МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ
КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)
(рівень вищої освіти)

на тему **Створення онлайн-магазину побутових товарів з використання React і NodeJS**

Виконав: студент 4 курсу, групи 6.1210-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
К. М. Коваленко
(ініціали та прізвище)
Керівник к.ф-м.н., доцент, доцент кафедри ЕІС та ПЗ
В.І. Попівщий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дісітел»
П. О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя

2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра Електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти перший(бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
(код та назва)
Освітня програма Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри Т. В. Критська
“ 01 ” 2024 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Коваленку Кирилу Максимовичу
(прізвище, ім’я, по батькові)

1.Тема роботи Створення онлайн-магазину побутових товарів з використання React і NodeJS
керівник роботи Попівщий Василій Іванович, к.т.н., доцент
(прізвище ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від 26.12.2023 № 2215-с

2. Строк подання студентом кваліфікаційної роботи
3. Вихідні дані кваліфікаційної роботи бакалавра
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
слайдів презентації

6. Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.03.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів бакалаврської роботи	Примітка
1	Аналіз існуючих рішень	03.01 – 25.01	виконано
2	Постановка завдання	26.01 – 29.01	виконано
3	Проектування архітектури	30.01 – 07.02	виконано
4	Розробка БД	08.02 – 22.02	виконано
5	Розробка серверної частини	23.02 – 07.03	виконано
6	Розробка клієнтської частини	08.03 – 22.03	виконано
7	Інтеграція компонентів	23.03 – 06.04	виконано
8	Тестування і оптимізація	07.04 – 21.04	виконано
9	Деплой системи	22.04 – 29.05	виконано

Студент _____ Коваленко М.К.
(підпис) (прізвище та ініціали)
Керівник роботи _____ Попівщий В.І.
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено
Нормоконтролер _____ Скрипник І.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок — 82

Рисунків — 18

Таблиць — 1

Джерел — 9

Коваленко К. М. Створення онлайн-магазину побутових товарів з використання React і NodeJS : кваліфікаційна робота бакалавра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник В. І. Попівщий. Запоріжжя : ЗНУ, 2024. 82 с.

У даному проекті розробляється онлайн-магазин на стекові MERN (MongoDB, Express, React, Node.js). Веб-застосунок забезпечує користувачам можливість переглядати, шукати та купувати продукти, взаємодіяти з продавцями, залишати відгуки, керувати кошиком та замовленнями, а також використовувати систему автентифікації та авторизації для різних ролей користувачів. Користувачі можуть створювати облікові записи, додавати продукти до списку бажань, а продавці мають можливість додавати та редагувати свої товари.

Веб-застосунок також включає функціонал для управління замовленнями та відгуками, забезпечуючи зручний інтерфейс для адміністраторів і продавців.

Проведено тестування працездатності створеного веб-застосунку, а також виконано порівняння функціональності застосунку з наявними ринковими рішеннями.

Ключові слова: MERN, онлайн-магазин, веб-застосунок, автентифікація, авторизація, відгуки, управління замовленнями.

ABSTRACT

Pages — 82

Drawings — 18

Tables — 1

Source — 9

Kovalenko K. M. Creation of an online store of household goods using React and NodeJS : Bachelor's Qualification Work in the Specialty 121 "Software Engineering" / Scientific Supervisor V. I. Popivshchyi. Zaporizhzhia : ZNU, 2024. 82 p.

In this project, an online store is developed on the MERN stack (MongoDB, Express, React, Node.js). The web application provides users with the ability to browse, search, and purchase products, interact with sellers, leave reviews, manage their cart and orders, and use an authentication and authorization system for different user roles. Users can create accounts, add products to their wishlist, and sellers can add and edit their products.

The web application also includes functionality for managing orders and reviews, providing a convenient interface for administrators and sellers.

The performance of the created web application has been tested, and its functionality has been compared with existing market solutions.

Keywords: MERN, online store, web application, authentication, authorization, reviews, order management.

ЗМІСТ

ВСТУП

Актуальність теми

Сучасні технології продовжують змінювати спосіб ведення бізнесу, і електронна комерція займає центральне місце в цьому процесі. Розробка інтернет-магазину з використанням стеку MERN (MongoDB, Express.js, React, Node.js) є актуальною задачею для багатьох компаній, які прагнуть запропонувати користувачам швидкий, зручний та функціональний спосіб покупки товарів і послуг онлайн.

