - CRUD коментарів до новин.
 - створення необхідних шляхів для взаємодії з сервером:
 - шляхи для користувачів;
 - шляхи для замовлень;
 - шляхи для новин;
 - шляхи для коментарів до новин;

- тестування серверної частини додатку.
- Розробка дизайну адміністративної частини додатку:
 - створення загальної концепції дизайну;
 - верстання елементів інтерфейсу адміністративної частини на основі React Material UI:
 - навігаційна панель;
 - темна та світла теми;
 - відображення всіх користувачів, замовлень, новин;
 - відображення одного замовлення та однієї новини;
 - форма для редагування замовлень;
 - форма для додавання і редагування новин;
 - компонент з фільтрацією, сортування, пошуком.
- Розробка адміністративної частини веб-додатку:
 - розробка API для забезпечення взаємодії з серверною частиною додатку;
 - розробка необхідних дій для взаємодії системи з API;
 - розробка маршрутизації системи;
 - розробка необхідних компонентів та сторінок веб-додатку;
 - підключення сторонніх API та бібліотек:
 - підключення віджету Cloudinary для збереження зображень в хмарному сервісі;
 - підключення TinyMCE для зручного створення самого тексту новини;
 - тестування адміністративної частини системи.
- Розробка дизайну клієнтської частини додатку:
 - створення загальної концепції дизайну;
 - Верстання елементів інтерфейсу клієнтської частини на основі React Material UI:
 - форма для входу і реєстрації;
 - відображення інформації авторизованого користувача в кабінеті:

- відображення загальної інформації користувача;
- відображення історії замовлень;
- відображення списку «Бажане».
- відображення списку «Бажане» неавторизованого користувача;
- форма для створення замовлення;
- компонент з подякою за створене замовлення;
- відображення списку новин;
- відображення однієї новини;
- головна сторінка:
 - відображення останніх доданих товарів;
 - відображення товарів, які можуть зацікавити користувача;
 - відображення останніх доданих новин.
- компонент з фільтрацією, сортування, пошуком.
- Розробка клієнтської частини веб-додатку:
 - розробка API для забезпечення взаємодії з серверною частиною;
 - розробка необхідних дій для взаємодії системи з API;
 - розробка маршрутизації додатку;
 - розробка необхідних компонентів та сторінок веб-додатку;
 - підключення сторонніх API для роботи додатку;
 - підключення API Нової Пошти для обрання адреси доставки;
 - підключення API LiqPay для проведення оплати замовлення;
 - тестування клієнтської частини додатку.

В результаті виконання поставленого завдання буде отримано частину серверу, яка дозволить роботу з даними на адміністративній та клієнтській частинах веб-додатку, які будуть доступні для користування на різних пристроях. Авторизований працівник в адміністративній частині додатку зможе переглядати загальну інформацію про користувача, дозволяти чи забороняти коментування, переглядати інформацію про всі та одне замовлення, редагувати інформацію про замовлення, переглядати список новин, додавати нові і редагувати старі.

					ІПЗ.КР.Б – 121 – 23 – ПЗ	
						12

Неавторизовані користувачі на клієнтській частині матимуть змогу додавати товари в список «Бажане», оформлювати та сплачувати замовлення, переглядати новини, реєструватися на сайті. Авторизовані користувачі додатково зможуть переглядати власну інформацію у своєму кабінеті, таку як загальна інформація, історія замовлень і список бажаного, а також додавати коментарі до новин і редагувати або ж видаляти їх.

Як в адміністративній частині, так і в клієнтській буде доступний пошук, фільтрація і сортування даних.

1.2. Аналіз аналогів програмного продукту.

Зі зростанням популярності вінілових платівок зростає і кількість магазинів, фокусом яких є їх продаж. Більшість цих магазинів мають власні веб-додатки для перегляду платівок та їх замовлення. Аналіз існуючих аналогів дозволить визначити основні функції, які повинні бути присутні у системі, що розробляється, а також визначити елементи, завдяки яким веб-додаток зможе створити унікальну пропозицію для клієнтів.

Розглянемо наявні аналоги інтернет-магазину вінілових платівок.

VinylClub – інтернет-магазин платівок. Магазин має досить великий асортимент не лише платівок, а й аксесуарів до них, включаючи програвачі. Наявний блог, реєстрація та список «Улюблене». З переваг можна назвати чистий дизайн, можливість оформити доставку як «Новою Поштою», так і доставкою до дверей. З мінусів можливість лише фільтрації за датами для блогу, реєстрація вимагає введення номеру телефону. Головна сторінка сайту зображена на рис. 1.1.

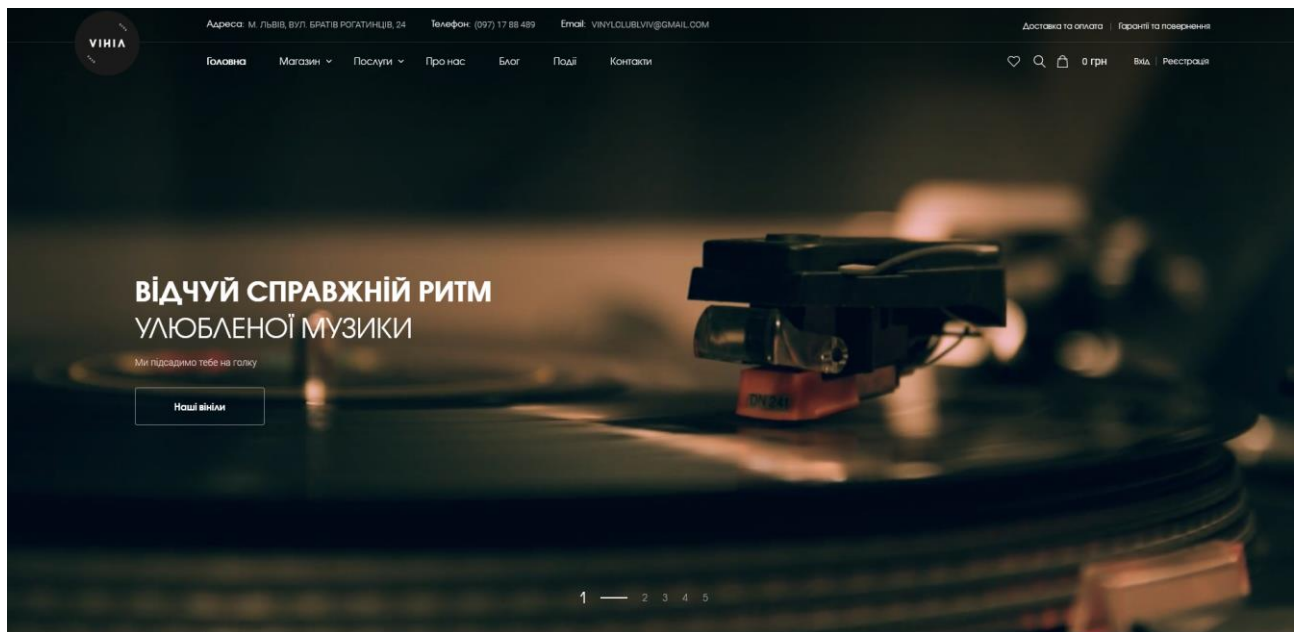


Рис. 1.1 – інтернет-магазин VinylClub.

The Sound of Vinyl – інтернет-магазин вінілових платівок. Достатній асортимент як платівок нових гуртів, так і класичних. Є можливість купівлі аксесуарів, а самі чистячих інструментів та наборів. Основні переваги: легкий у розумінні інтерфейс, швидка реєстрація. Недоліки: відсутність списку «Бажане», відсутність одразу наявної інформації про вартість доставки, відсутність блогу. Головна сторінка сайту зображена на рис. 1.2.

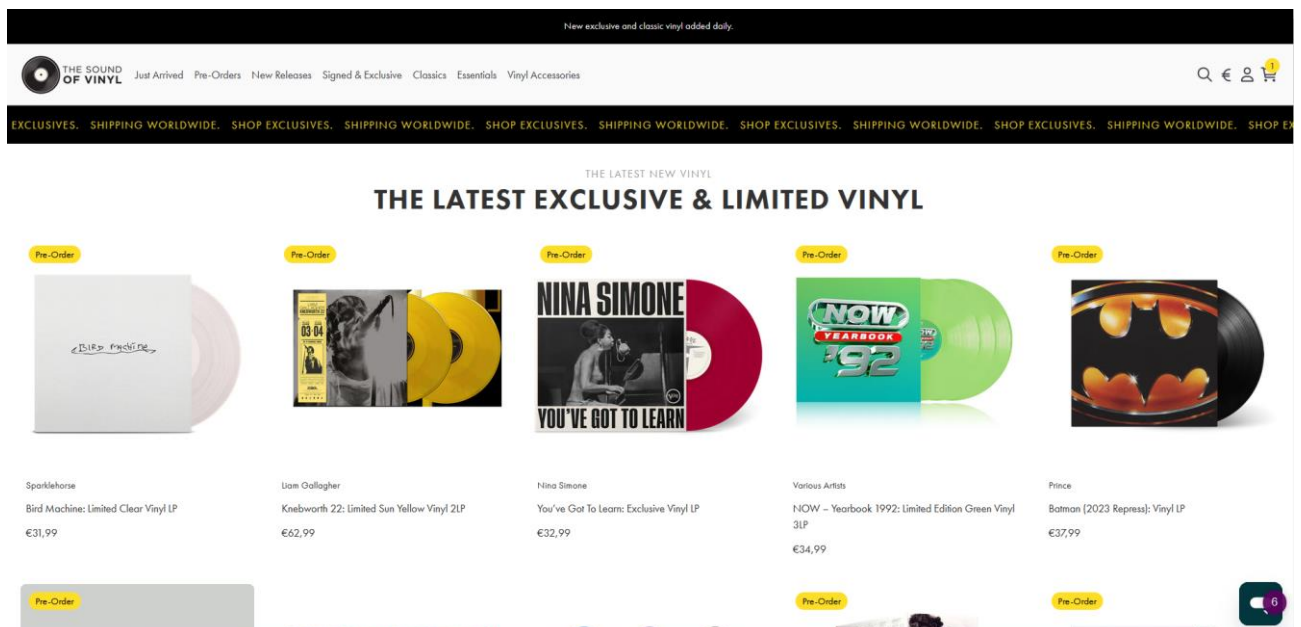


Рис. 1.2 – інтернет-магазин The Sound Of Vinyl

RecordSale – інтернет магазин вінілу, де можна придбати вже бувші у вжитку платівки. Серед переваг можна назвати оригінальний дизайн сайту і

					ІПЗ.КР.Б – 121 – 23 – ІЗ	
						14

велику кількість товарів. Головні недоліки – це відсутня можливість перегляду списку «Бажаного» (на сторінці товару є кнопка для додавання товару у такий список, проте самої сторінки або елемента з ним немає). Наявні записи з короткими новинами, проте при натисканні на них відкривається сторінка з платівками певного виконавця, відсутнє сортування або будь-яка фільтрація новин. Відсутня можливість перегляду історії власних замовлень у кабінеті. Головна сторінка сайту зображена на рис. 1.3.

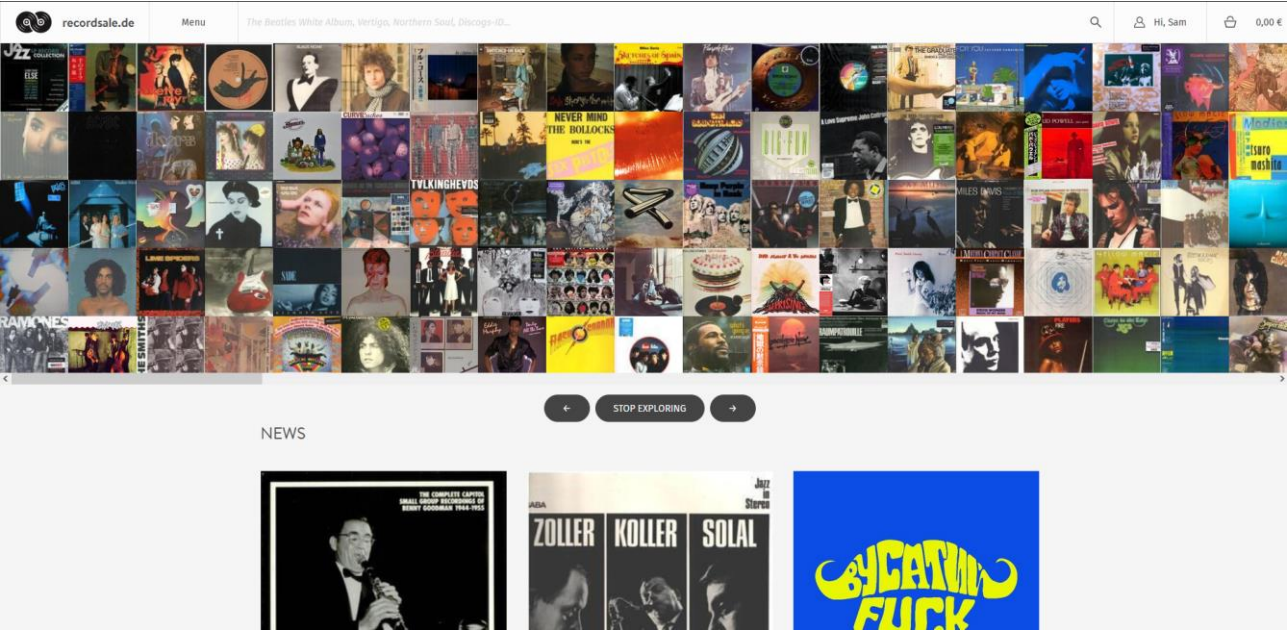


Рис. 1.3 – інтернет-магазин RecordSale.

Отже, порівняння за критеріями наведене в таблиці 1.1.

Таблиця 1.1

	Реєстрація	Кабінет	Список «Бажане»	Блог	Історія замовлень
VinylClub	проблемна	Так	Так	Так	Невідомо
The Sound of Vinyl	швидка	Так	Ні	Ні	Так
RecordSale	швидка	Так	Ні	Так	Ні

Після проведення аналізу існуючих аналогічних веб-додатків можна зробити висновок, що кожен з них має власні переваги та недоліки. Це дає

можливість користувачам обирати, де саме вони бажають здійснювати власні покупки.

Зважаючи на інформацію, отриману в ході аналізу аналогів, виникає необхідність у розробці веб-додатку з більш спеціалізованою функціональністю.

Отже, особливостями даної веб-системи будуть: лаконічний дизайн, наявність списку «Бажане», швидка реєстрація, перегляд історії замовлень, наявність блогу новин з широкою фільтрацією.

1.3. Вибір архітектури веб-додатку «Musical Corner».

З метою створення веб-додатку можливо використати наступні архітектурні шаблони програмного забезпечення:

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

Найкраще для розробки інтернет-магазину підходять такі шаблони як клієнт-сервер або MVC. Однак, для оптимальної реалізації системи доцільно буде використати шаблон клієнт-сервер.

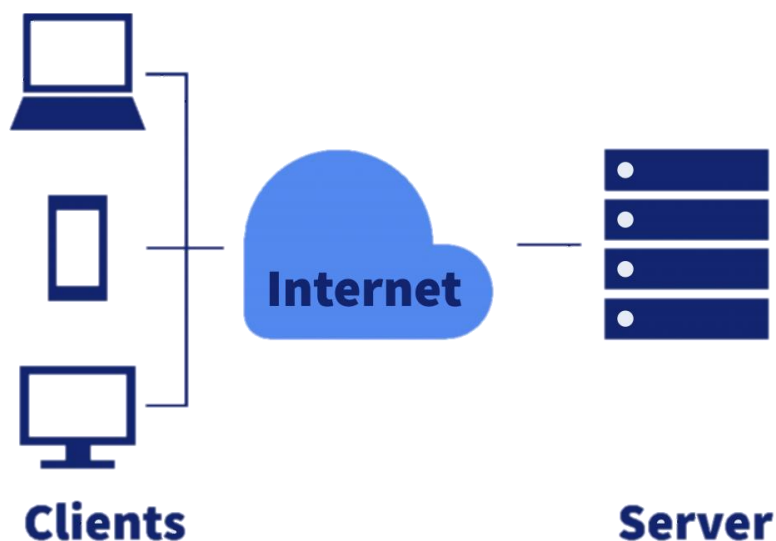


Рис. 1.4. Схема шаблону клієнт-сервер

Архітектурний шаблон клієнт-сервер є одним з найпоширеніших у розробці програмного забезпечення. Цей шаблон використовується для створення систем, де взаємодія між користувачем і сервером вважається ключовою.

Система, яка має клієнт-серверну архітектуру, складається з двох основних компонентів:

- клієнт: програмний компонент, який насамперед і взаємодіє з користувачем. Він може бути представлений веб-браузером, мобільним додатком або настільною програмою. Основна функція цього компоненту полягає у відображенні інтерфейсу користувача та збору введених даних. Вона надсилає запити до сервера і отримує відповіді для подальшого відображення результатів користувачу;

- сервер: центральний компонент системи, який здійснює обробку запитів від клієнта і забезпечує надання відповідей. Сервер зберігає бізнес-логіку, забезпечує доступ до бази даних, обробку запитів та відповідей. Він може бути представлений фізичним сервером або хмарним сервісом.

Шаблон клієнт-сервер чудово підходить для розробки інтернет-магазину, оскільки:

- шаблон дозволяє чітко розділити функції та обов'язки між клієнтською і серверною частинами системи. Клієнтська сторона відповідає за відображення інтерфейсу, взаємодію з користувачем, обробку локальних даних і передачу запитів до серверу, тоді як серверна сторона забезпечує обробку бізнес-логіки, доступ до бази даних, керування даними та надання відповідей клієнту, що спрощує розробку, підтримку та масштабування системи;

- використання шаблону дозволяє забезпечити вищий рівень безпеки. Конфіденційні дані, такі як дані платежів або особисті дані користувачів, можуть бути збережені і оброблені на серверній стороні, що забезпечить кращий захист;

- шаблон дозволяє застосовувати стратегії кешування даних на клієнтській стороні для їх збереження та повторного використання, що дозволить зменшити навантаження на сервер і покращити продуктивність системи, забезпечуючи швидкі відгуки на запити користувачів.

Отже, архітектурний шаблон клієнт-сервер чудово підходить для розробки запланованого додатку. Даний підхід гарантує успішну реалізацію системи і її майбутню модифікацію.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.

При розробці веб-додатку необхідно вибрати такі інструменти, які максимально спрощують та пришвидшують процес розробки. Така вимога обмежує кількість можливих засобів розробки, скорочуючи їх різноманітність.

Для розробки веб-додатків найчастіше використовується мова програмування PHP або платформа Node.js, яка використовується для виконання високопродуктивних мережевих додатків, написаних мовою JavaScript. Зважаючи на вимоги проекту було обрано Node.js, адже він має кращі показники з продуктивності, дозволяє безперешкодно працювати як з реляційними, так і з NoSQL базами даних і ефективно і оперативно надає дані в режимі реального часу, що робить його чудовим вибором для проектів з високим навантаження [2].

Для розробки серверної частини було обрано фреймворк Express, який є один з найпопулярніших засобів для написання серверної частини системи [3].

Серед його переваг є:

- Мінімалістичність та простота у використанні: даний фреймворк пропонує досить просту архітектуру та є інтуїтивно зрозумілим, що дозволяє розробникам легко створювати веб-додатки.

- Гнучкість у виборі технологій та реалізації функціональності. Можливо використання різних модулів та бібліотек для розширення функціональності фреймворку та втілення потреб проекту, що розробляється.

– Наявність потужних засобів для маршрутизації та обробки запитів, що надає можливість легко встановлювати маршрути до різних URL-адрес, що забезпечить чітку організацію проєкту.

– Висока популярність фреймворку означає і велику спільноту, що в свою чергу дає змогу знайти багато ресурсів, документацію, плагіни та підтримку, що спрощує роботу і допоможе розв’язати потенційні проблеми.

В процесі обрання бази даних для інтернет-магазину важливо приділити увагу таким аспектам як типи даних, обсяг даних, швидкодії, а також навчикам та досвіду розробників з тією чи іншою базою даних і майбутнім потребам проєкту.

Такі бази даних як MongoDB, PostgreSQL і DynamoDB є найкращими для фреймворку Express.js.

З огляду на майбутнє розширення інтернет-магазину для продажу не тільки вінілових платівок, а й аксесуарів, таких як програвачі та засоби по догляду за платівками, варто використовувати нереляційну базу даних. Це пояснюється можливістю зберігати документи з різними полями в одній колекції, що сприяє більш гнучкому управлінню та розширенню асортименту магазину.

Для розробки бази даних веб-додатку перевага надається MongoDB, а не DynamoDB, оскільки MongoDB має гнучку структуру документів, що дозволяє зберігати дані з різними полями і структурами в одній колекції, в той час як у DynamoDB схема таблиці повинна бути наперед визначена, і зміна цієї схеми може бути складною. Багатофункціональність MongoDB також сприяє її вибору, адже вона має такі можливості як текстовий пошук, геопросторові запити, агрегації та індекси. Потужна мова запитів MongoDB (MongoDB Query Language) дозволяє складати складні запити до даних, а DynamoDB використовує простий ключ-значення та запити на основі індексів. Для роботи з MongoDB у середовищі Node.js варто використовувати бібліотеку Mongoose, яка спрощує, полегшує та надає високорівневний

інтерфейс для взаємодії з обраною базою даних, надає додатковий функціонал і зручні методи для роботи з даними.

Найпопулярнішими фреймворками серед розробників для фронтенд частини проєкту є React, Angular і Vue.

Переваги React.js включають в себе компонентний підхід, що сприяє легкому організуванню та перевикористанню коду, а також його гнучкість, яка дає можливість вибирати інші бібліотеки залежно від потреб.

Angular має свої переваги, такі як повноцінний фреймворк, вбудовані функції для маршрутизації, керування станами та тестування. Також варто відзначити його строгість, яка гарантує дотримання деяких стандартів та підходів.

У Vue.js є свої переваги, такі як легкість використання, висока швидкодія та гнучкість, що дозволяє використовувати його як фреймворк або бібліотеку в залежності від потреб проєкту.

Зважаючи на компетентність та досвід розробників, а також на розмір команди, яка складається з двох осіб, було вирішено використовувати React для розробки клієнтської частини додатку, завдяки його гнучкості.

Висновки до розділу 1.

У першому розділі були описані основні вимоги до веб-додатку магазину вінілових платівок «Musical Corner». Після аналізу існуючих додатків були визначені їх переваги та недоліки, а також виокремлені особливості, які повинні бути присутніми у майбутньому веб-додатку. В результаті аналізу інструментів розробки було прийнято рішення використовувати Node.js, Express.js як фреймворк, MongoDB як базу даних і бібліотеку Mongoose для роботи з нею, та React як бібліотеку 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

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 та купівлі вінілових платівок, а також читання інформаційного блогу з новинами світу музики.

Вимоги користувачів до частин, які забезпечують функції користувачів, замовлень і новин.

Усі користувачі клієнтської частини мають доступ до наступних функцій:

- реєстрація та автентифікація;
- додавання платівок до списку «Бажане»;
- перегляд списку «Бажане»;
- оформлення замовлення;
- проведення оплати;
- перегляд списку всіх новин;
- фільтрація, сортування та пошук новин;
- перегляд однієї новини;
- перегляд коментарів до новин від інших користувачів.

Автентифіковані користувачі клієнтської частини матимуть доступ до наступних додаткових функцій:

- перегляд власної інформації у своєму профілі;
- редагування власної інформації;

- перегляд історії замовлень;
- фільтрація, сортування та пошук серед власних замовлень;
- написання коментарів до новин;
- редагування власних коментарів;
- видалення власних коментарів.

Автентифіковані користувачі адміністративної частини матимуть доступ до наступних функцій:

- перегляд всіх замовлень;
- фільтрація, сортування та пошук замовлень;
- перегляд інформації про конкретне замовлення;
- редагування інформації про замовлення;
- перегляд всіх зареєстрованих користувачів;
- фільтрація, сортування та пошук користувачів;
- редагування їхньої можливості коментування;
- перегляд всіх новин;
- фільтрація, сортування та пошук новин;
- перегляд однієї новини;
- створення новини;
- редагування новини;
- видалення новини.

Функціональні вимоги:

- підключення сторонніх API для реалізації оплати, вибору способу доставки та оформлення текстів новин;
- наявність фільтрації, сортування та пошуку на клієнтській та адміністративній частинах;
- можливість реєстрації та автентифікації на клієнтській частині.

Нефункціональні вимоги:

• Сприйняття:

- час, потрібний для навчання роботи з додатком для звичайних

					ІПЗ.КР.Б – 121 – 23 – ПЗ	
						22

користувачів — 1 хвилини, для досвідчених — 30 хвилин;

- час відповіді системи для звичайних запитів не повинен перевищувати 3 секунди, а для більш складних запитів — 7 секунд;

- інтерфейс повинен бути інтуїтивно зручним та зрозумілим для користувача та не вимагати додаткової підготовки.

- **Продуктивність.** Система повинна підтримувати велику кількість користувачів, що одночасно працюють з додатком.

- Додаток має працювати в сучасних браузерах.

- Масштабування повинно проходити без збоїв.

Аналіз вимог користувачів та функціональних вимог до проєкту дає змогу виділити сутності, що забезпечать точну реалізацію програмного комплексу системи. Розглянемо як працівник зможе взаємодіяти з адміністративною частиною веб-додатку, а саме модулями, які відповідають за користувачів, замовлення та новини (рис. 2.1).

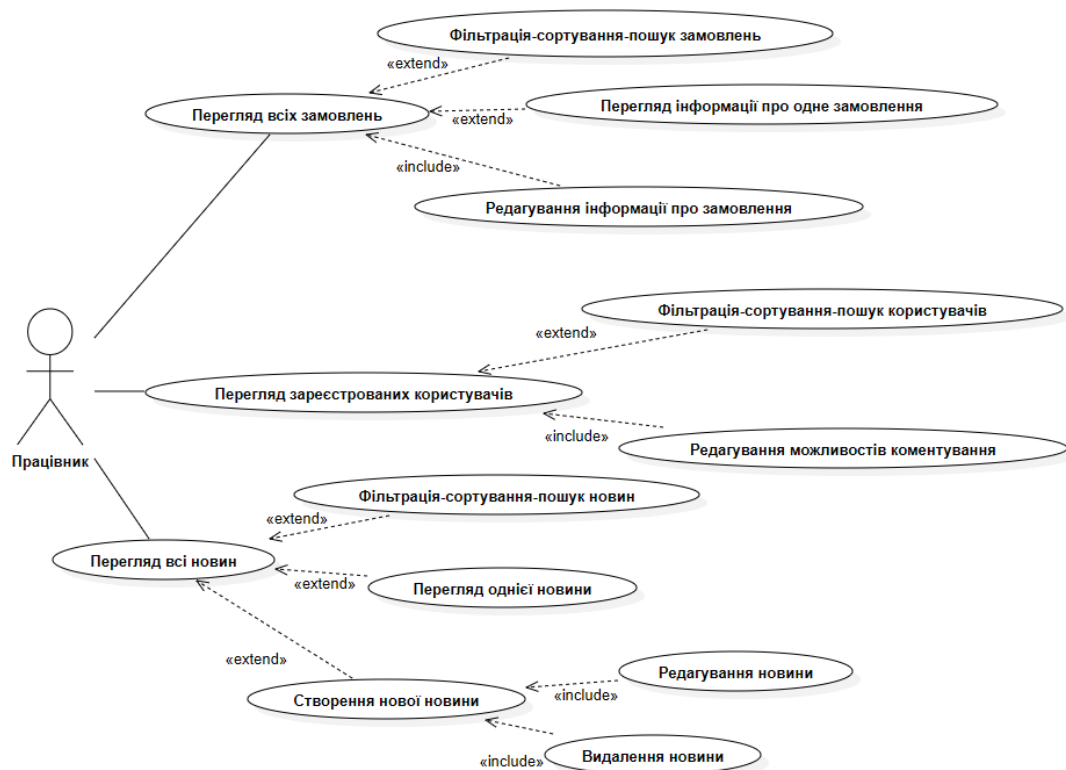


Рис. 2.1. – Варіанти використання для адміністративної частини.

Працівник зможе переглядати список усіх замовлень. Присутні такі додаткові функції як фільтрація, сортування та пошук замовлень, перегляд інформації про замовлення та редагування інформації про замовлення.

Працівник також матиме змогу переглядати інформацію про користувачів. Додатковими функціями є фільтрація, сортування та пошук серед користувачів, редагування їхньої можливості коментування.

Працівник зможе взаємодіяти з новинами, а саме проводити фільтрацію, сортування та пошук, є можливість перегляду однієї новини, також можна створити нову новину і редагувати та видалити її.

Розглянемо додатково можливості користувача і автентифікованого користувача (рис. 2.2).

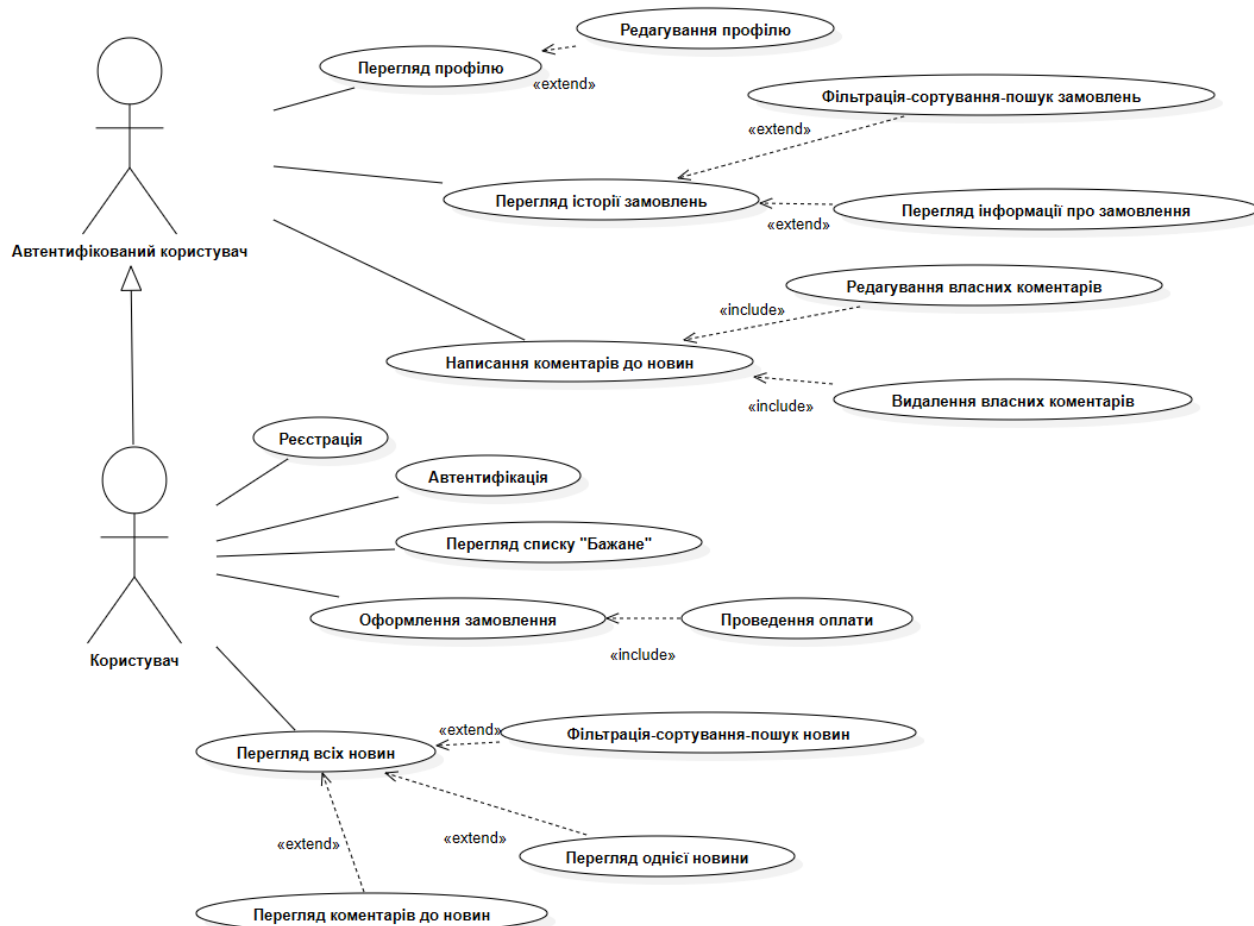


Рис. 2.2. – Варіанти використання для клієнтської частини.

Користувач зможе зареєструватися або ж автентифікуватися в системі. Є можливість перегляду списку «Бажане», оформлення замовлення, що включає в себе його оплату. Також є можливим перегляд всіх новин, також можна буде

провести їх фільтрацію, сортування або ж здійснити пошук, переглянути певну новину та коментарі до неї.

Автентифікований користувач матиме доступ до таких додаткових функцій: перегляд профілю та змогу його редагування; перегляд власної історії замовлень, а також її фільтрацію, сортування та пошук та перегляд інформації про конкретне замовлення; написання коментарів до новин, а також редагування та видалення власних коментарів.

2.2. Розробка структури файлів системи.

Загалом система розділена на серверну, клієнтську і частину адміністрування для забезпечення кращої роботи і вищого рівня безпеки.

Серверна частина системи буде організована за модульним підходом для поліпшення структури коду та забезпечення його легкості розширення. Код сервера розбивається на окремі модулі, які підключаються до серверного ядра, подібно до використання бібліотек.

Файлова структура проекту буде включати наступні компоненти:

- Моделі (models): використовуються для створення колекцій в базі даних. Моделі дозволяють визначати схеми даних та проводити операції збереження, оновлення, видалення та отримання даних з бази даних.
- Контролери (controllers): функції, які взаємодіють з базою даних. Контролери виконують операції обробки даних, які надходять від клієнта, та забезпечують взаємодію з моделями для збереження або отримання даних.
- Middleware: функції, які обробляють запити та відповіді сервера в проміжку між отриманням запиту та відправленням відповіді. Middleware дозволяють виконувати додаткові функції на рівні сервера, наприклад, перевірка аутентифікації, авторизації або обробка помилок, перед передачею запиту до наступного обробника або перед відправленням відповіді клієнту.
- Шляхи (routes): визначають, коли і які контролери будуть виконуватись для конкретних запитів. Шляхи встановлюють зв'язок між

маршрутами URL та відповідними контролерами, вказуючи, як сервер має обробляти різні типи запитів.

Також наявна папка db, яка містить у собі файли для підключення до бази даних. Структура файлової системи серверної частини зображена на рисунку 2.3.

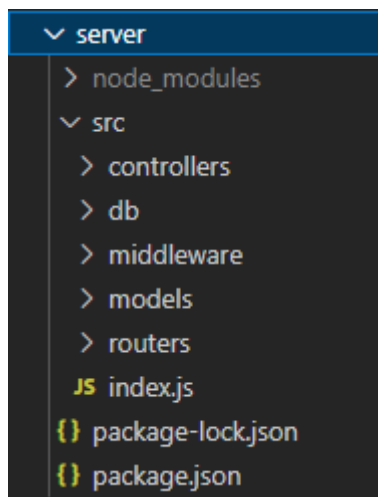


Рис. 2.3. – Структура файлів серверної частини.

Клієнтська частина системи, реалізована з використанням React, також буде мати модульну структуру файлів, що сприяє організованості та розширюваності кодової бази.

Файлова структура проекту буде включати наступні компоненти:

- Модуль "actions" містить файли, які визначають дії (actions) - функції, які ініціюють зміни стану додатка. Ці файли можуть містити функції для отримання даних з сервера, оновлення стану компонентів, обробки введення користувача тощо.
- Модуль "api" містить файл, який відповідає за взаємодію з сервером через API. Він містить функції для виконання HTTP-запитів до сервера, обробки відповідей та передачі даних між клієнтом і сервером.
- Модуль "components" містить компоненти, які відповідають за відображення різних елементів користувацького інтерфейсу. Кожен компонент може мати власний файл, який містить JSX-код, стилі та пов'язану логіку. Компоненти можуть бути повторно використовуваними та групуватися відповідно до функціональності.

- Модуль "reducers" має файли, які визначають редуктори (reducers) - функції, які обробляють дії та змінюють стан додатка. Редуктори визначають, які частини стану повинні бути змінені відповідно до отриманих дій. Кожен редуктор може мати свій власний файл і відповідати за певну частину стану.

- Модуль "routes" містить файли, які визначають маршрути (routes - шляхи для навігації в додатку), тобто компоненти, які відповідають за відображення сторінок.

Така модульна структура дозволяє логічно групувати файли за їх функціональністю, полегшує розуміння коду, сприяє перевикористанню компонентів та полегшує розширення додатка в майбутньому.

Адміністративна частина сайту, яка призначена для використання працівниками, має цілком подібну будову файлової системи.

Структура файлової системи клієнтської та адміністративної частин додатку зображена на рисунку 2.4.

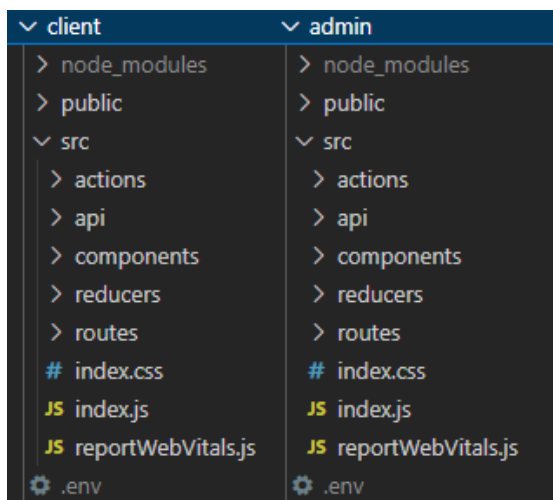


Рис. 2.4. – Структура файлів клієнтської та адміністративної частин.

2.3. Розробка структури бази даних.

У процесі розробки будь-якої сучасної програмної системи ключовою складовою є структура бази даних. Вона визначає спосіб зберігання та організацію даних, які використовуються для підтримки функціональності системи. Як базу даних для даного проєкту було обрано MongoDB. При розробці

структури бази даних на основі MongoDB необхідно враховувати її особливості та визначити оптимальний спосіб моделювання даних для вирішення потреб системи.

В базі даних будуть міститися колекції, які відповідають за функціонування модулів пов'язаних з користувачами, замовленнями та новинами.

Колекція «users» зберігає в собі дані про зареєстрованих користувачів веб-додатку. Документи цієї колекції мають такі поля:

- email – текстове поле, яке містить електронну пошту користувача;
- password – текстове поле, яке містить пароль користувача;
- fullName – об'єкт, який містить повне ім'я користувача:
 - firstName – текстове поле, яке містить ім'я;
 - lastName – текстове поле, яке містить прізвище;
 - patronymic – текстове поле, яке містить ім'я по-батькові.
- favourites – масив, який містить список бажаних платівок;
- address – об'єкт, який містить адресу користувача:
 - region – текстове поле, яке містить область;
 - city – текстове поле, яке містить місто;
 - street – текстове поле, яке містить вулицю;
 - apartment – текстове поле, яке містить номер квартири;
 - zipCode – текстове поле, яке містить поштовий індекс.
- phoneNumber – текстове поле, яке містить номер телефону;
- isBanned – булеан поле, яке містить інформацію чи забанений користувач;
- tokens – масив об'єктів, який потрібен для авторизації.

Колекція «orders» зберігає інформацію про замовлення користувачів. Документи цієї колекції мають наступні поля:

- products – масив об'єктів, які містять інформацію про замовлені товари:
 - id – поле типу ObjectId, яке містить ID товару слугує як посилання на відповідний документ в колекції товарів;
 - count – поле типу Number, яке містить кількість замовленого товару.

- totalPrice – поле типу Number, яке містить загальну вартість замовлення;
- email – текстове поле, яке електронну пошту замовника;
- address – текстове поле, яке містить адресу доставки;
- recipientName – текстове поле, яке містить ім'я отримувача;
- customerName – текстове поле, яке містить ім'я замовника;
- phoneNumber – текстове поле, яке містить номер телефону замовника;
- status – текстове поле, яке містить інформацію про статус замовлення;
- userId – поле типу ObjectId, яке містить ID зареєстрованого користувача і слугує як посилання на нього в колекції користувачів;
- dateOrdered – поле типу Date, яке містить дату створення замовлення;
- orderId – поле типу Number, яке містить номер замовлення.