Електронна комерція продовжує зростати швидкими темпами. Згідно з дослідженнями, обсяг світового ринку електронної комерції щорічно збільшується, і все більше споживачів віддають перевагу покупкам онлайн. Це пов'язано з багатьма факторами, такими як зручність, широкий вибір товарів, можливість порівнювати ціни та отримувати відгуки від інших покупців.

MERN стек включає в себе потужні інструменти, які забезпечують високу продуктивність та гнучкість. MongoDB є нереляційною базою даних, яка забезпечує зберігання даних у форматі JSON, що полегшує роботу з даними та їхнє масштабування. Express.js – це фреймворк для Node.js, який спрощує створення серверних додатків і API. React – бібліотека для побудови користувацьких інтерфейсів, яка дозволяє створювати динамічні та інтерактивні веб-додатки. Node.js, серверне середовище, забезпечує високу продуктивність та дозволяє використовувати JavaScript як на клієнтській, так і на серверній стороні.

Використання MERN стеку для розробки інтернет-магазину дозволяє створювати додатки з високою продуктивністю та масштабованістю. Це забезпечує швидкий відгук на запити користувачів, можливість обробляти великі обсяги даних та підтримувати велику кількість одночасних підключень. Крім того, використання єдиного мови програмування – JavaScript – на всіх етапах

розробки (від фронтенду до бекенду) спрощує процес розробки, тестування та підтримки коду.

Розробка інтернет-магазину на MERN стеку також дає можливість легко інтегрувати різні сервіси та технології, такі як платіжні системи, системи управління контентом, аналітичні інструменти та багато інших. Це робить додаток більш функціональним та зручним для користувачів, що, в свою чергу, сприяє зростанню бізнесу та збільшенню прибутків.

Таким чином, розробка інтернет-магазину з використанням MERN стеку є актуальною і перспективною темою, яка відповідає сучасним вимогам ринку та потребам користувачів.

Мета дослідження

Метою дослідження є розробка ефективного та функціонального інтернет-магазину на основі MERN стеку. Це передбачає оцінку переваг і недоліків MongoDB, Express.js, React і Node.js для створення сучасного веб-додатку. Особлива увага приділяється забезпеченню високої продуктивності, гнучкості та масштабованості системи. Дослідження також включає інтеграцію різних сервісів для покращення функціональності та зручності користувачів.

Завдання дослідження

Завдання цього дослідження полягають у детальному аналізі, проектуванні та реалізації інтернет-магазину на основі MERN стеку. Кожне з завдань спрямоване на досягнення мети дослідження та забезпечення високої якості кінцевого продукту:

1. Проаналізувати технології MERN стеку (MongoDB, Express.js, React, Node.js) та визначити їхню придатність для розробки інтернет-магазину.
2. Розробити архітектуру інтернет-магазину з урахуванням вимог до продуктивності, гнучкості та масштабованості.

3. Реалізувати основні функціональні компоненти інтернет-магазину, такі як каталог товарів, кошик, система користувачів та обробка замовлень.
4. Інтегрувати платіжні системи та інші необхідні сервіси для забезпечення повної функціональності інтернет-магазину.
5. Провести тестування та оптимізацію розробленого додатку для забезпечення його надійної роботи та швидкодії.
6. Оцінка ефективності рекомендаційної системи: Аналіз зворотного зв'язку від користувачів щодо адекватності рекомендованої музики їхньому настрою та вдосконалення механізму рекомендацій.
7. Розробити документацію для користувачів та розробників для полегшення подальшої підтримки та розвитку системи.

Об’єкт дослідження

Об'єктом дослідження є процес розробки інтернет-магазину з використанням MERN стеку, що включає MongoDB, Express.js, React та Node.js. Це дослідження охоплює як технічні аспекти створення веб-додатку, так і інтеграцію різних сервісів для забезпечення повної функціональності. Особлива увага приділяється розробці архітектури, вибору технологій та оптимізації продуктивності системи. Дослідження також зосереджене на забезпеченні зручного користувацького інтерфейсу та ефективного управління даними. Крім того, об'єктом дослідження є вивчення методів інтеграції платіжних систем та збору відгуків користувачів для покращення якості обслуговування.