Колекція «articles» зберігає інформацію про новини, які розміщуються у блозі. Документи цієї колекції мають наступні поля:

- shortTitle – текстове поле, яке містить заголовок новини;
- longTitle – текстове поле, яке містить опис новини;
- images – масив текстових полів, які містять посилання на зображення;
- text – текстове поле, яке містить текст новини;
- postDate – поле типу Date, яке містить дату створення новини;
- editDate – поле типу Date, яке містить дату останнього редагування новини;
- author – об'єкт, який містить інформацію про автора статті:
 - fullName – текстове поле, яке містить ім'я автора;
 - displayName – текстове поле, яке містить псевдонім автора;
 - socialMedia – масив текстових полів, що містять посилання на соціальні мережі автора;
 - email – текстове поле, яке містить електронну пошту.
- genres – масив полів типу ObjectId, які містять ID коментарів до новини.

Колекція «articleComments» зберігає інформацію про коментарі до новин, які можуть залишити зареєстровані користувачі. Документи цієї колекції мають наступні поля:

- userId – поле типу ObjectId, яке містить ID зареєстрованого користувача і слугує як посилання на нього в колекції користувачів.
- articleId – поле типу ObjectId, яке містить ID новини і слугує як посилання на неї в колекції новин.
- text – текстове поле, яке містить сам текст коментаря.
- creationDate – поле типу Date, яке містить дату створення коментаря.
- updateDate – поле типу Date, яке містить дату останнього редагування коментаря.

2.4. Реалізація функціоналу веб-додатку.

Процес реалізації функціоналу веб-додатку розпочинається з розробки серверної частини.

Першим кроком є створення схем, які визначають структуру документів, що будуть зберігатись у відповідній колекції бази даних [4]. Ці схеми далі будуть перетворені у моделі.

Для функціонування додатку однією з головних колекцій бази даних є колекція замовлень, яка зберігає необхідні дані про них. Код схеми цієї колекції вказана на рис. 2.5.

```
let orderSchema = new mongoose.Schema({
  products: [{
    id: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Product",
      required: true
    },
    count: { type: Number, required: true }
  }],
  totalPrice: { type: Number, required: true },
  email: { type: String, required: [true, 'Поле "Електронна адреса" не може бути порожнім'], lowercase: true },
  address: { type: String, required: [true, 'Поле "Адреса" не може бути порожнім'], trim: true },
  recipientName: { type: String, required: [true, 'Поле "Ім\'я одержувача" не може бути порожнім'], trim: true },
  customerName: { type: String, required: [true, 'Поле "Ім\'я покупця" не може бути порожнім'], trim: true },
  phoneNumber: { type: String, required: [true, 'Поле "Номер телефону" не може бути порожнім'], trim: true },
  status: { type: String, default: '📦 обробці' },
  userId: { type: mongoose.Schema.Types.ObjectId, ref: "User" },
  dateOrdered: { type: Date, default: Date.now },
  orderId: { type: Number }
});
```

Рис. 2.5. – Схема замовлення.

Кожен ключ в даному коді визначає поле в документі бази даних, його тип. Також вказані додаткові атрибути: `required` – означає, що поле є обов’язковим і має бути заповненим при створенні та оновленні документа, якщо це не виконається буде відправлене повідомлення з відповідним текстом помилки; `trim` – прибирає зайві пробіли з кінця та початку рядка; `default` вказує дані за замовчуванням; `ref` вказує на потрібну модель, до якої відноситься поле, що встановлює зв’язок між поточною моделлю та іншою моделлю, наприклад поле `userId` посилається на модель `User` і матиме в собі посилання на відповідний об’єкт тієї моделі.

Великою перевагою бібліотеки `Mongoose` є можливість створення валідації для полів. Так, наприклад в схемі користувача (`User`), можна провести валідацію для паролю (рис. 2.6).

```
password: {
  type: String,
  required: [true, 'Поле "Пароль" не може бути порожнім'],
  validate(value) {
    if (value.length < 7) {
      throw new Error("Пароль має містити менше 7 символів");
    }
    if (validator.contains(value, "password", { ignoreCase: true })) {
      throw new Error("Пароль не може містити слово: "password");
    }
  },
  trim: true,
},
```

Рис. 2.6. – Валідація поля `password` схеми `User`.

Для встановлення зв’язку між моделями важливо створити віртуальне поле. Наприклад, необхідне створення віртуального поля для того, щоб мати змогу встановити зв’язок між моделлю `User` і `ArticleComment`, щоб мати змогу визначити який користувач написав цей чи інший коментар (рис. 2.7). В атрибуті `ref` вказується з якою моделлю встановлюється зв’язок, `localField` вказує на поле в поточній моделі, яке використовується для зв’язку, `foreignField` вказує на поле в моделі з якою встановлюється зв’язок, воно містить унікальний ідентифікатор.

```
userSchema.virtual("articleComments", {
  ref: "ArticleComment",
  localField: "_id",
  foreignField: "userId",
});
```

Рис. 2.7. – Створення віртуального поля.

Для безпеки перед зберіганням даних такі дані як паролі необхідно захистити. Для цього варто використати проміжний обробник, доступний в Mongoose, який дозволить перед збереженням даних захешувати пароль, в результаті чого у відповідне поле документа запишеться хешоване значення, що покращить рівень безпеки (рис. 2.8).

```
userSchema.pre("save", async function (next) {
  const user = this;
  if (user.isModified("password")) {
    user.password = await bcrypt.hash(user.password, 8);
  }
  next();
});
```

Рис. 2.8. – Хешування паролю перед збереженням.

Повний вміст файлу з моделлю користувача наявний у додатку А.

Контролери створюються для взаємодії з базою даних, а саме для обробки запитів, взаємодії з моделями, формування та відправлення відповідей.

Контролери для замовлень є одними з ключових в системі, адже вони необхідні для роботи системи за вимогами.

Важливо створити контролер для отримання всіх даних про замовлення для працівника, щоб забезпечити зручність роботи (рис. 2.9).

```
// Get all orders for employee
exports.orders_list_employee = async (req, res) => {
  try {
    const orders = await Order.find().populate("products.id");
    res.status(200).send({ orders: orders });
  } catch (error) {
    res.status(500).send(error.message);
  }
};
```

Рис. 2.9. – Контролер для отримання даних для працівника.

Дана асинхрона функція робить запит до моделі замовлень за допомогою методу find(), який повертає всі документи відповідної колекції, а метод populate() використовується для заповнення поля products. У разі успішного виконання функції повертається масив з усіма документами, в іншому випадку повідомляється про помилку.

Для отримання замовлень користувача є окрема функція, яка працює аналогічним чином, проте метод `find()` містить в собі фільтр за `userId` і в такому випадку повертаються лише документи, які мають таке саме `userId`, тобто належать даному користувачу. Код цієї функції вказаний на рис. 2.10.

```
// Get all orders for user
exports.orders_list = async (req, res) => {
  try {
    const orders = await Order.find({ userId: req.user.id }).populate(
      "products.id"
    );
    res.status(200).send({ orders: orders });
  } catch (error) {
    res.status(500).send(error.message);
  }
};
```

Рис. 2.10. – Контролер для отримання замовлень користувача.

Важливими функціями для покращення роботи веб-додатку є можливість сортування, фільтрація та пошуку даних. Для цього варто створити окремий контролер (рис. 2.11).

```
exports.orders_by_search = async (req, res) => {
  const { searchQuery, status, start, end, sort } = req.query;
  let filter = {};
  let sorting = {};
  if (req.user) { filter.userId = { $eq: req.user.id }; }
  if (
    (typeof searchQuery === "string" || searchQuery instanceof String) &&
    searchQuery.length > 0
  ) { filter.orderId = { $eq: searchQuery }; }
  if (status) { filter.status = { $eq: status }; }
  if (start !== "0" && end !== "0") {
    filter.dateOrdered = { $gte: start, $lte: end };
  }
  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 orders = await Order.find(filter).populate("products.id")
      .sort(sorting);
    res.status(200).send({ orders: orders });
  } catch (error) {
    res.status(500).send(error.message);
  }
};
```

Рис. 2.11. – Контролер для отримання замовлень за фільтрацією.

Даний контролер отримує дані запити і, якщо дані існують, то об'єкт `filter` отримує нові поля за яких буде відбуватися фільтрація даних. Якщо в запиті

присутні дані для сортування, то вони записуються у відповідний об'єкт. Пошук здійснюється лише по номеру замовлення.

Варто зазначити, що якщо запит приходить від користувача, то автоматично однією з умов фільтрації є пошук даних, які мають те саме ID користувача. Відбувається пошук з вказаними даними для фільтрації і сортування, проводиться заповнення полів id в products відповідними документами завдяки методу populate() і у разі успіху повертається масив зі знайденими документами. Якщо ж ні, то у відповідь повертається помилка.

Контролер для отримання деталей одного замовлення користувача здійснює пошук по id замовлення за допомогою методу findById() і також здійснює заповнення полів id в products відповідними документами завдяки методу populate(). У разі, якщо такого замовлення не було знайдено, відправляється повідомлення з кодом помилки 404. Якщо ж таке замовлення існує і ID користувача, вказане в ньому, збігається з ID користувача, який зробив запит, то об'єкт цього замовлення повертається користувачу, якщо ж ні, то відправляється повідомлення з кодом помилки 403 або ж з кодом 500. Код цього контролера зображений на рис. 2.12.

```
// Get order details as user
exports.order_details_user = async (req, res) => {
  try {
    const order = await Order.findById(req.params.id).populate("products.id");
    if (!order) {
      return res.status(404).send();
    }
    if (order.userId && req.user.id === order.userId.toString()) {
      res.status(200).send(order);
    } else {
      return res.status(403).send("Ви не маєте доступу");
    }
  } catch (error) {
    res.status(500).send(error.message);
  }
};
```

Рис. 2.12. – Контролер для отримання одного замовлень користувача.

Контролер для отримання інформації про одне замовлення, коли запит надходить від працівника, працює аналогічним чином, проте відсутня перевірка чи збігаються ID користувача.

Важливою функцією є створення нового замовлення. В контролері, який відповідає за такий функціонал, створюється новий об'єкт моделі Order з даними, які надійшли від користувача. Також цей контролер забезпечує зменшення кількості замовленого товару на складі шляхом пошуку цього товару та оновлення його кількості за допомогою методу findByIdAndUpdate(). Далі нове замовлення зберігається за допомогою методу save() і в результаті успіху інформація про це разом з кодом 200 і новим замовленням відправляється клієнту. Якщо ж виникли помилки, то в такому випадку відправляється код 500 і повідомлення помилки. Код контролера зображений на рис. 2.13.

```
// Create new order
exports.order_create = async (req, res) => {
  try {
    const order = new Order({ ...req.body });
    for (i = 0; i < req.body.products.length; i++) {
      await Product.findByIdAndUpdate(req.body.products[i].id, {
        $inc: { itemsAvailable: -req.body.products[i].count },
      });
    }
    await order.save();
    res.status(200).send({ success: true, order: order });
  } catch (error) {
    res.status(500).send(error.message);
  }
};
```

Рис. 2.13. – Контролер для створення замовлення.

Користувач також має змогу відмінити власне замовлення, змінивши його статус. Для цього відповідний контролер спочатку шукає відповідне замовлення за його ID. Якщо такого замовлення не існує, то відправляється статус з кодом 404. Якщо таке замовлення існує, його статус не є «Відправлено», а статус, який бажає поставити користувач це «Скасовано» і ID користувача, що робить запит, збігається з ID користувача в замовленні, то статус замовлення оновлюється і відповідь про успіх разом з кодом 200 і оновленим замовленням відправляється користувачу. Якщо ж виникла якась помилка, то натомість відправляється

повідомлення про цю помилку і статус 500. Код контролера зображений на рис. 2.14.

```
// Order canceled by user
exports.order_cancel = async (req, res) => {
  try {
    const order = await Order.findById(req.params.id);
    if (!order) { return res.status(404).send(); }
    if (req.body.status === "Скасовано" && order.status !== "Відправлено" &&
        req.user.id === order.userId.toString()) {
      order.status = req.body.status;
    }
    await order.save();
    res.status(200).send(order);
  } catch (error) {
    res.status(500).send(error.message);
  }
};
```

Рис. 2.14. – Контролер для відміни замовлення.

Контролер для зміни даних замовлень для працівника працює аналогічним чином. Проте, в цьому випадку відсутня перевірка ID користувача, а також, якщо статус замовлення не є «Відправлено», то у працівника є можливість зміни інші поля на прохання користувача. Код цього контролера зображений на рис. 2.15.

```
// Update an existing order (as an employee)
exports.order_update = async (req, res) => {
  try {
    const order = await Order.findById(req.params.id);
    if (!order) {
      return res.status(404).send();
    }
    if (req.body.status && order.status !== "Відправлено") {
      order.status = req.body.status;
    }
    if (order.status !== "Відправлено") {
      if (req.body.address) { order.address = req.body.address; }
      if (req.body.recipientName) { order.recipientName = req.body.recipientName; }
      if (req.body.customerName) { order.customerName = req.body.customerName; }
      if (req.body.phoneNumber) { order.phoneNumber = req.body.phoneNumber; }
      if (req.body.email) { order.email = req.body.email; }
    }
    await order.save();
    res.status(200).send(order);
  } catch (error) {
    res.status(500).send(error.message);
  }
};
```

Рис. 2.15. – Контролер для оновлення замовлення працівником.

Варто зазначити, що контролер, який відповідає за видалення записів з інформацією про замовлення відсутній, адже видалення замовлень є поганою практикою.

Кінцеві точки необхідні для встановлення точок доступу до різних функціональних частин додатку. Вони визначають, які контролери та за яких умов мають бути викликані для обробки запиту, тобто встановлюють зв'язок між URL-шляхом і функцією, яка буде виконуватись при зверненні до цього URL.

Оскільки доступ до деяких функцій мають лише працівники або автентифіковані користувачі, перед викликом відповідного контролера для обробки запиту використовується middleware функції `userAuth` і `employeeAuth`, які перевіряють авторизацію користувачів, які надають запит.

Кінцеві точки для замовлень зображені у таблиці 2.1.

Таблиця 2.1

Кінцеві точки для замовлень

Кінцева точка	HTTP-метод	Необхідність автентифікації	Опис
/orders	GET	Так	Повертає список всіх замовлень для працівника.
/admin/orders/search	GET	Так	Повертає список замовлень, які підпадають під фільтрацію, сортування, пошук для працівника.
/users/orders/all	GET	Так	Повертає список усіх замовлень користувача.
/orders/search	GET	Так	Повертає список замовлень, які підпадають під

Продовження таблиці 2.1.

			фільтрацію, сортування, пошук для користувача.
/orders/create	POST	Ні	Повертає створене замовлення.
/orders/:id	GET	Так	Повертає деталі замовлення для працівника.
/users/orders/:id	GET	Так	Повертає деталі замовлення для користувача.
/orders/update/:id	PATCH	Так	Повертає оновлене замовлення для працівника.
/users/orders/update/:id	PATCH	Так	Повертає оновлене замовлення для користувача.

Клієнтська та адміністративна частини веб-додатку реалізовані з використанням однакової архітектури та підходів, але відрізняються функціоналом, який надається користувачам.

В обох частинах проекту присутні файли, що визначають API функції, які забезпечують взаємодію з сервером шляхом відправлення відповідних запитів до нього. Ці API функції включають HTTP методи (наприклад, GET, POST, PUT, DELETE) та URL кінцевих точок сервера, до яких здійснюються запити.

Код функцій, які відповідають за отримання та відправлення даних про замовлення на сервер з адміністративної частини наведено на рис. 2.16.

```

export const fetchOrders = () => API.get(`/orders`);
export const fetchOrdersBySearch = (searchQuery) =>
  API.get(
    `/admin/orders/search?searchQuery=${searchQuery.search}&status=${
      searchQuery.status || ""
    }&start=${searchQuery.start}&end=${searchQuery.end}&sort=${
      searchQuery.sort
    }`
  );
export const fetchOrder = (id) => API.get(`/orders/${id}`);
export const updateOrder = (id, updatedOrder) => API.patch(`/orders/update/${id}`, updatedOrder);

```

Рис. 2.16 – API функції замовлень адміністративної частини.

Обидві частини мають модуль actions, в якому в окремих файлах розміщуються функції, які стосуються тієї чи іншої колекції (або ж автентифікації).

Дані функції є екшенами Redux, які використовуються для виклику змін в стані додатку [7]. Загалом роботу цих функцій можна розглянути на прикладі функції `getOrders()`, яка використовується в адміністративній частині веб-додатку (рис. 2.17).

```

export const getOrders = () => async (dispatch) => {
  try {
    const { data } = await api.fetchOrders();
    dispatch({ type: "FETCH_ALL", payload: data.orders });
  } catch (error) {
    console.log(error.message);
  }
};

```

Рис. 2.17. – Функція `getOrders()`.

Спочатку йде виклик `fetchOrders()`, яка відповідає за отримання даних всіх замовлень з сервера, в результаті чого отримані дані записуються в об'єкт `data`.

Далі, викликається функція `dispatch()`, яка виконує відправку екшенів до Redux Store, з об'єктом екшена в якому вказаний тип і дані, які будуть передаватися і зберігатися в стані додатку. Якщо ж при виконанні даного блоку виникла помилка, то повідомлення з неї виведеться до консолі.

Функцію `getOrders()` можна надалі викликати в різних компонентах використовуючи `dispatch`, для отримання та оновлення даних в Redux Store.

Керування станом об'єкту відповідно до дій (екшенів) здійснених у додатку відбувається за допомогою редюсерів [7].

Принцип роботи редюсерів можна переглянути на прикладі замовлень (рис. 2.18).

```
const orders = (state = {orders: []}, action) => {
  switch(action.type) {
    case 'FETCH_ALL':
      return {...state, orders: action.payload};
    case 'FETCH_ORDER':
      return {...state, order: action.payload};
    case 'ORDER_STATISTICS_NUMBERS':
      return {...state, statisticsNumbers: action.payload};
    case 'ORDER_STATISTICS_PRICES':
      return {...state, statisticsPrices: action.payload};
    default:
      return state;
  }
};

export default orders;
```

Рис. 2.18. – Редюсер для замовлень.

Редюсер orders приймає два параметри: state (початковий стан об'єкту) і action (екшн, який викликав зміну стану). Функція використовує switch для перевірки типу екшена. Залежно від його типу виконується певна логіка для зміну стану. У кожному випадку створюється новий об'єкт шляхом розпакування поточного стану (state) і заміни відповідного поля за допомогою action.payload, яке містить дані, які будуть збережені в стані.

Наприклад, у випадку 'FETCH_ALL' створюється новий об'єкт стану, де поле orders буде містити значення action.payload, яке передає список замовлень. Аналогічно і для інших типів екшенів. Якщо ж тип екшена не співпадає з жодним з визначених випадків, то повертається незмінний поточний стан state.

Варто зазначити, що у разі зміни заборони користувачу коментувати редюсер буде працювати наступним чином: буде використовуватись випадок 'BAN_USER', при якому використовується метод map(), який перебирає всіх користувачів у поточному масиві state.users. Коли знайдений користувач з action.payload._id (ідентифікатор, який міститься в action.payload), його стан змінюється на новий стан користувача. Інші ж користувачі залишається

незмінними, а оновлений масив користувачів стає частиною нового стану. Програмний код даного варіанту зображений на рис. 2.19.

```
const users = (state = {users: []}, action) => {
  switch(action.type) {
    case 'BAN_USER':
      return {...state, users: state.users.map((user) =>
        (user._id === action.payload._id ? action.payload : user))};
    case 'FETCH_USERS': ...
    case 'FETCH_USER': ...
    default: ...
  }
}
```

Рис. 2.19. – Редюсер для користувачів.

Модуль components містить компоненти, які можуть повторно використовуватись в роутерах. Наприклад, в компонентах адміністративної частини міститься файл FormattedDate.js, який дає змогу перетворити дату у легкий для сприйняття для користувачів системи варіант за допомогою бібліотеки Moment [13]. У файлі знаходиться дві функції, FormattedDate і FormattedDateArticle, які приймають дату, приводять її до потрібного формату, та повертають для відображення у відповідному полі на відповідному компоненті.

Один з компонентів на клієнтської частини відповідає за картки для відображення короткої інформації про новину. Цей компонент приймає дані новини і використовується як на головній сторінці, так і на сторінці зі списком всіх новин.

Модуль routes містить компоненти, які містять функції, які відповідають за вигляд самих сторінок веб-додатку та відображення і обробку даних. Кожен компонент починається з імпортування необхідних компонентів з бібліотеки React Material UI, React Redux, React, React Router Dom для відображення даних та виконання функцій. Також підключаються раніше написані екшенс для забезпечення необхідного функціоналу.

Для роботи зі станом додатку необхідно використовувати хуки useEffect і useSelector, які належать до бібліотеки React Redux і надають потужні

можливості для управління станом і взаємодії з Redux Store.

Хук `useEffect`, який дозволяє виконувати певну функцію після кожного рендеру компонента або при зміні певних залежностей, приймає два аргументи: функцію, яка буде виконана і масив залежностей [5]. При першому рендері компонента виконується функція, яка була передана у хук. Далі ця функція буде виконуватися знову, лише якщо відбудеться зміни елементів в масиві залежностей. Якщо масив залежностей буде представлений порожнім масивом, то функція всередині хука виконається лише один раз. Якщо масив залежностей буде повністю відсутній, то функція буде спрацьовувати як при першому рендері, так і при кожному наступному.

Хук `useSelector` використовується для отримання значень зі стану збереженого в Redux Store. Він дозволяє компонентам отримувати лише необхідні дані зі стану без необхідності передавати весь стан як `props` через ієрархію компонентів.

Хук приймає функцію-селектор як аргумент, яка визначає, яка саме частина стану буде обрана. При зміні цього вибраного стану, компонент буде перерендерюватись.

Вигляд отримання інформації про замовлення за допомогою вище описаних хуків зображений на рис. 2.20.

```
export default function Orders() {  
  const dispatch = useDispatch();  
  const orders = useSelector((state) => state.orders);  
  
  useEffect(() => {  
    dispatch(getOrders());  
  }, [dispatch]);  
}
```

Рис. 2.20. – Робота зі станом в замовленнях.

Форми для додавання і оновлення інформації про новину або замовлення представлені одним компонентом, тому для їх функціонування також є надзвичайно важливими такі хуки як `useParams` і `useState`.

Хук `useParams` дозволяє отримати параметри маршруту в компоненті, які повертає у формі об'єкту, в якому ключами є імена параметрів, які були визначені в маршруті, а значення – фактичні значення, які знаходяться в URL.

У випадку використання форми для оновлення інформації про новину в URL буде передано ідентифікатор цього замовлення, який необхідний для отримання дані, які складають цю новину, з метою заповнення полів вже існуючою інформацією для полегшення проведення операції оновлення.

Щоб отримати інформацію про новину використовується хук `useEffect`, який за допомогою методу `dispatch` викликає відповідну функцію пошуку з екшенс.

Для заповнення полів та управління і відображення їх змін використовується хук `useState`, який дозволяє відстежувати стан у функціональному компоненті [6]. Даний хук приймає початкове значення стану і повертає масив, в якому перший елемент – поточне значення стану, а другий елемент – функція, яка дозволяє змінювати значення стану. Використання хука `useState` для оновлення та створення новини зображено на рис. 2.21.

```
export default function ArticleForm() {  
  const dispatch = useDispatch();  
  const navigate = useNavigate();  
  const [formData, setFormData] = useState(initialState);  
  const [imagesList, setImagesList] = useState([]);  
  const [authorData, setAuthor] = useState({});  
  const [articleText, setArticleText] = useState("");  
}
```

Рис. 2.21. – `useState` для форми новини.

Загальна інформація про статтю міститься в `formData`. Проте, щоб оновлювати та зберігати інформацію в об'єктах та масивам доводиться використовувати окремі для цього функції, що і призводить до використання окремих `useState`. Текст новини також має окремий `useState`, оскільки для написання тексту новини використовується не звичайне текстове поле, а редактор `TinyMCE` [12].

Проте, лише хука `useState` замало для відображення початкових даних, необхідно також використати `useEffect`, за допомогою якого можна буде задати початкові значення станів (рис. 2.22).

```
useEffect(() => {
  if (article) {
    setArticleText(article.text);
    setFormData({
      ...article,
      genres: article.genres.map((genre) => genre._id),
    });
    setAuthor({
      ...article.author,
    });
    setImagesList(article.images);
  }
}, [article]);
```

Рис. 2.22. – `useEffect` для встановлення початкових значень станів.

Оскільки форма використовується не тільки для оновлення даних, а й для створення нової новини, то спочатку проходить перевірка чи був знайдений відповідний запис з уже існуючою новиною і тоді відбувається заповнення початкових станів.

Щоб зберегти та відобразити нові зміни до полів, використовуються функції типу `handleChange()`, які спрацьовують при зміні у полів вводу. Приклад такою функції представлений функцією, яка спрацьовує при зміні інформації про автора, зображений на рис. 2.23.

```
const handleAuthorChange = (e) => {
  setAuthor({
    ...authorData,
    [e.target.name]: e.target.value,
  });
};
```

Рис. 2.23. – Функція `handleAuthorChange`.

При відправці форми спрацьовує функція `handleSubmit`, в якій всі дані необхідні для формування документа новини в колекції збираються до купи. Також, в залежності чи наявний ідентифікатор визначається чи викликається функція оновлення новини, чи створення новини. Обидві функції викликаються за допомогою методу `dispatch` і мають додатковий параметр `navigate`, який забезпечить перенаправлення на сторінку зі списком усіх новин.

Форма оформлення замовлення на клієнтській частині веб-додатку, при першому рендері компоненту звертається до API Нової Пошти для отримання актуального списку міст, які обслуговуються Новою Поштою [10]. Для цього використовується хук `useEffect` і `Fetch API`, яке дає змогу звернутися до API Нової Пошти і отримати потрібні дані. Запит зображений на рис. 2.24.

```
useEffect(() => {
  fetch(novaPoshtaURL, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      apiKey: NOVA_POSHTA_API,
      modelName: "Address",
      calledMethod: "getCities",
      methodProperties: {},
    }),
  })
    .then((response) => response.json())
    .then((response) => setAreasList(response.data));
}, []);
```

Рис. 2.24. – Отримання даних міст від API Нової Пошти.

Коли користувач вибирає бажане місто для доставки, за допомогою `useState` ці дані записуються і відправляється новий запит до API Нової Пошти для отримання списку відділень в обраному місті [11]. Запит зображений на рис. 2.25.

```
useEffect(() => {
  if (city) {
    fetch(novaPoshtaURL, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        apiKey: NOVA_POSHTA_API,
        modelName: "Address",
        calledMethod: "getWarehouses",
        methodProperties: {
          CityName: city,
          TypeOfWarehouseRef: "841339c7-591a-42e2-8233-7a0a00f0ed6f",
        },
      }),
    })
      .then((response) => response.json())
      .then((response) => setDepartmentList(response.data));
  }
}, [city]);
```

Рис. 2.25. – Отримання даних відділень від API Нової Пошти.

Для збільшення кількості можливостей на формі можна також обрати варіант доставки. Рендеринг необхідних полів контролюється за допомогою тернарної операції, яка перевіряє значення змінної `shippingType`, стан якої в свою чергу контролюється за допомогою хуку `useState`. Саме ж значення `shippingType` змінюється наступним чином: при зміні (натисканні) відповідного перемикача викликається метод `setShippingType`, який записує у змінну `shippingType` відповідне значення.

При оформленні замовлення також є можливість вказати чи отримувач є іншою людиною. За це відповідає змінна `checked`, стан якої за замовчування `false`, що означає, що замовник і одержувач одна людина, що буде враховано у функції `handleSubmit`, яка обробляє відправлення форми. Якщо її стан змінити на `true`, за допомогою відповідного чекбоксу, то з'явиться додаткова секція на сторінці, де можна буде ввести дані отримувача.

При правильному заповненні усіх даних користувач зможе провести оплату замовлення. Оплата виконується за допомогою системи `LiqPay`, для чого було підключено їх API [8]. Спочатку відбувається формування даних та підпису відносно документації API `LiqPay` [9]. Формування підпису та даних зображено на рис. 2.26.

```
let json_string = JSON.stringify({
  public_key: LIQPAY_PUBLIC_KEY,
  version: "3",
  action: "pay",
  amount: state.totalPrice,
  currency: "UAH",
  description: "Придбання платівок",
  order_id: orderId,
  result_url: "http://localhost:3001/order/thankyou",
});

const data = json_string.toString(Base64);

const sign_string = LIQPAY_PRIVATE_KEY + data + LIQPAY_PRIVATE_KEY;

const signature = sha1(sign_string).toString(Base64);
```

Рис. 2.26. – Формування даних для запиту до API `LiqPay`.

Тіло запиту має в собі дані про вартість замовлення, валюту, призначення, ідентифікатор замовлення та посилання на сторінку, яка відкриється у разі успішної оплати.

Повний програмний код сторінки, яка дозволяє створити нове замовлення, наведено у додатку Б.

Далі тіло шифрується як бінарні дані. Формується також підпис, який містить приватний ключ до API та зашифровані дані, які далі шифруються за допомогою SHA-1 і знову зводяться до бінарного вигляду.

Оскільки для забезпечення функціоналу веб-додатку використовуються різні сторонні API, для підключення яких необхідно отримати власні ключі, які не повинні бути вказані у вихідному коді для забезпеченні безпеки конфіденційної інформації, ключі підключення до них зберігаються у файлі з розширенням `env`.

Такий файл містить змінні оточення, які використовуються для налаштування та конфігурації додатків або середовищ виконання. Цей файл використовується для відокремлення конфігураційних даних від вихідного коду додатку, що дозволяє змінювати ці значення без необхідності перекомпіляції або змінювання самого коду.

Основна мета використання `.env` полягає у збереженні конфіденційної інформації, такої як паролі, ключі API або секретні токени, у безпечному місці. Цей файл дозволяє зберігати такі конфіденційні дані окремо від вихідного коду, що підвищує безпеку додатку.

Файл `.env`, який містить інформацію для роботи функціоналу клієнта, зображений на рис. 2.27 (ключі доступу були змінені).

```
client > .env
1  REACT_APP_SERVER_URL='http://localhost:7000/'
2  REACT_APP_SPOTIFY_CLIENT_ID='asdfghjklqwertyuiosdfghjkl'
3  REACT_APP_SPOTIFY_CLIENT_SECRET='dfghjklxcvbnmdfghjkl'
4  REACT_APP_NOVAPOSTA_API_KEY='sdfghjklxcvbnmertzuiivbn'
5  REACT_APP_LIQPAY_PRIVATE_KEY='ertzuiodfghjklfghjkkjhgfdfghjkhv'
6  REACT_APP_LIQPAY_PUBLIC_KEY='wertzuiopüdfghjklöxcvbnm'
```

Рис. 2.27. – Файл `.env`.

Однією з функцій, яка покращує використання веб-додатку для користувача, є можливість додавання товарів до списку «Бажане». Ця функція доступна не лише автентифікованим користувачам, адже відбувається використання localStorage.

localStorage – це об'єкт, що забезпечує механізм збереження даних в браузері клієнта. Він дозволяє зберігати дані у вигляді пар "ключ-значення" без обмеження часу життя даних. Дані, збережені у ньому, залишаються доступними навіть після закриття браузера або перезавантаження сторінки. Це робить його корисним для збереження стану додатків, налаштувань користувача, кешування даних або будь-яких інших інформаційних потреб, які потребують довготривалого збереження.

Для реалізації функціоналу списку «Бажане» необхідно спочатку перевірити чи автентифікований користувач. Далі, за допомогою хуку useEffect, якщо користувач все ж таки автентифікований, то необхідно створити елемент favourites в localStorage і заповнити його бажаними товарами з даних користувача і визначити іконки на картках з товаром, які позначають чи є товар у списку «Бажане». В іншому ж випадку елемент favourites, якщо він не існує, буде порожнім масивом (рис. 2.28).

```
useEffect(() => {
  if (user) {
    localStorage.setItem("favourites", JSON.stringify(user.favourites));
  } else if(localStorage.getItem("favourites") === null) {
    localStorage.setItem("favourites", JSON.stringify([]));
  }
  JSON.parse(localStorage.getItem("favourites")) &&
  JSON.parse(localStorage.getItem("favourites")).find((e) => e === id)
    ? setFavIcon(true)
    : setFavIcon(false);
}, [user]);
```

Рис. 2.28. – Початкове визначення списку «Бажане».

Для додавання та видалення товарів зі списку «Бажане» було створено дві функції, які спрацьовують на відповідну активну кнопку.

Для обробки додавання платівки до списку спочатку отримується поточне значення favourites з localStorage, яке за допомогою JSON.parse перетворюється на масив. Потім ідентифікатор товару додається до масиву, використовуючи метод push, і оновлюється значення в localStorage. Якщо користувач автентифікований, то за допомогою методу dispatch викликається функція, яка оновлює дані списку бажаних товарів користувача. Також змінюється іконка, яка відповідає за додавання/вилучення платівок зі списку (рис. 2.29).

```
const handleFavourites = () => {
  let favourites = JSON.parse(localStorage.getItem("favourites"));
  favourites.push(id);
  localStorage.setItem("favourites", JSON.stringify(favourites));
  if (user) dispatch(updateFavourites({ favourites: favourites }));
  setFavIcon(true);
};
```

Рис. 2.29. – Опрацювання додавання в список «Бажане».

Для обробки вилучення платівки зі списку «Бажане» спочатку отримується поточне значення favourites з localStorage, яке за допомогою JSON.parse перетворюється на масив. Далі виконується фільтрація масиву за допомогою методу filter(), в результаті якої у масиві залишаються лише елементи, які не дорвнюють заданому id. Оновлений масив зберігається в localStorage. За умови, що користувач автентифікований, якщо довжина масиву favourites складає 0, то за допомогою методу dispatch викликається функція, що відповідає за оновлення даних списку бажаних товарів в яку передається порожній масив, якщо ж favourites має довжину більше 0, то у функцію оновлення передається оновлений масив товарів. Також змінюється іконка (рис. 2.30).

```
const handleFavouritesDelete = () => {
  let favourites = JSON.parse(localStorage.getItem("favourites"));
  localStorage.setItem(
    "favourites",
    JSON.stringify(favourites.filter((e) => e !== id))
  );
  favourites = JSON.parse(localStorage.getItem("favourites"));
  if (user) {
    if (favourites.length === 0)
      dispatch(updateFavourites({ favourites: [] }));
    else dispatch(updateFavourites({ favourites: favourites }));
  }
  setFavIcon(false);
};
```

Рис. 2.30. – Опрацювання вилучення зі списку «Бажане».

Висновки до розділу 2.

У другому розділі було здійснено розподіл обов’язків з виконання проєкту та налагоджено роботу у групі, визначено вимоги користувачів щодо функціоналу системи. На основі цих вимог були складені діаграми використання функцій системи для користувачів та працівників.

Крім того, була розроблена структура файлів системи, яка відображає організацію різних модулів та компонентів у веб-додатку.

Була описана також структура бази даних, де було визначено колекції, що будуть використовуватись для збереження даних у системі.

Було реалізовано задуманий функціонал веб-додатку та детально описано алгоритм роботи важливих функцій на прикладах.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ДОДАТКОМ

3.1. Порядок встановлення та налаштування системи.

Розроблений додаток має архітектуру клієнт-сервер, що дозволяє розділити серверну, клієнтську та адміністративну частини і розмістити їх окремо. Для розгортання серверної частини додатку варто враховувати мінімальні вимоги, що залежать від очікуваного трафіку додатку. Однак, рекомендується врахувати ці вимоги з запасом, щоб забезпечити належну продуктивність. Тому, для розгортання серверної частини додатку рекомендується мати сервер з такими характеристиками:

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

Для запуску серверної частини додатку необхідно виконати наступні дії:

1. Встановити Node.js.

2. За допомогою команди `npm install` провести встановлення залежностей додатку, вказаних у файлі `package.json`.

3. Запустити сервер за допомогою команди `npm run`.

Для розгортання клієнтської та адміністративної частин додатку необхідно задовольнити аналогічні потреби і провести аналогічні дії.

Для взаємодії з веб-додатком підійде будь-який сучасний веб-браузер, наприклад Google Chrome версії 45+, Mozilla Firefox версії 38+ та інші.

3.2. Структура інтерфейсу. Інтерфейс та порядок роботи з системою.

Для користування адміністративною частиною системи користувач зобов'язаний автентифікуватися в системі. Після успішної автентифікації користувач побачить головну сторінку з актуальною статистикою веб-додатку, яку розробив Демченко Ярослав.

На кожній сторінці присутня навігаційна панель (рис. 3.1). Верхня частина має кнопки для перемикання теми і кнопку профіля, на якій зображені ініціали

					ІПЗ.КР.Б – 121 – 23 – ПЗ	51

працівника і при наведенні на яку показується ім'я працівника. При натисканні на кнопку профілю відкривається навігаційне меню працівника, яке розробив Демченко Ярослав.

З лівого боку міститься навігаційне меню, для переходу між сторінками веб-додатку. Дане меню можна розгорнути для зручності (рис. 3.2).

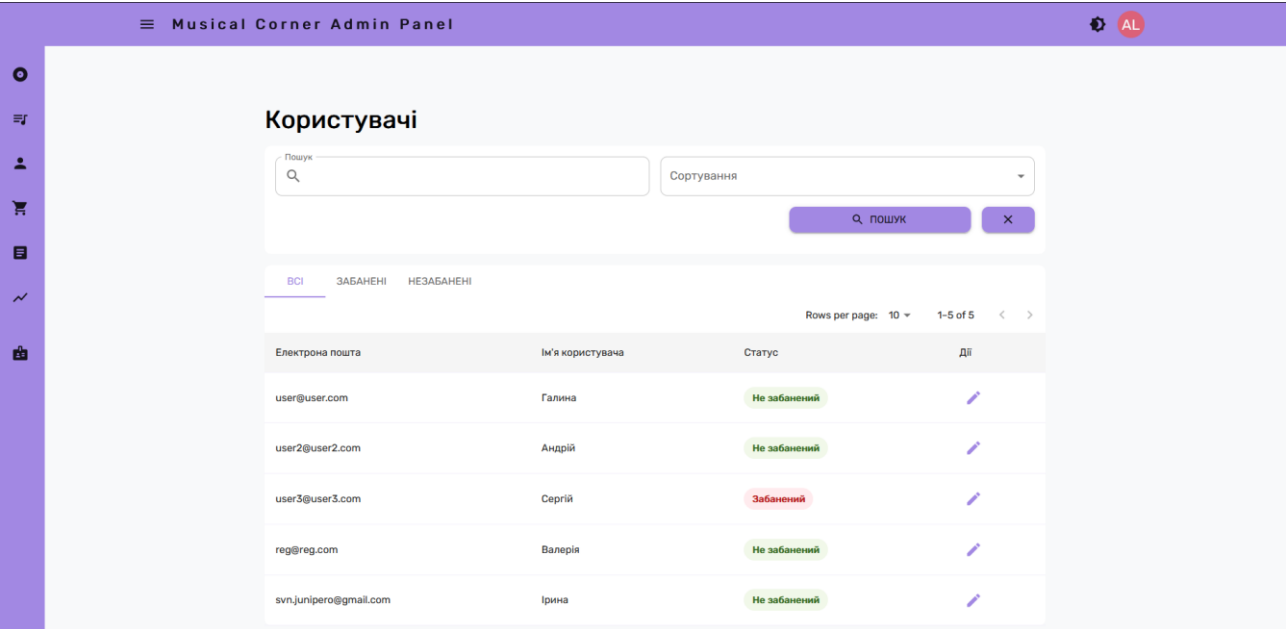


Рис. 3.1. – Сторінка користувачів.