Предмет дослідження

Предметом дослідження є методи, інструменти та підходи до розробки інтернет-магазину з використанням MERN стеку, що складається з MongoDB, Express.js, React та Node.js. Дослідження включає детальний аналіз кожного з компонентів стеку, їхніх можливостей, переваг та недоліків, а також методів

інтеграції цих технологій для створення ефективного та продуктивного веб-додатку. Особлива увага приділяється проектуванню архітектури інтернет-магазину, реалізації основних функціональних модулів, таких як каталог товарів, кошик, система управління користувачами та обробка замовлень. Інтеграція платіжних систем є важливим аспектом дослідження, що включає в себе вибір найбільш підходящих сервісів та забезпечення їх безпечного та надійного функціонування. Крім того, предметом дослідження є методи управління запасами товарів, збір та аналіз відгуків користувачів для підвищення якості обслуговування, а також процеси тестування та оптимізації додатку для досягнення високої продуктивності та стабільності роботи.

Методи дослідження

У ході дослідження було застосовано як теоретичні, так і експериментальні методи, що дозволило всебічно вивчити процес розробки інтернет-магазину на основі MERN стеку.

Теоретичні методи включають:

1. Вивчення наукових статей, книг, документації та інших джерел, що стосуються MERN стеку, розробки веб-додатків та електронної комерції.
2. Оцінка архітектурних рішень та методів інтеграції різних компонентів MERN стеку для забезпечення їхньої ефективної взаємодії.
3. Створення моделей та схем для візуалізації структури та взаємодії компонентів інтернет-магазину, а також процесів обробки даних та взаємодії з користувачами.
4. Зіставлення різних технологій та інструментів для вибору найефективніших рішень для розробки інтернет-магазину.

Експериментальні методи дослідження охоплюють:

1. Створення прототипів окремих компонентів інтернет-магазину для оцінки їхньої функціональності та продуктивності.
2. Написання коду для реалізації функціональних модулів інтернет-магазину, проведення юніт-тестів та інтеграційних тестів для виявлення та виправлення помилок.
3. Впровадження інтеграційних тестів для перевірки взаємодії між різними компонентами системи та оцінки їхньої сумісності.
4. Використання інструментів моніторингу для оцінки продуктивності додатку, виявлення вузьких місць та їх оптимізація.
5. Проведення тестування з участю користувачів для оцінки зручності використання інтерфейсу інтернет-магазину та отримання зворотного зв’язку для подальшого покращення додатку.

Практичне значення одержаних результатів

Практичне значення одержаних результатів полягає у створенні ефективних рішень для розробки сучасних веб-додатків, зокрема інтернет-магазинів, на основі MERN стеку. Застосовані методи та отримані висновки можуть бути корисними для розробників, які прагнуть створити високоефективні та масштабовані веб-додатки. Розроблена архітектура інтернет-магазину може служити шаблоном для подібних проектів, що зменшить час на проектування та збільшить продуктивність розробки. Описані методи інтеграції платіжних систем дозволяють швидко та безпечно впроваджувати оплату в інтернет-магазинах. Крім того, отримані результати щодо оптимізації продуктивності та тестування забезпечують стабільну роботу додатку, що підвищує якість обслуговування користувачів.

Глосарій

MERN стек (MERN stack) - Набір технологій для розробки веб-додатків, що включає MongoDB, Express.js, React та Node.js.

MongoDB - Нереляційна база даних, що використовує документи у форматі JSON для зберігання даних.

Express.js - Фреймворк для Node.js, що спрощує розробку серверних додатків та API.

React - Бібліотека для створення користувацьких інтерфейсів, яка дозволяє розробляти динамічні та інтерактивні веб-додатки.

Node.js - Серверне середовище, яке дозволяє запускати JavaScript на сервері та забезпечує високу продуктивність додатків.

API (Application Programming Interface) - Інтерфейс для взаємодії різних програмних компонентів.

JWT (JSON Web Token) - Стандарт відкритого формату для безпечного обміну даними між сторонами у вигляді JSON-об’єктів.

Redux - Бібліотека для управління станом додатку в JavaScript, часто використовується разом з React.

Компонент (Component) - Основний будівельний блок у React, який представляє собою частину користувацького інтерфейсу.

Маршрутизація (Routing) - Процес визначення шляхів (маршрутів) для переходу між різними сторінками або компонентами веб-додатку.