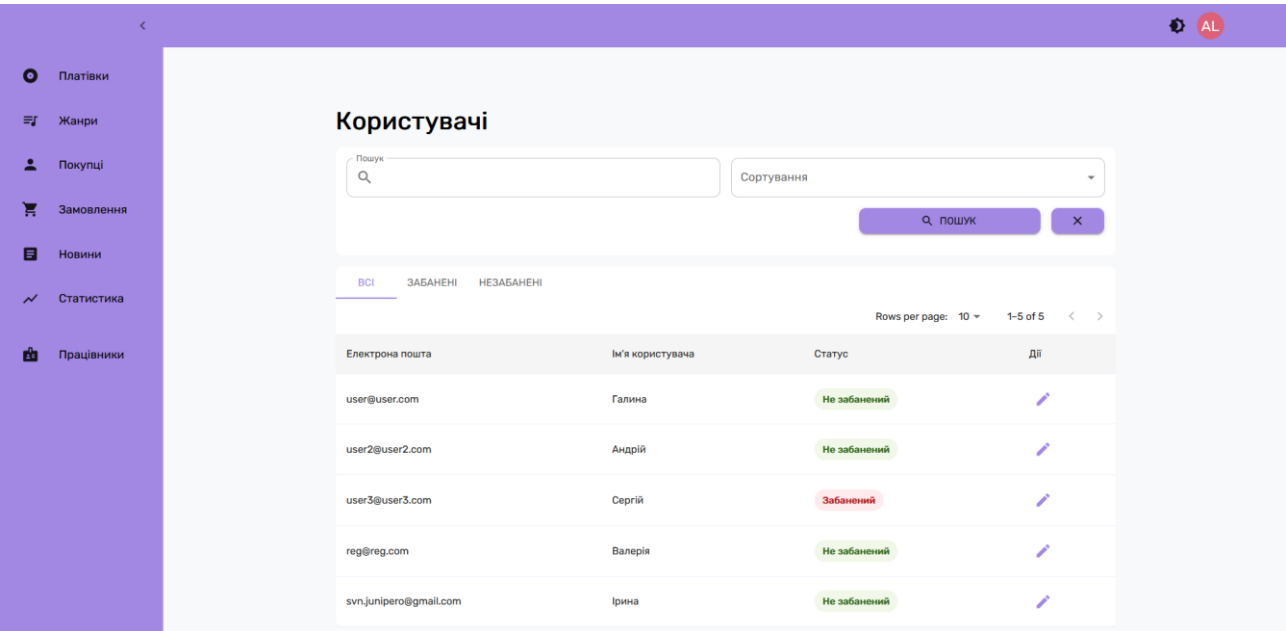


Рис. 3.2. – Розгорнута бокова навігаційна панель.

При натисканні на іконку теми, в залежності від поточної теми, вона зміниться на темну або світлу (рис. 3.3). Дана функція присутня також на клієнтській частині системи.

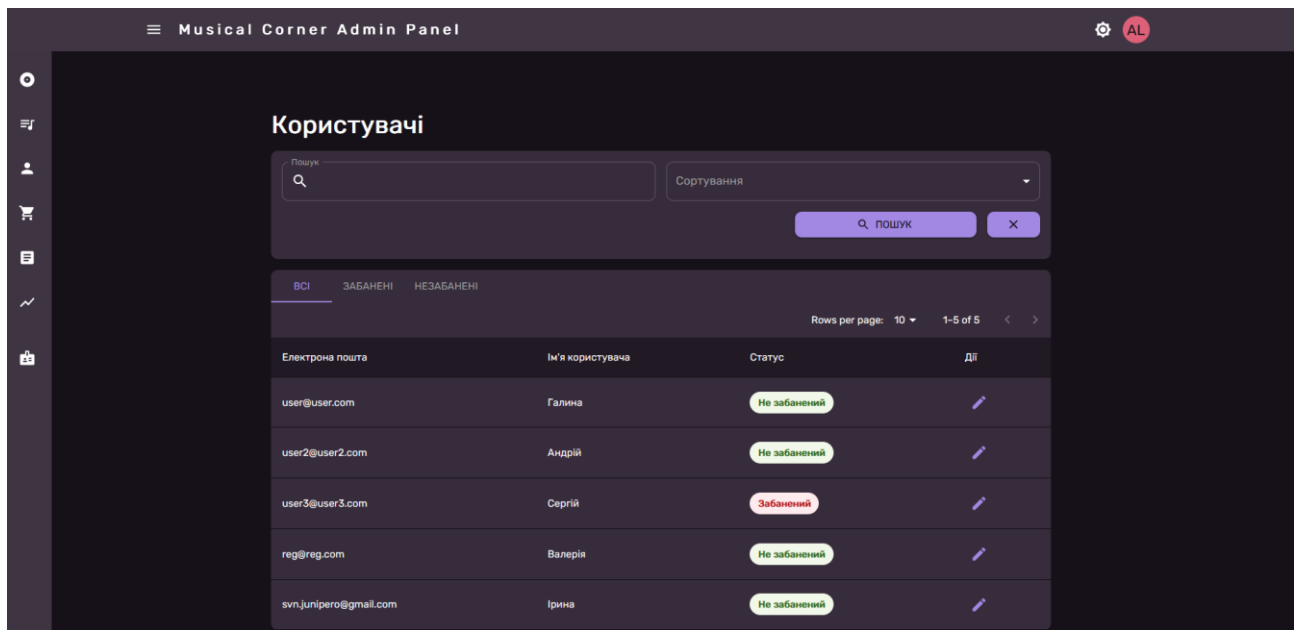


Рис. 3.3. – Темна тема.

На навігаційному меню можна натиснути на іконку користувачів і перейти на сторінку з інформацією про них.

Сторінка всіх користувачів зображає таблицю з короткою інформацією про зареєстрованих користувачів (рис. 3.1). Наявні вкладки для кращої фільтрації, при натисканні, наприклад, на вкладку «Забанені» будуть відображені користувачі, які не мають можливості коментувати та залишати відгуки (рис. 3.4).

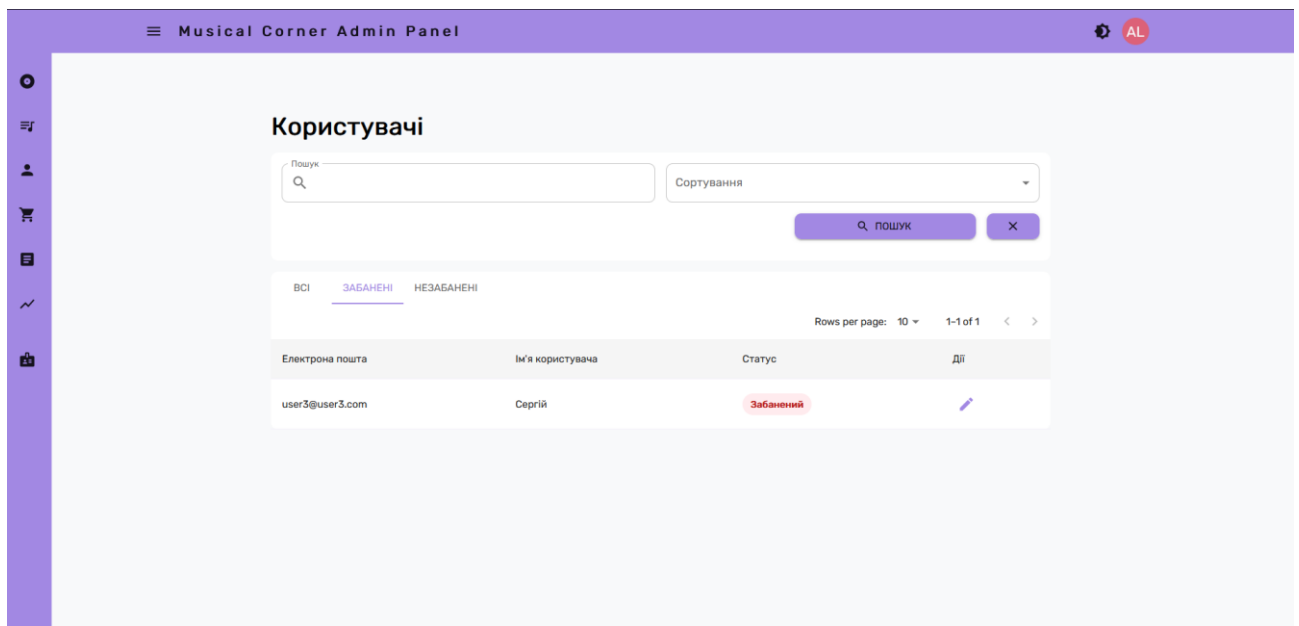


Рис. 3.4. – Відображення забаних користувачів.

На сторінці наявна також панель з інструментами для пошуку та сортування даних. Наприклад, можна відсортувати за іменем (рис. 3.5).

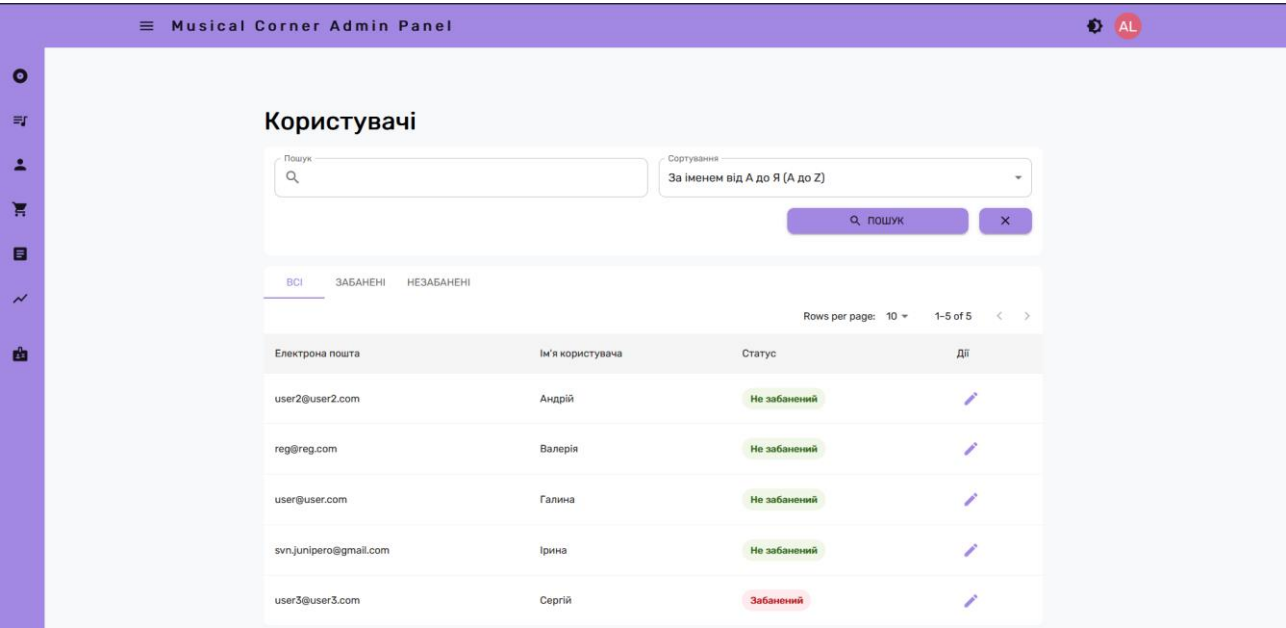


Рис. 3.5. – Відсортовані дані за іменем.

При натисканні на кнопку, яка міститься в колонці «Дії», з'явиться модаль, який дає змогу забанити користувача (рис. 3.6).

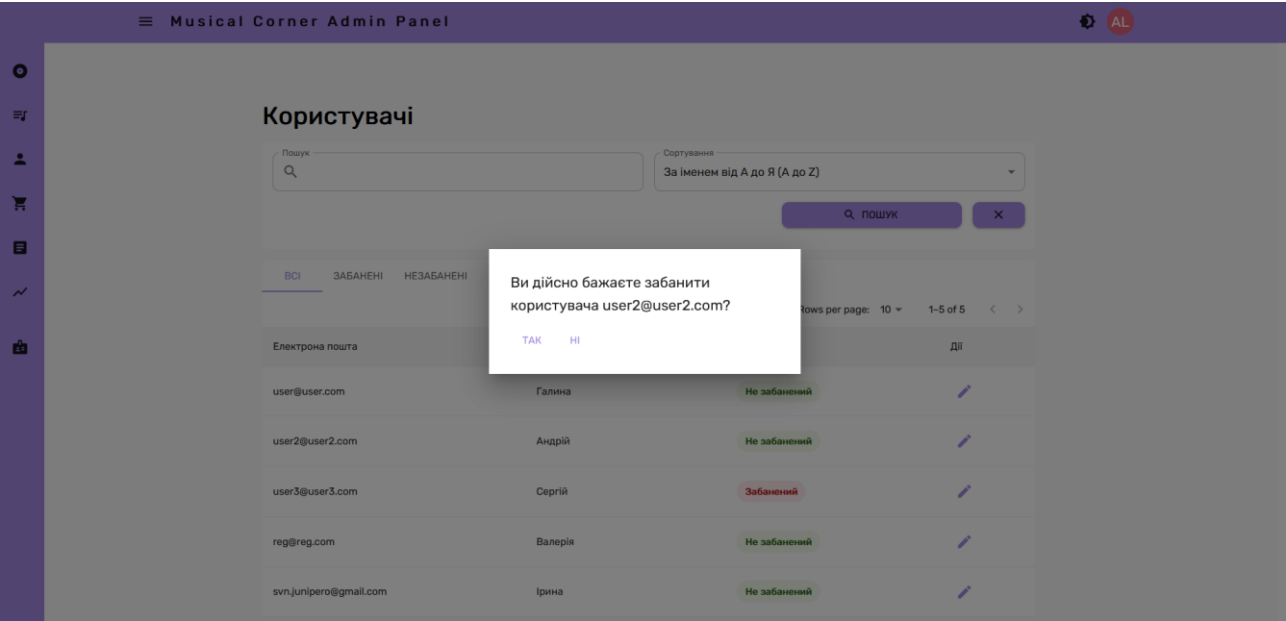


Рис. 3.6. – Модаль.

При натисканні кнопки «Так» на модалі, відповідний користувач буде забанений та оновлена інформація про нього з'явиться на сторінці (рис. 3.7).

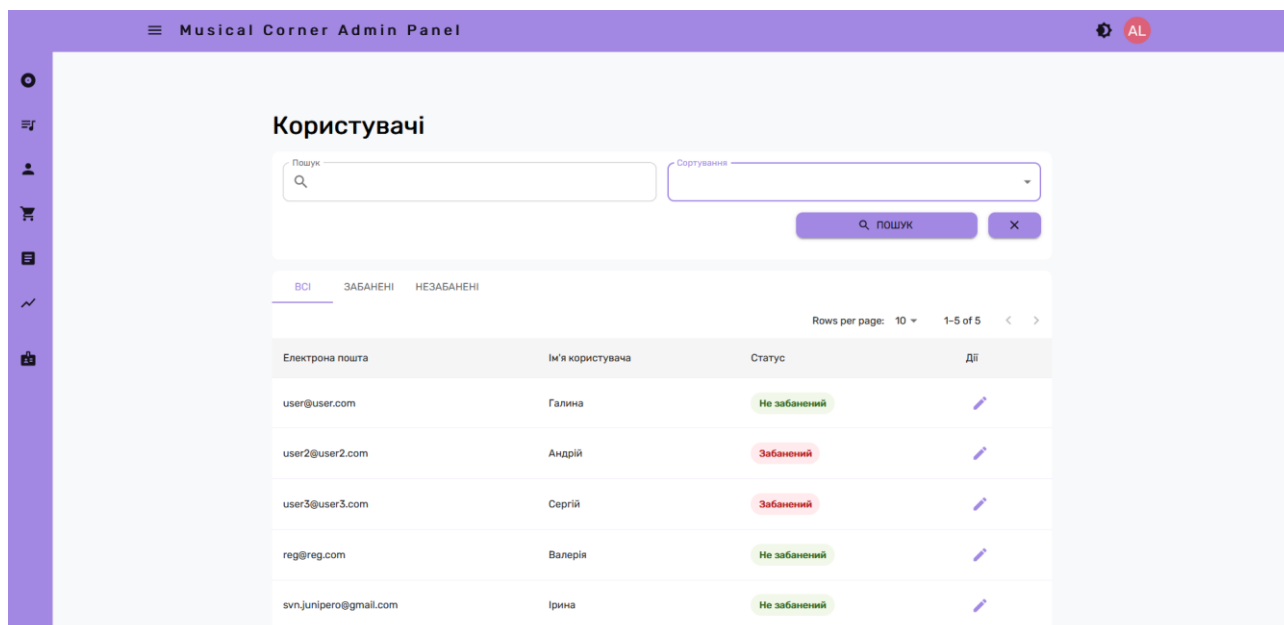


Рис. 3.7. – Оновлена інформація.

Далі можна перейти на сторінку замовлень завдяки натисканні відповідної кнопки в навігаційному меню (рис. 3.8).

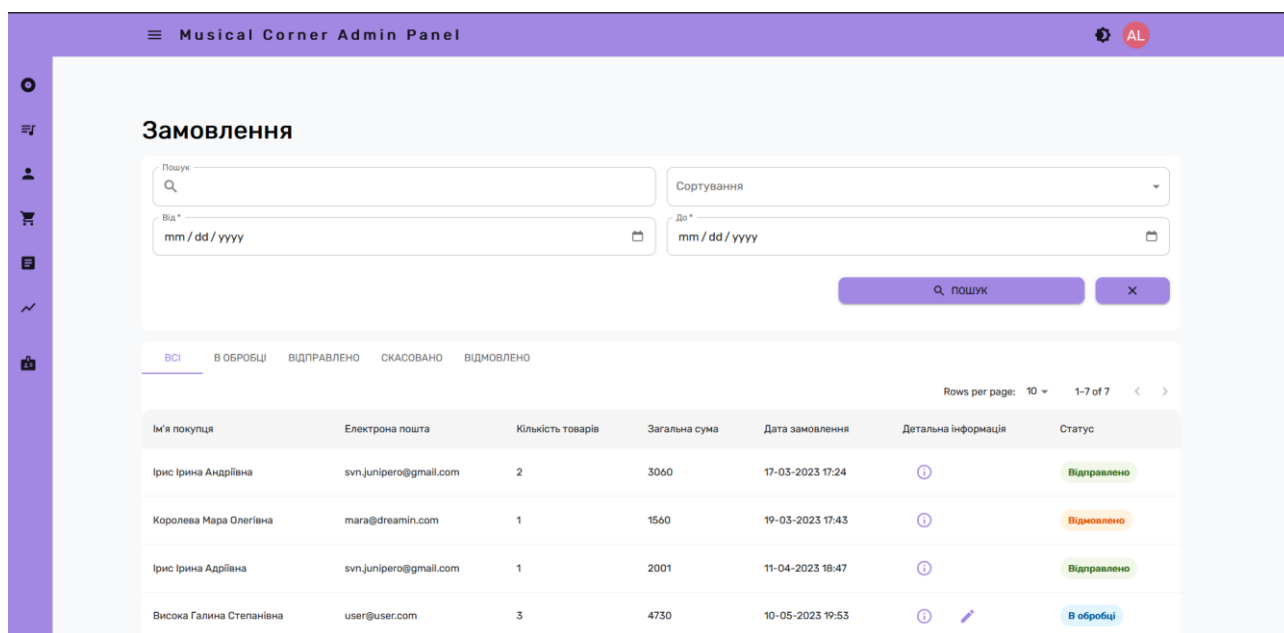


Рис. 3.8. – Сторінка з замовленнями.

На сторінці зображена таблиця з основною інформацією про всі замовлення.

Наявні також вкладки для перегляду відправлених та скасованих, а також тих, які знаходяться в обробці або яким було відмовлено. Наприклад, можна переглянути всі скасовані замовлення (рис. 3.9).