Проміжне програмне забезпечення (Middleware) - Програмне забезпечення, що виконує функції посередника між різними компонентами додатку, часто використовується для обробки запитів у Express.js.

CRUD операції (Create, Read, Update, Delete) - Основні операції для роботи з базами даних або іншими сховищами даних, що дозволяють створювати, читати, оновлювати та видаляти записи.

DOM (Document Object Model) - Об’єктна модель документа, яка представляє структуру HTML або XML документів у вигляді дерева, що дозволяє скриптам динамічно змінювати вміст, структуру та стиль документа.

JSX (JavaScript XML) - Розширення синтаксису JavaScript, яке дозволяє писати HTML-подібний код у файлах JavaScript, часто використовується у React для опису вигляду компонентів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз програмних продуктів аналогів

Розробка інтернет-магазину на основі MERN стеку потребує аналізу існуючих програмних продуктів-аналогів, які можуть слугувати орієнтиром для визначення функціональних можливостей та вимог до розроблюваного додатку. Нижче наведено огляд деяких популярних платформ електронної комерції та їх характеристик.

Shopify— Shopify є однією з найпопулярніших платформ для створення інтернет-магазинів. Вона пропонує готові рішення для швидкого запуску онлайн-продажів, включаючи шаблони для різних типів бізнесу, інтеграцію з платіжними системами та аналітику продажів.

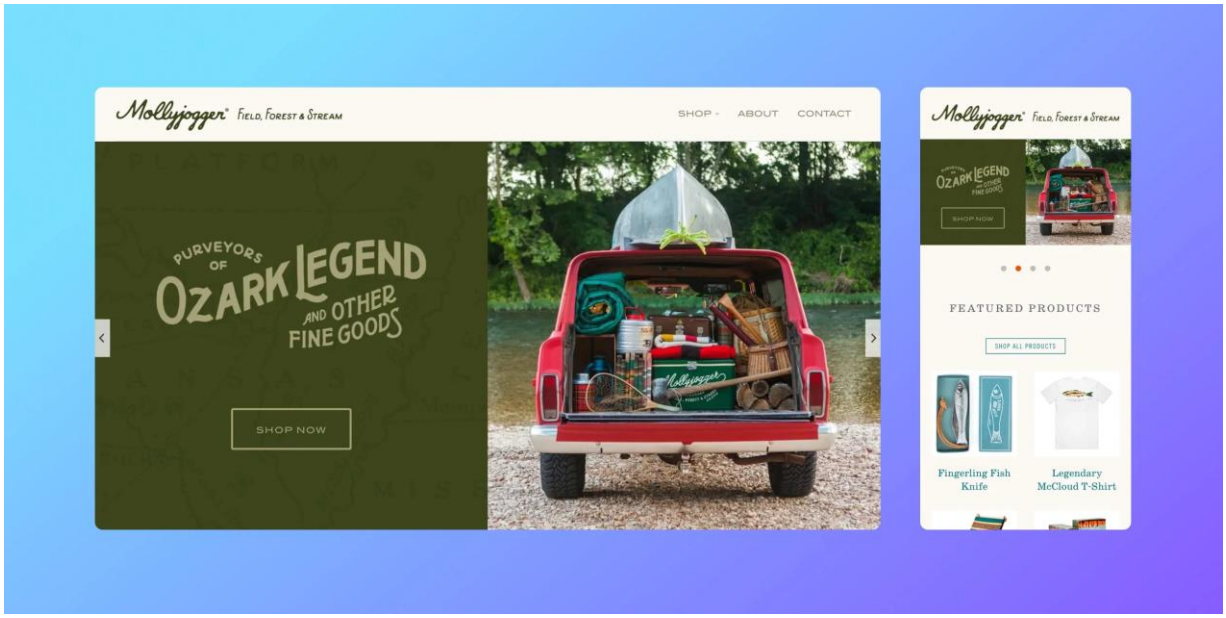

Рисунок 1 — *Інтерфейс Shopify*

Переваги:

- Простота використання: Легкий старт і простий у налаштуванні інтерфейс.
- Надійність: Висока надійність та стабільність роботи платформи.
- Підтримка додатків: Велика кількість доступних додатків та інтеграцій.

Недоліки:

- Обмежена можливість кастомізації: Обмежені можливості для глибокого налаштування.
- Вартість: Висока вартість підписки, особливо для великих магазинів.

WooCommerce — WooCommerce є плагіном для WordPress, що дозволяє легко перетворити сайт на платформі WordPress у повноцінний інтернет-магазин.

Рисунок 2 — *Інтерфейс* WooCommerce

Переваги:

- Гнучкість: Велика кількість тем та розширень для налаштування магазину.
- Інтеграція: Широкі можливості для інтеграції з різними сервісами.
- Відкритий код: Можливість повної кастомізації та розширення функціоналу.

Недоліки:

- Складність налаштування: Може бути складним для новачків.
- Вимоги до хостингу: Потребує надійного хостингу для стабільної роботи.

Magento — Magento є потужною платформою для електронної комерції, орієнтованою на великий бізнес та підприємства. Вона пропонує широкий спектр функцій для управління товарами, замовленнями, користувачами та аналітикою.

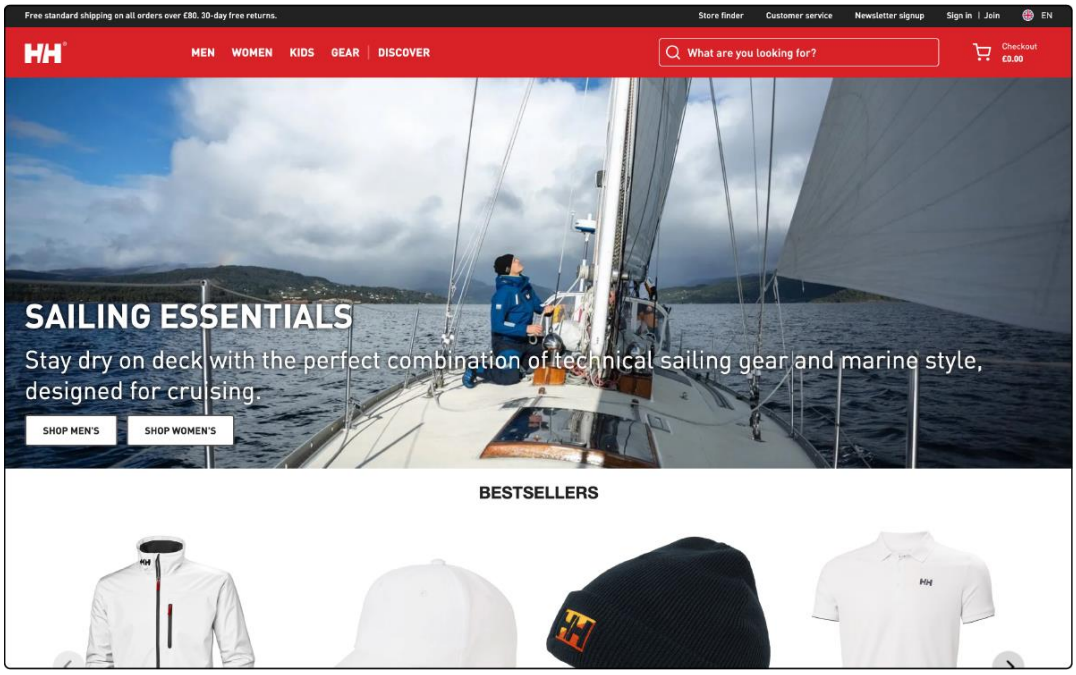

Рисунок 3 — *Magento*

Переваги:

- Масштабованість: Підходить для великих бізнесів та підприємств.
- Функціональність: Широкий спектр функцій для управління магазином.
- Гнучкість: Можливість повної кастомізації під специфічні вимоги бізнесу.

Недоліки:

- Складність: Складне налаштування та управління.
- Вартість: Висока вартість обслуговування та впровадження.

BigCommerce — BigCommerce є ще однією популярною платформою для створення інтернет-магазинів, яка пропонує широкий набір інструментів для управління товарами, замовленнями та маркетингом.