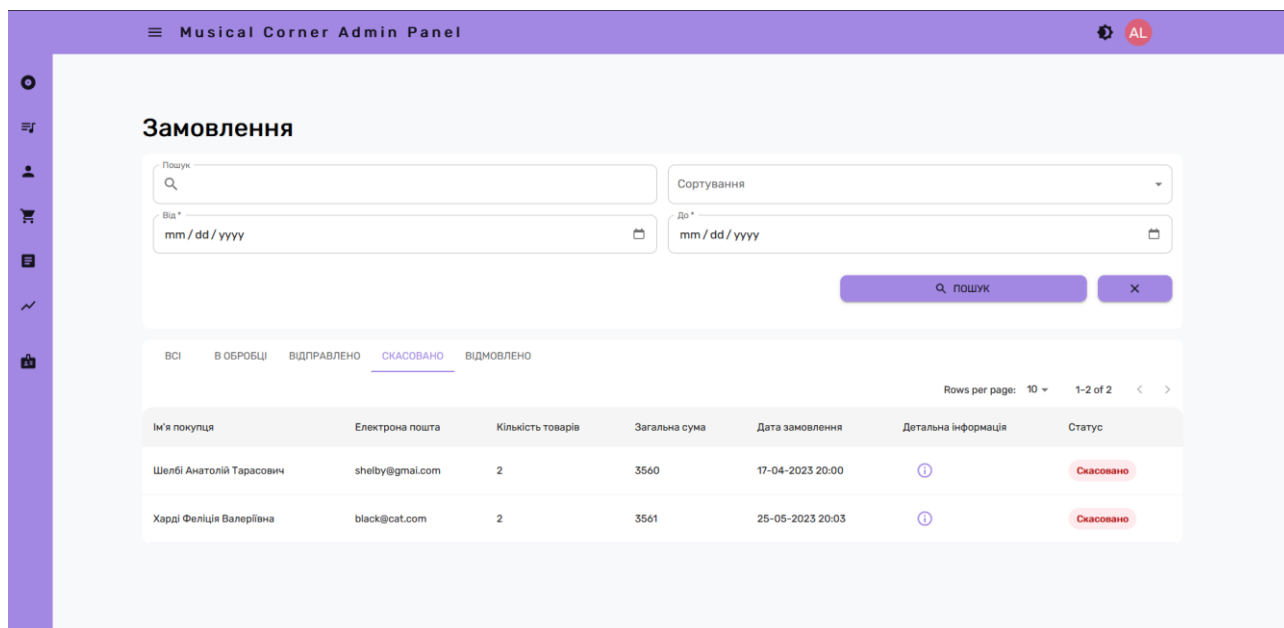


Рис. 3.9. – Скасовані замовлення.

Можливий також перехід на сторінку, де доступний перегляд детальної інформації про замовлення, при натисканні на відповідну кнопку інформації (рис. 3.10).

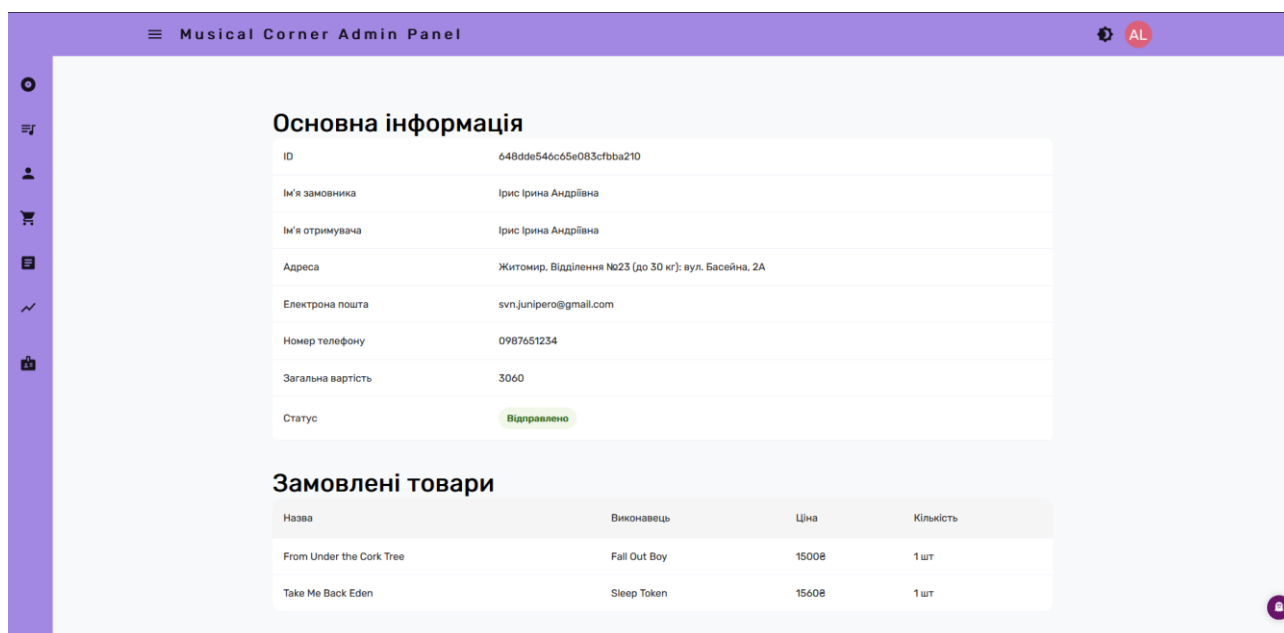


Рис. 3.10. – Інформація про замовлення.

Якщо замовлення має статус «В обробці», то можливе редагування його даних при натисканні на відповідну кнопку. Після цього відкриється форма, на якій можна буде змінити потрібні дані (рис. 3.11).

Оновлення інформації про замовлення

Загальна інформація

Електронна пошта покупця *
user@user.com

Адреса *
Львівська, Львів, Каїтуха, 35, 11, 10050

Прізвище та ім'я отримувача *
Висока Галина Степанівна

Прізвище та ім'я покупця *
Висока Галина Степанівна

Номер телефону *
0987654123

Статус:
В обробці

ВІДМІНИТИ Оновити

Рис. 3.11. – Редагування інформації про замовлення.

На сторінці зі списком замовлень також наявний блок, завдяки якому можна здійснити пошук за номером замовлення, відсортувати всі замовлення, або відфільтрувати за датою (рис. 3.12).

Замовлення

Пошук

Сортування

Від: 03 / 01 / 2023

До: 05 / 01 / 2023

Пошук X

ВСІ В ОБРОБЦІ ВІДПРАВЛЕНО СКАСОВАНО ВІДМОВЛЕНО

Rows per page: 10 1-4 of 4

Ім'я покупця	Електронна пошта	Кількість товарів	Загальна сума	Дата замовлення	Детальна інформація	Статус
Ірис Ірина Андріївна	svn.junipero@gmail.com	2	3060	17-03-2023 17:24	📄	Відправлено
Королева Мара Олегівна	marara@dreamin.com	1	1560	19-03-2023 17:43	📄	Відмовлено
Ірис Ірина Андріївна	svn.junipero@gmail.com	1	2001	11-04-2023 18:47	📄	Відправлено
Шелбі Анатолій Тарасович	shelby@gmail.com	2	3560	17-04-2023 20:00	📄	Скасовано

Рис. 3.12. – Відфільтровані замовлення за датою.

Завдяки навігаційному меню також можливий перехід на сторінку зі списком всіх новин (рис. 3.13).

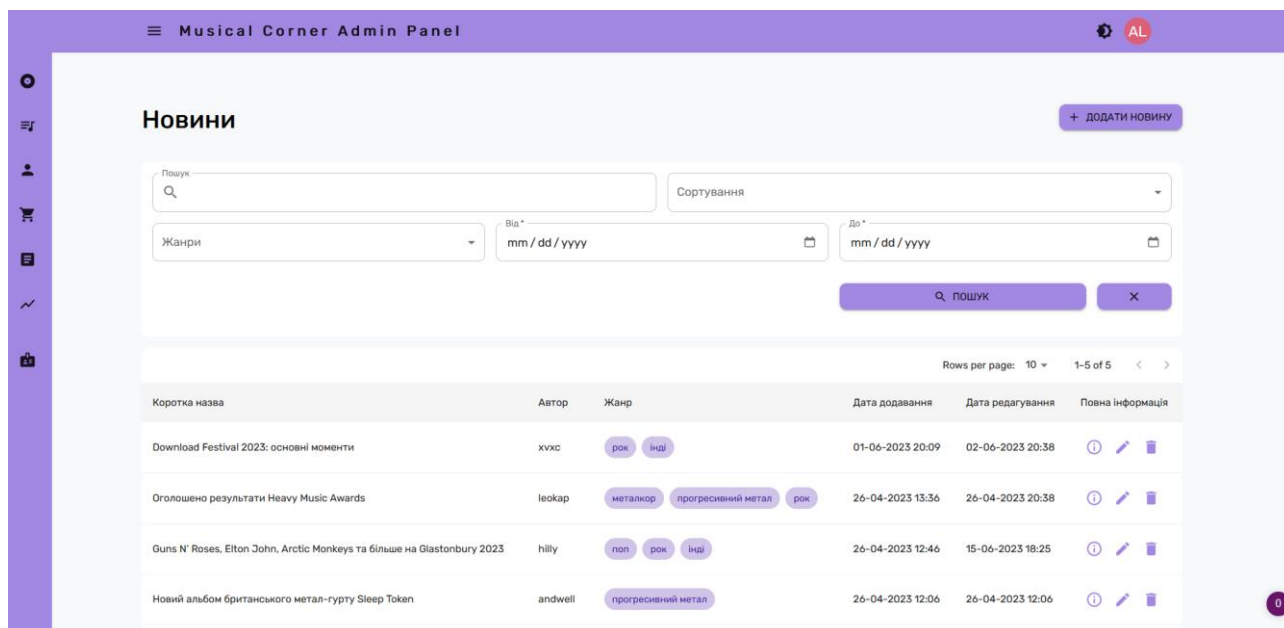


Рис. 3.13. – Сторінка з інформацією про новини.

На даній сторінці, як і на інших, присутній блок для пошуку, фільтрації та сортування даних, який працює абсолютно аналогічно.

Користувач може переглянути текст всієї новини, натиснувши на потрібну кнопку (рис. 3.14).

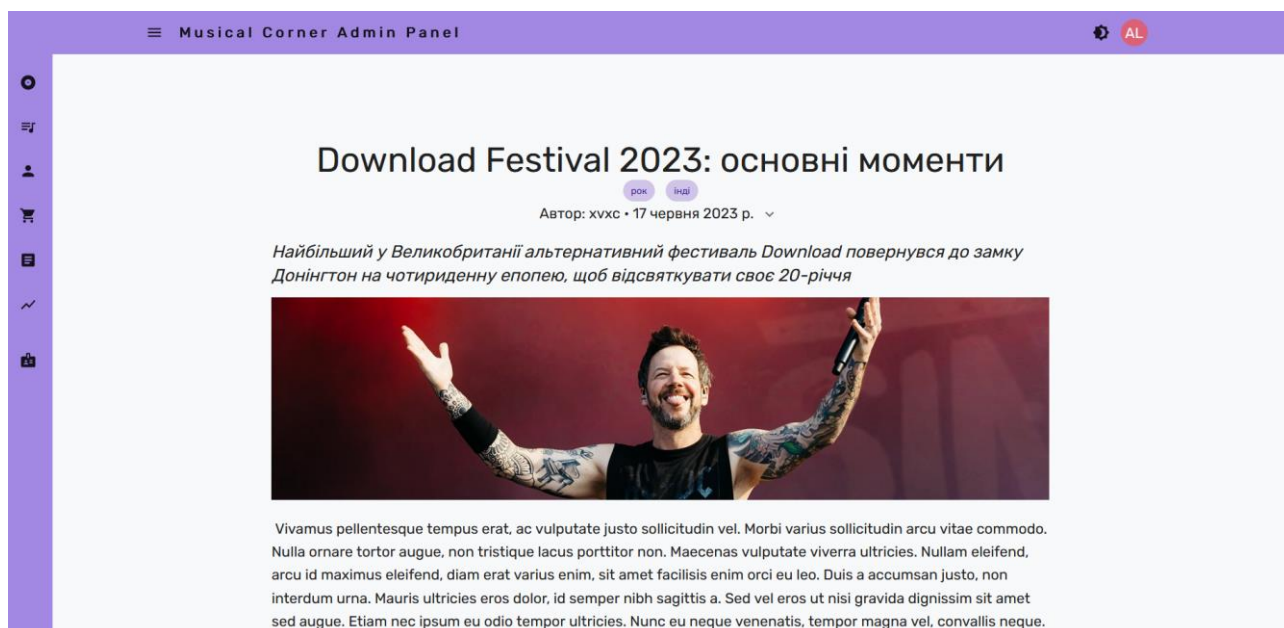


Рис. 3.14. – Сторінка текстом новини.

Є можливим додавання нової новини. Для цього необхідно натиснути відповідну кнопку, яка знаходиться у правому верхньому куті. Форма для створення нової новини та редагування існуючої є однаковою, відмінністю є

лише те, що при редагуванні новини форма заповнюється вже наявними даними (рис. 3.15).

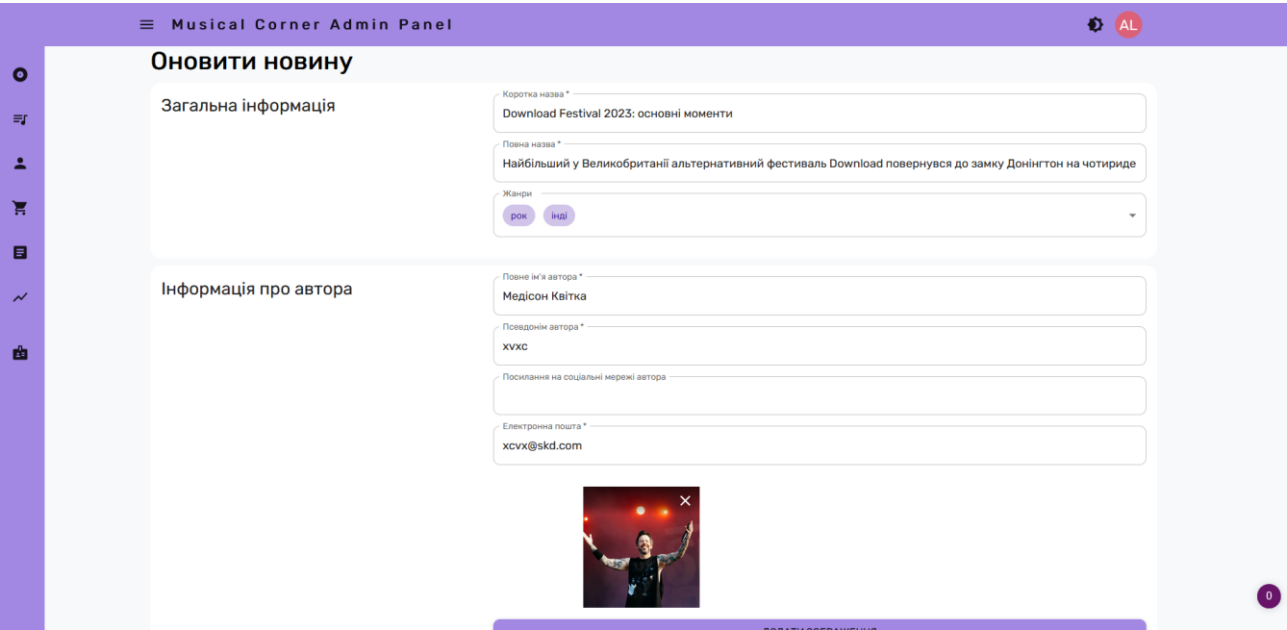


Рис. 3.15. – Редагування новини.

Видалення новини працює аналогічним чином як і заборона коментування для користувачів.

Користувач клієнтської частини при відвідування веб-додатку спершу побачить головну сторінку (рис. 3.16), на якій наявна карусель, а також останні додані платівки, випадково підібрані платівки, які можуть зацікавити користувача, а також останні додані новини (рис. 3.17).

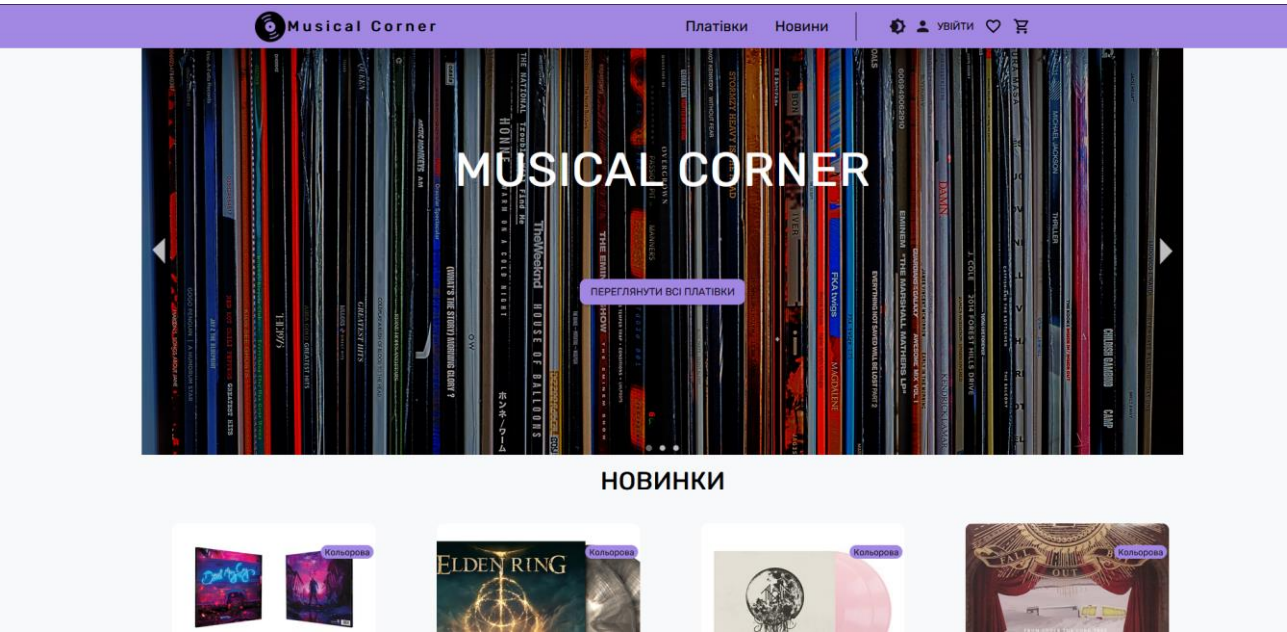


Рис. 3.16. – Головна сторінка сайту.

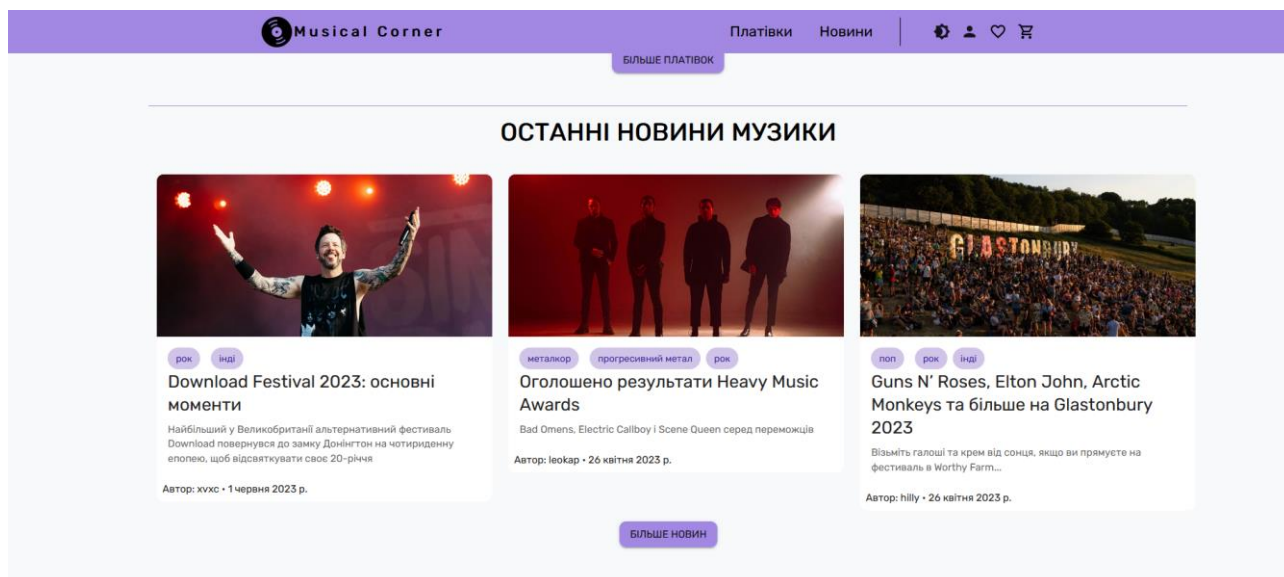


Рис. 3.17. – Останні новини.

Користувач матиме змогу увійти до вже створеного раніше профілю, натиснувши на відповідну кнопку у правій частині навігаційної панелі.

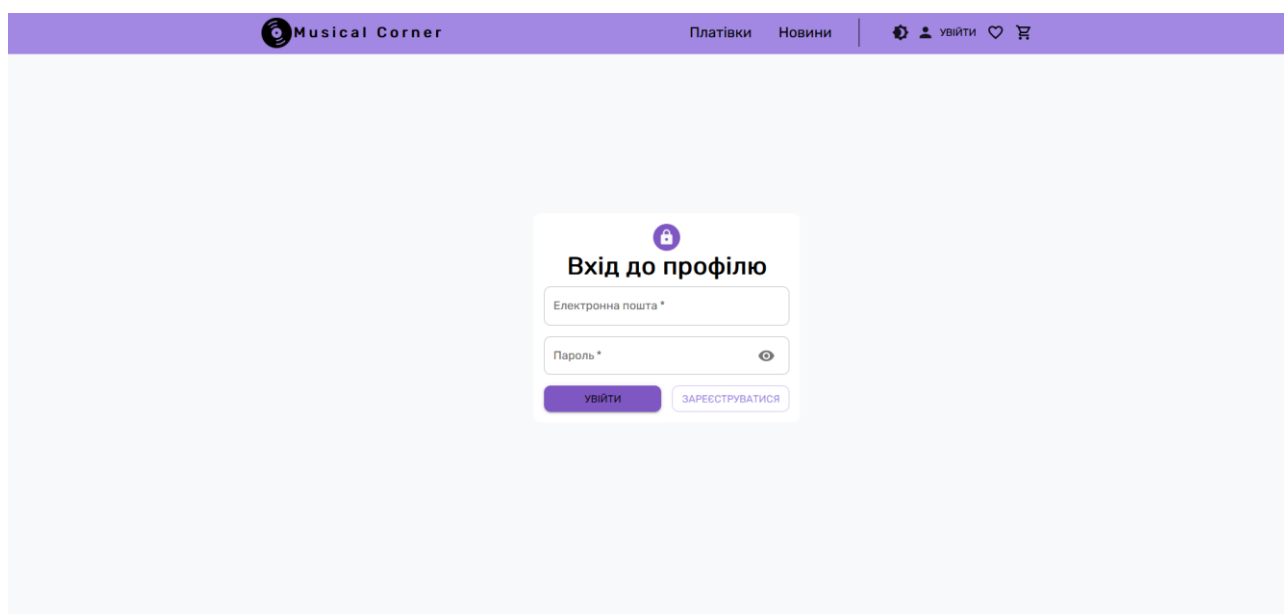


Рис. 3.17. – Вхід до профілю.

Якщо ж користувач є незареєстрованим, то достатньо натиснути кнопку «Зареєструватися», після чого з'являться поля для реєстрації у системі (рис. 3.18).

На формі наявні поля як обов'язкові для заповнення, так і не обов'язкові, але які полегшають оформлення замовлення.

Регістрація

Електронна пошта *

Пароль *

Ім'я *

Прізвище

По-батькові

Увійти Зареєструватися

Рис. 3.18. – Форма реєстрації у системі.

Після авторизації у системі, користувач матиме можливість переглянути власний профіль натиснувши на відповідну кнопку на навігаційній панелі.

Профіль користувача розділений на декілька вкладок. Вкладка «Загальна інформація» дозволяє перегляд інформації користувача, такої як ім'я, поштова адреса та адреса доставки (рис. 3.19).

Загальна інформація

Ім'я Галина

Прізвище Висока

По-батькові -

Електронна пошта user@user.com

Номер телефону -

Адреса

Область Львівська

Місто Львів

Вулиця Каїтуча, 35

Поштовий індекс 10050

РЕДАГУВАТИ

Рис. 3.19. – Профіль – Загальна інформація.

У цій вкладці також доступна кнопка «Редагувати», яка перенаправить користувача на сторінку, де можна редагувати інформацію власного профілю (рис. 3.20).

Рис. 3.20. – Оновлення інформації.

Вкладка «Мої замовлення» Відображає список замовлень користувача (рис. 3.21). У даній вкладці наявний блок для пошуку замовлення за номером, сортування та фільтрації за датою. Також можливе сортування за статусом.

Рис. 3.21. – Замовлення.

Вкладка «Бажані товари» відображає список бажаних товарів (рис. 3.22). В даному списку можна перейти на сторінку з повною інформацією про платівку, а також додати її до кошику. Карти з короткою інформацією про товар були виконані Демченко Ярославом.

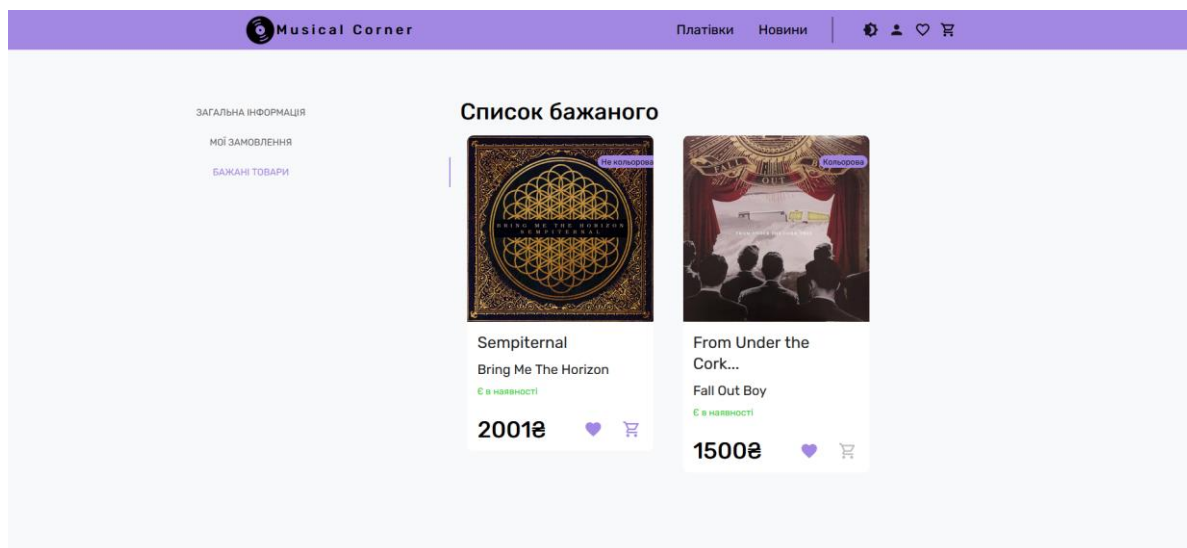


Рис. 3.22. – Бажані товари.

Перегляд власного списку бажаних товарів також доступний неавторизованим користувачам.

Після додавання товару у кошик є можливим оформити замовлення.

Форма для оформлення замовлення (рис. 3.23) складається з блоку, в якому необхідно вказати повне ім'я клієнта, номер телефону та поштову адресу. Наявна можливість вказати отримувача як іншу люди, після чого відкриється додатковий блок, де потрібно буде вказати інформацію отримувача.

Рис. 3.24. – Оформлення замовлення.

Наявний блок з платівками до замовлення, де вказана її назва, виконавець, ціна і кількість замовлених одиниць, а також загальна сума до сплати.

Блок адреси доставки надає можливість обрати доставку у відділення Нової пошти або доставку кур'єром.

Якщо замовлення оформлює авторизований користувач, то поля, для яких дані вже є наявними в системі, будуть автоматично заповнені. Після заповнення усіх полів варто натиснути кнопку «Оформити замовлення», яка перенаправить на сторінку оплати.

Для проведення оплати використовувалось API LiqPay, тому сторінка оплати використовує стандартний вигляд оплати через цю систему (рис. 3.25).

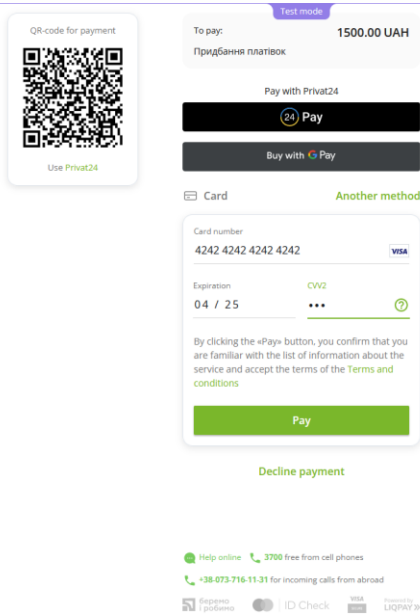


Рис. 3.25. – Оплата замовлення.

При правильному заповненні всіх даних картки, та натискання кнопки «Pay», користувача буде перенаправлено на сторінку з подякою за покупку (рис. 3.26).

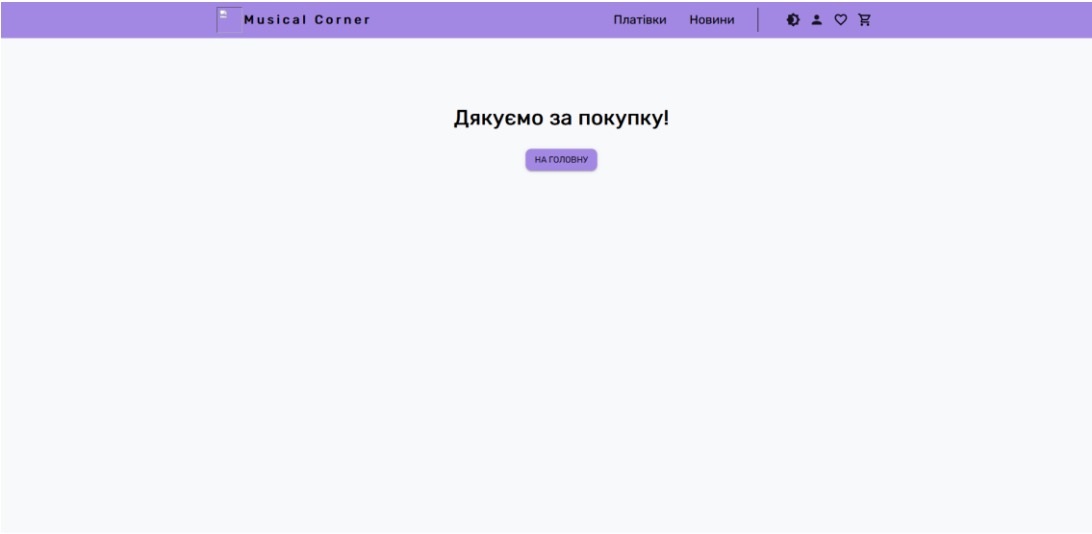


Рис. 3.26. – Подяка за успішне замовлення.

На навігаційній панелі також наявна кнопка «Новини», яка дозволить перейти на сторінку, де зображені всі новини та блок, який дозволить проводити пошук існуючих новин, фільтрацію та сортування (рис. 3.27).

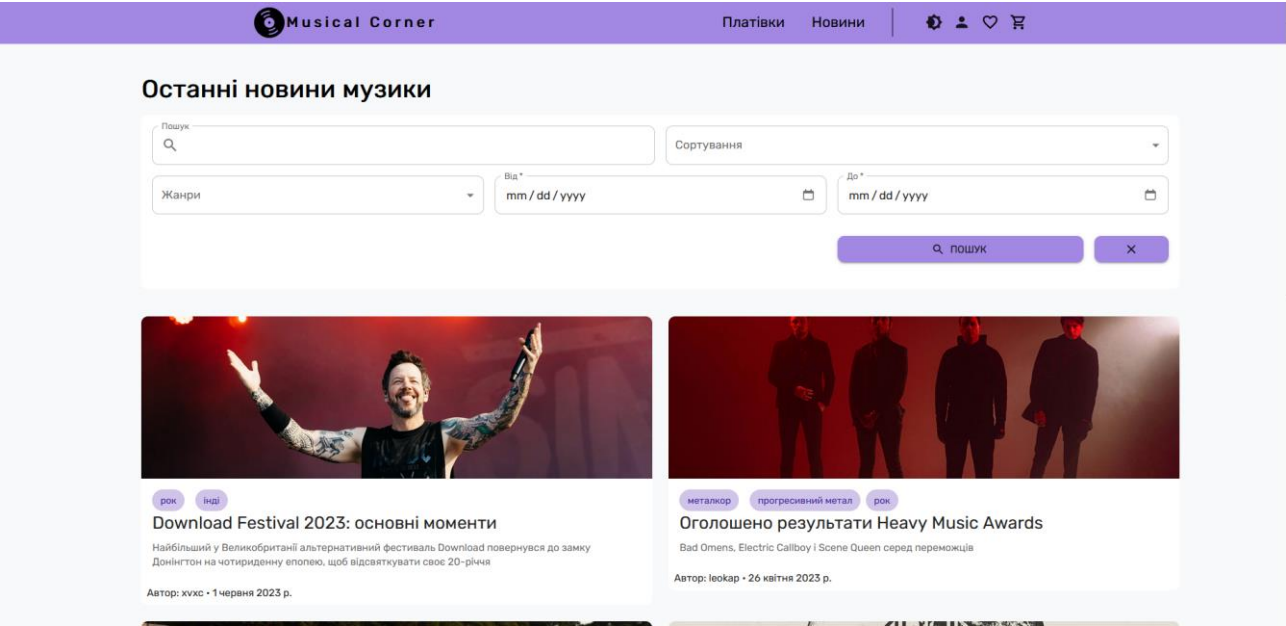


Рис. 3.27. – Сторінка з усіма новинами.

При натисканні на одну з новин відбудеться перехід на сторінку, де міститься повний текст новини. Під нею наявне поле для створення нового коментаря, якщо користувач автентифікований у системі, а також відображаються інші коментарі (рис. 3.28).

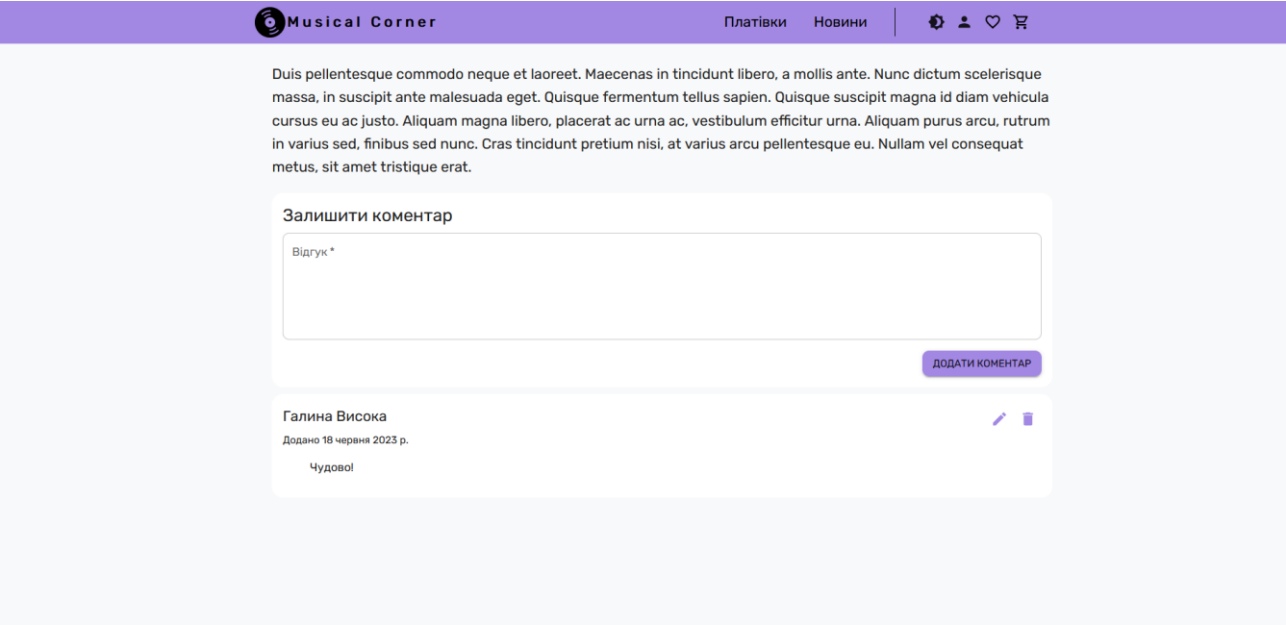


Рис. 3.28. – Коментарі до новини.

Користувач має можливість відредагувати або видалити власний коментар, шляхом натискання на відповідні кнопки.

3.3. Тестування роботи системи.

Перевірка на працездатність є необхідним процесом під час розробки програмного коду системи. Тестування веб-додатку, в якому насамперед проводиться купівля товарів, включає перевірку його функціональності, надійності та конкретності роботи.

Тестування веб-додатку включає такі основні види:

- тестування функціональності;
- тестування безпеки;
- тестування користувацького інтерфейсу;
- тестування сумісності.

Тестування функціональності. Нижче наведено деякі тест-кейси для перевірки роботи блоку новин.

Тест-кейс №1. Перегляд новини.

Опис: перевірка перегляду конкретної новини.

Кроки:

1. Перейти на сторінку з усіма новинами.
2. Натиснути на конкретну новину.

Очікуваний результат: відбудеться перенаправлення на сторінку конкретної новини.

Фактичний результат: відбулось перенаправлення на сторінку конкретної новини.

Тест-кейс №2. Додавання коментаря (автентифікований користувач).

Опис: перевірка додавання коментаря як автентифікований користувач.

Кроки:

1. Автентифікуватися в системі.
2. Перейти на сторінку новин.

3. Перейти на сторінку конкретної новини.
4. Заповнити поле «Коментар».
5. Натиснути кнопку «Додати коментар».

Очікуваний результат: відбудеться додавання коментаря, який відразу відобразиться на сторінці.

Фактичний результат: відбулося додавання коментаря, який відразу відобразився на сторінці.

Тестування на сумісність підтвердило коректне функціонування веб-додатку на різних пристроях та у різних браузерах, включаючи Edge, Google Chrome та Mozilla Firefox.

Під час виконання всіх видів тестування, всі виявлені проблеми та недоліки системи були усунені.

Висновки до розділу 3.

У цьому розділі були визначені вимоги щодо розгортання системи, наведений порядок її встановлення та рекомендовані браузери для ефективної взаємодії з веб-додатком.

Була представлена структура інтерфейсу веб-додатку та проілюстрована робота з системою, а саме робота з адміністративною та клієнтською частиною, які дають змогу взаємодіяти з зареєстрованими користувачами, замовленнями та новинами.

Також було проведено функціональне тестування основних частин системи, а також тестування на сумісність. Виявлені проблеми та недоліки під час тестування були усунені.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було спроектовано та реалізовано веб-додаток магазину вінілових платівок «Musical Corner», а саме функціонал користувачів, замовлень та новин.

У першому розділі, після аналізу предметної області, були описані основні аспекти веб-додатку. Аналіз існуючих програмних аналогів дозволив визначити основні функції, необхідні для функціонування системи. Було обрано архітектуру веб-додатку, а саме клієнт-серверну архітектуру, яка задовольнить вимоги до роботи системи. Відповідно до обраної архітектури було визначено інструментальні засоби для розробки, а саме середовище Node.js, Express.js для розробки серверної частини, MongoDB як базу даних, React як бібліотеку для створення інтерфейсу користувача на клієнті.

У другому розділі було здійснено розподіл обов'язків з виконання проєкту та налагоджено роботу у групі, описано функціональні вимоги користувача до роботи веб-додатку, а саме частин, які відповідають за роботу з користувачами, замовленнями та новинами. Було описано також нефункціональні вимоги до системи. На основі вимог користувача було створено діаграми, які ілюструють варіанти використання системи. Далі було розроблено та детально описано структуру файлової системи. Було детально описано структуру бази даних та її колекції і документів. Було реалізовано функціонал веб-додатку, детально описано основні та важливі функції додатку, пояснено алгоритм їх дії на прикладах.

У третьому розділі було визначено вимоги щодо розгортання системи, наведений порядок її встановлення та рекомендовані браузері для комфортної роботи з веб-додатком. Була представлена структура інтерфейсу системи та проілюстрована робота з нею, а саме робота з адміністративною та клієнтською частинами, які надають можливість взаємодіяти з функціями користувачів, замовлень та новин. Також було проведено функціональне тестування основних частин веб-додатку, а також тестування на сумісність. Всі виявлені помилки роботи були вирішені.

					ІПЗ.КР.Б – 121 – 23 – ПЗ	
						68

Реалізований програмний продукт готовий до використання. Веб-додаток може бути використаний для перегляду та купівлі вінілових платівок, а також перегляду новин світу музики та управління контентом системи.

В подальшому даний веб-додаток може бути розширений шляхом додавання та впровадження нових функціональних можливостей. Завдяки обраній архітектура розширення веб-системи буде відбуватися значно простіше та швидше.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Li Cohen. Vinyl record sales top CDs for the first time in more than 30 years – CBS News [Електронний ресурс] / Li Cohen – 10 березня 2023 – Режим доступу до ресурсу: <https://www.cbsnews.com/news/vinyl-record-sales-top-cds-first-time-since-1987/>.
2. Joana Almeida. Node js vs. PHP: An In-Depth Comparison Guide for Web Development – DistantJob [Електронний ресурс] / Joana Almeida – 9 червня 2022 – Режим доступу до ресурсу: <https://distantjob.com/blog/nodejs-vs-php/>.
3. Tejas Kaneriya. 12 Best Nodejs Frameworks for App Development in 2023 – Simform [Електронний ресурс] / Tejas Kaneriya – 14 травня 2023 – Режим доступу до ресурсу: <https://www.simform.com/blog/best-nodejs-frameworks/>.
4. Mongoose Documentation. Schemas [Електронний ресурс] / Mongoose Documentation – Режим доступу до ресурсу: <https://mongoosejs.com/docs/guide.html>
5. React Documentation. useState [Електронний ресурс] / React Documentation – Режим доступу до ресурсу: <https://react.dev/reference/react/useState>.
6. React Documentation. useEffect [Електронний ресурс] / React Documentation – Режим доступу до ресурсу: <https://react.dev/reference/react/useEffect>
7. Redux Documentation. Redux Fundamentals, Part 3: State, Actions, and Reducers [Електронний ресурс] / Redux Documentation – Режим доступу до ресурсу: <https://redux.js.org/tutorials/fundamentals/part-3-state-actions-reducers>.
8. LiqPay API Documentation. Checkout [Електронний ресурс] / LiqPay API Documentation – Режим доступу до ресурсу: <https://www.liqpay.ua/documentation/api/acquiring/checkout/doc>.
9. LiqPay API Documentation. Правила формування запиту на проведення платежу [Електронний ресурс] / LiqPay API Documentation – Режим доступу до ресурсу: https://www.liqpay.ua/documentation/data_signature.

10. Нова Пошта – Портал розробників. Довідник міст компанії [Електронний ресурс] / Нова Пошта – Портал розробників – Режим доступу до ресурсу: <https://developers.novaposhta.ua/view/model/a0cf0f5f-8512-11ec-8ced-005056b2dbe1/method/a1e6f0a7-8512-11ec-8ced-005056b2dbe1>.
11. Нова Пошта – Портал розробників. Довідник відділень та поштоматів [Електронний ресурс] / Нова Пошта – Портал розробників – Режим доступу до ресурсу: <https://developers.novaposhta.ua/view/model/a0cf0f5f-8512-11ec-8ced-005056b2dbe1/method/a2322f38-8512-11ec-8ced-005056b2dbe1>.
12. TinyMCE Documentation. TinyMCE React integration technical reference [Електронний ресурс] / TinyMCE Documentation – Режим доступу до ресурсу: <https://www.tiny.cloud/docs/tinymce/6/react-ref/#installing-the-tinymce-react-integration-using-npm-or-yarn>
13. Moment.js. Format Dates [Електронний ресурс] / Moment.js – Режим доступу до ресурсу: <https://momentjs.com/>

ДОДАТКИ

					ІІЗ.КР.Б – 121 – 23 – ІІЗ	
						72

Файл для створення моделі користувача

```

const mongoose = require("mongoose");
const uniqueValidator = require("mongoose-unique-validator");
const validator = require("validator");
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");

let userSchema = new mongoose.Schema({
  email: {
    type: String,
    required: [true, 'Поле "Емейл" не може бути порожнім'],
    unique: true,
    lowercase: true,
  },
  password: {
    type: String,
    required: [true, 'Поле "Пароль" не може бути порожнім'],
    validate(value) {
      if (value.length < 7) {
        throw new Error("Пароль має містити менше 7 символів");
      }
      if (validator.contains(value, "password", { ignoreCase: true })) {
        throw new Error('Пароль не може містити слово: "password"');
      }
    },
    trim: true,
  },
  fullName: {
    firstName: {
      type: String,
      required: [true, 'Поле "Ім\''я" не може бути порожнім'],
      trim: true,
    },
    lastName: {
      type: String,
      trim: true,
    },
    patronymic: {
      type: String,
      trim: true,
    },
  },
  favourites: {
    type: [mongoose.Schema.Types.ObjectId],
    ref: "Product",
  },
  address: {
    region: {

```

```

        type: String,
        trim: true,
    },
    city: {
        type: String,
        trim: true,
    },
    street: {
        type: String,
        trim: true,
    },
    apartment: { type: String,
        trim: true },
    zipCode: {
        type: String,
        trim: true,
    },
},
},
phoneNumber: {
    type: String,
    trim: true,
    validate(value) {
        if (value.length != 10) {
            throw new Error("Невірний номер телефону (формат: 0123456789)");
        }
        if (value[0] != 0) {
            throw new Error("Невірний номер телефону (формат: 0123456789)");
        }
        if (!validator.isNumeric(value, {no_symbols: true})) {
            throw new Error("Невірний номер телефону (вказано не числа)")
        }
    },
},
},
isBanned: {
    type: Boolean,
    default: false,
},
tokens: [
    {
        token: {
            type: String,
            required: true,
        },
    },
],
});

userSchema.virtual("articleComments", {
    ref: "ArticleComment",
    localField: "_id",
    foreignField: "userId",

```

```

});

userSchema.virtual("productComments", {
  ref: "ProductComment",
  localField: "_id",
  foreignField: "userId",
});

userSchema.virtual("orders", {
  ref: "Order",
  localField: "_id",
  foreignField: "userId",
});

userSchema.pre("save", async function (next) {
  const user = this;
  if (user.isModified("password")) {
    user.password = await bcrypt.hash(user.password, 8);
  }
  next();
});

userSchema.pre("findOneAndUpdate", async function (next) {
  const user = this;
  if (user._update.password) {
    user._update.password = await bcrypt.hash(user._update.password, 8);
  }
  next();
});

userSchema.statics.findOneByCredentials = async (email, password) => {
  const user = await User.findOne({ email });
  if (!user) {
    throw new Error("Користувача з такою електронною поштою не існує");
  }
  const isMatch = await bcrypt.compare(password, user.password);
  if (!isMatch) {
    throw new Error("Невірний пароль");
  }
  return user;
};

userSchema.statics.comparePasswords = async (id, password) => {
  const user = await User.findById(id);
  if (!user) {
    throw new Error("Невірний користувача");
  }
  const isMatch = await bcrypt.compare(password, user.password);
  if (!isMatch) {
    return false;
  }
}

```

```

    return true;
};

userSchema.methods.generateAuthToken = async function () {
    const user = this;
    const token = jwt.sign({ _id: user._id.toString() }, "kdweueksdsjko");
    user.tokens = user.tokens.concat({ token });
    await user.save();
    return token;
};

userSchema.methods.toJSON = function () {
    const user = this;
    const userObject = user.toObject();
    delete userObject.password;
    delete userObject.tokens;
    return userObject;
};

userSchema.plugin(uniqueValidator, {
    message: "Користувач з такою електронною адресою вже існує",
});

userSchema.index({ "$*": "text" });

const User = mongoose.model("User", userSchema);

module.exports = User;

```

Файл сторінка, яка дозволяє створити замовлення

```

const initialState = {
  products: [{ id: "", count: "" }],
  totalPrice: 0,
  email: "",
  address: "",
  recipientName: "",
  customerName: "",
  phoneNumber: "",
  orderId: "",
};

const initialClientState = {
  firstName: "",
  lastName: "",
  patronymic: "",
};

const orderId = Math.floor(1 + Math.random() * 900000000);

const OPTIONS_LIMIT = 50;
const defaultFilterOptions = createFilterOptions();

const filterOptions = (options, state) => {
  return defaultFilterOptions(options, state).slice(0, OPTIONS_LIMIT);
};

const novaPoshtaURL = "https://api.novaposhta.ua/v2.0/json/";
const NOVA_POSHTA_API = process.env.REACT_APP_NOVAPOSHTA_API_KEY;
const LIQPAY_PRIVATE_KEY = process.env.REACT_APP_LIQPAY_PRIVAT_KEY;
const LIQPAY_PUBLIC_KEY = process.env.REACT_APP_LIQPAY_PUBLIC_KEY;

const fetchDepartmentByCity = (city) => {
  const data = {
    apiKey: NOVA_POSHTA_API,
    modelName: "Address",
    calledMethod: "getWarehouses",
    methodProperties: {
      CityName: city,
      TypeOfWarehouseRef: "841339c7-591a-42e2-8233-7a0a00f0ed6f",
    },
  };
  return data;
};

export default function Order() {
  const { state } = useLocation();
  const dispatch = useDispatch();

```

```

const cartRecords = useSelector((state) => state.records.cartRecords);
const user = JSON.parse(localStorage.getItem("profile"))?.user;
const [checked, setChecked] = useState(false);
const [formData, setFormData] = useState(initialState);
const [recipientData, setRecipientData] = useState(initialClientState);
const [customerData, setCustomerData] = useState(initialClientState);
const [shippingType, setShippingType] = useState("courier");
const [customerAddress, setCustomerAddress] = useState({
  region: "",
  city: "",
  street: "",
  apartment: "",
  zipCode: "",
});

const [areasList, setAreasList] = useState([]);
const [city, setCity] = useState("");
const [departmentList, setDepartmentList] = useState([]);
const [departmentAddress, setDepartmentAddress] = useState("");

let json_string = JSON.stringify({
  public_key: LIQPAY_PUBLIC_KEY,
  version: "3",
  action: "pay",
  amount: state.totalPrice,
  currency: "UAH",
  description: "Придбання платівок",
  order_id: orderId,
  result_url: "http://localhost:3001/order/thankyou",
});

const data = json_string.toString(Base64);

const sign_string = LIQPAY_PRIVATE_KEY + data + LIQPAY_PRIVATE_KEY;

const signature = sha1(sign_string).toString(Base64);

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

useEffect(() => {
  fetch(novaPoshtaURL, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      apiKey: NOVA_POSHTA_API,

```

```

        modelName: "Address",
        calledMethod: "getCities",
        methodProperties: {},
    })),
    })
    .then((response) => response.json())
    .then((response) => setAreasList(response.data));
}, []);

useEffect(() => {
    if (user) {
        setCustomerData({ ...user.fullName });
        formData.email = user.email;
        formData.phoneNumber = user.phoneNumber;
        formData.userId = user._id;
        setCustomerAddress({...user.address})
    }
}, []);

useEffect(() => {
    if (city) {
        fetch(novaPoshtaURL, {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({
                apiKey: NOVA_POSHTA_API,
                modelName: "Address",
                calledMethod: "getWarehouses",
                methodProperties: {
                    CityName: city,
                    TypeOfWarehouseRef: "841339c7-591a-42e2-8233-7a0a00f0ed6f",
                },
            }),
        })
        .then((response) => response.json())
        .then((response) => setDepartmentList(response.data));
    }
}, [city]);

const handleCheckboxChange = (e) => {
    setChecked(e.target.checked);
};

const handleRecipientChange = (e) => {
    setRecipientData({ ...recipientData, [e.target.name]: e.target.value });
};

const handleCustomerChange = (e) => {
    setCustomerData({ ...customerData, [e.target.name]: e.target.value });
};

```

```

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

const handleAutocompleteChange = (event, newValue) => {
  setCity(newValue);
};

const handleDepartmentAddressChange = (event) => {
  setDepartmentAddress(event.target.value);
};

const handleCustomerAddressChange = (e) => {
  setCustomerAddress({ ...customerAddress, [e.target.name]: e.target.value });
};

const handleSubmit = async (e) => {
  formData.customerName =
    customerData.lastName +
    " " +
    customerData.firstName +
    " " +
    customerData.patronymic;

  if (!checked) {
    formData.recipientName = formData.customerName;
  } else {
    formData.recipientName =
      recipientData.lastName +
      " " +
      recipientData.firstName +
      " " +
      recipientData.patronymic;
  }
  formData.products = state.records;
  formData.totalPrice = state.totalPrice;
  if (shippingType === "post") {
    formData.address = city + ", " + departmentAddress;
  } else {
    formData.address =
      customerAddress.region +
      ", " +
      customerAddress.city +
      ", " +
      customerAddress.street +
      ", " +
      customerAddress.apartment +
      ", " +
      customerAddress.zipCode;
  }
  formData.orderId = orderId;
};

```

```

    localStorage.setItem("orderData", JSON.stringify(formData));
  };

  return (
    <>
      <form
        autoComplete="off"
        onSubmit={handleSubmit}
        method="POST"
        action="https://www.liqpay.ua/api/3/checkout"
        accept-charset="utf-8"
      >
        <Typography variant="h4">Замовлення № {orderId}</Typography>
        <Grid container spacing={2} sx={{ mb: 2, mt: 2 }}>
          <Grid item xs={12} sm={8} md={8}>
            <Paper sx={{ p: 2 }}>
              <Typography variant="h4">Контактні дані</Typography>
              <Grid container spacing={2} sx={{ mb: 2, mt: 2 }}>
                <Grid item xs={12} sm={4} md={4}>
                  <Input
                    name="firstName"
                    label="Ім'я"
                    handleChange={handleCustomerChange}
                    type="text"
                    value={customerData.firstName || ""}
                  />
                </Grid>
                <Grid item xs={12} sm={4} md={4}>
                  <Input
                    name="lastName"
                    label="Прізвище"
                    handleChange={handleCustomerChange}
                    type="text"
                    value={customerData.lastName || ""}
                  />
                </Grid>
                <Grid item xs={12} sm={4} md={4}>
                  <Input
                    name="patronymic"
                    label="По-батькові"
                    handleChange={handleCustomerChange}
                    type="text"
                    value={customerData.patronymic || ""}
                  />
                </Grid>
              </Grid>
            </Paper>
          </Grid>
          <Grid container spacing={2} sx={{ mb: 2 }}>
            <Grid item xs={12} sm={6} md={6}>
              <Input

```

```

        name="phoneNumber"
        label="Номер телефону"
        handleChange={handleFormDataChange}
        type="text"
        value={formData.phoneNumber || ""}
      />
    </Grid>
    <Grid item xs={12} sm={6} md={6}>
      <Input
        name="email"
        label="Електронна пошта"
        handleChange={handleFormDataChange}
        type="email"
        value={formData.email || ""}
      />
    </Grid>
  </Grid>
  <FormControlLabel
    label="Отримувач інша людина"
    control={
      <Checkbox
        checked={checked}
        onChange={handleCheckboxChange}
        inputProps={{ "aria-label": "controlled" }}
      />
    }
  />
</Paper>
{checked ? (
  <Paper sx={{ p: 2, mt: 2 }}>
    <Typography variant="h4">Контактні дані отримувача</Typography>
    <Grid container spacing={2} sx={{ mb: 2, mt: 2 }}>
      <Grid item xs={12} sm={4} md={4}>
        <Input
          name="firstName"
          label="Ім'я"
          handleChange={handleRecipientChange}
          type="text"
        />
      </Grid>
      <Grid item xs={12} sm={4} md={4}>
        <Input
          name="lastName"
          label="Прізвище"
          handleChange={handleRecipientChange}
          type="text"
        />
      </Grid>
      <Grid item xs={12} sm={4} md={4}>
        <Input
          name="patronymic"

```

```

        label="По-батькові"
        handleChange={handleRecipientChange}
        type="text"
      />
    </Grid>
  </Grid>

  <Grid container spacing={2} sx={{ mb: 2 }}>
    <Grid item xs={12} sm={6} md={6}>
      <Input
        name="phoneNumber"
        label="Номер телефону"
        handleChange={handleFormDataChange}
        type="text"
      />
    </Grid>
  </Grid>
</Paper>
) : (
  <></>
)}

<Paper sx={{ p: 2, mt: 2 }}>
  <Typography variant="h4">Доставка</Typography>
  <FormControl>
    <RadioGroup
      row
      value={shippingType}
      onChange={(e) => {
        setShippingType(e.target.value);
      }}
    >
      <FormControlLabel
        value="courier"
        control={<Radio />}
        label="Доставка кур'єром до дверей"
      />
      <FormControlLabel
        value="post"
        control={<Radio />}
        label="Доставка на відділення Нової Пошти"
      />
    </RadioGroup>
  </FormControl>
  {shippingType === "post" ? (
    <Grid container spacing={2} sx={{ mt: 2, mb: 2 }}>
      <Grid item xs={12} sm={6} md={6}>
        <Autocomplete
          filterOptions={filterOptions}
          value={city}
          required

```

```

        onChange={handleAutocompleteChange}
        id="controllable-states-demo"
        options={areasList.map(
          (option, index) => option.Description
        )}
        renderInput={({params} => (
          <TextField {...params} label="Населений пункт" />
        ))}
      />
    </Grid>
    <Grid item xs={12} sm={6} md={6}>
      {city ? (
        <>
          {departmentList.length > 0 ? (
            <FormControl fullWidth sx={{ mb: 2 }}>
              <InputLabel>Відділення</InputLabel>
              <Select
                onChange={handleDepartmentAddressChange}
                value={departmentAddress}
                required
                input={
                  <OutlinedInput
                    id="select-multiple-chip"
                    label="Відділення"
                  />
                }
              />
            </Select>
          ) : (
            <Typography>немає</Typography>
          )}
        </>
      ) : (
        <></>
      )}
    </Grid>
  </Grid>
) : (
  <Grid container spacing={2} sx={{ mt: 2, mb: 2 }}>
    <Grid item xs={12} sm={6} md={6}>

```

```

        <Input
          name="region"
          label="Область"
          handleChange={handleCustomerAddressChange}
          type="text"
          value={customerAddress.region || ""}
        />
      </Grid>
      <Grid item xs={12} sm={6} md={6}>
        <Input
          name="city"
          label="Місто"
          handleChange={handleCustomerAddressChange}
          type="text"
          value={customerAddress.city || ""}
        />
      </Grid>
      <Grid item xs={12} sm={6} md={6}>
        <Input
          name="street"
          label="Вулиця"
          handleChange={handleCustomerAddressChange}
          type="text"
          value={customerAddress.street || ""}
        />
      </Grid>
      <Grid item xs={12} sm={3} md={3}>
        <Input
          name="apartment"
          label="Квартира"
          handleChange={handleCustomerAddressChange}
          type="text"
          value={customerAddress.apartment || ""}
        />
      </Grid>
      <Grid item xs={12} sm={3} md={3}>
        <Input
          name="zipCode"
          label="Поштовий індекс"
          handleChange={handleCustomerAddressChange}
          type="text"
          value={customerAddress.zipCode || ""}
        />
      </Grid>
    </Grid>
  )}
</Paper>
</Grid>

<Grid item xs={12} sm={4} md={4}>
  <Paper sx={{ p: 2, mb: 2 }}>

```

```

<Typography variant="h4">Товари для замовлення</Typography>
{cartRecords &&
  cartRecords.map((product, index) => (
    <>
      <OrderCartItem
        product={product}
        cart={state.records[index]}
      />{" "}
      <Divider variant="middle" />
    </>
  )))
<Stack
  direction="row"
  justifyContent="space-between"
  alignItems="center"
  spacing={2}
  sx={{ mt: 1, p: 2 }}
>
  <Typography>До оплати:</Typography>
  <Typography>{state.totalPrice + "₾"}</Typography>
</Stack>
</Paper>
<Stack
  direction="row"
  spacing={2}
  justifyContent="flex-end"
  alignItems="center"
>
  <input type="hidden" name="data" value={data} />
  <input type="hidden" name="signature" value={signature} />
  {(shippingType === "post" && departmentAddress) ||
  (shippingType === "courier" && customerAddress) ? (
    <Button type="submit" variant="contained" color="primary">
      {" "}
      Оформити замовлення
    </Button>
  ) : (
    <></>
  )}
</Stack>
</Grid>
</Grid>
</form>
</>
);
}

```