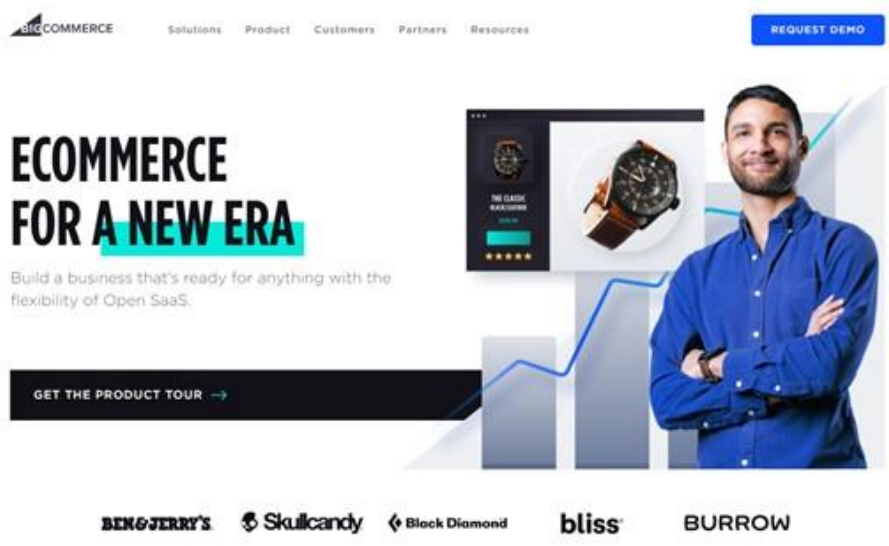

Рисунок 4 — *BigCommerce*

Переваги:

- Продуктивність: Висока продуктивність та швидкість роботи.
- Простота використання: Інтуїтивно зрозумілий інтерфейс.
- Інтеграції: Підтримка великої кількості інтеграцій з зовнішніми сервісами.

Недоліки:

- Кастомізація: Обмежені можливості для глибокого налаштування.
- Вартість: Висока вартість підписки.

Таблиця 1 *Порівняння характеристик Shopify, WooCommerce, Magento та BigCommerce*

Властивість	Shopify	WooCommerce	Magento	BigCommerce
Простота використання	Висока	Середня	Низька	Висока

Гнучкість кастомізації	Обмежена	Висока	Висока	Обмежена
Вартість	Висока	Залежить від хостингу	Висока	Висока
Масштабованість	Висока	Середня	Висока	Висока
Інтеграція з платіжними системами	Широка підтримка	Широка підтримка	Широка підтримка	Широка підтримка
Підтримка додатків	Велика кількість додатків	Велика кількість плагінів	Велика кількість розширень	Велика кількість додатків
Вимоги до технічних знань	Низькі	Середні	Високі	Низькі
Продуктивність	Висока	Залежить від хостингу	Висока	Висока
Можливість масштабування	Висока	Обмежена хостингом	Висока	Висока
Підтримка клієнтів	Висока	Спільнота та комерційні підтримки	Спільнота та комерційні підтримки	Висока

На основі порівняльної таблиці можна зробити висновок, що кожна з платформ має свої переваги та недоліки. Shopify виділяється своєю простотою використання та надійністю, але має обмежені можливості кастомізації та високу вартість. WooCommerce забезпечує високу гнучкість і можливість кастомізації, але потребує більше технічних знань і надійного хостингу. Magento підходить для великих бізнесів завдяки своїй масштабованості та функціональності, однак потребує значних фінансових та технічних ресурсів. BigCommerce пропонує хорошу продуктивність і гнучкість, але має свої особливості, які слід враховувати при виборі.

1.2 Постановка завдання

Постановка завдання включає визначення ключових етапів та цілей, які необхідно досягти в процесі розробки інтернет-магазину на основі MERN стеку. Кожне завдання спрямоване на забезпечення високої якості кінцевого продукту, його продуктивності та зручності у використанні.Основний функціонал застосунку:

1. Аналіз вимог до системи:
    - Визначити функціональні та нефункціональні вимоги до інтернет-магазину..
    - Провести дослідження цільової аудиторії та її потреб.
    - Описати основні бізнес-процеси, які повинні бути реалізовані в додатку.
2. Проектування архітектури додатку:
    - Розробити архітектуру інтернет-магазину з урахуванням вимог до продуктивності, гнучкості та масштабованості.
    - Визначити структуру бази даних та зв'язки між її елементами.
    - Спроектувати інтерфейс користувача, враховуючи зручність та інтуїтивність навігації.
3. Реалізація бекенду::
    - Розробити серверну частину додатку з використанням Node.js та Express.js.
    - Налаштувати базу даних MongoDB для зберігання інформації про товари, користувачів та замовлення.
    - Реалізувати API для взаємодії між клієнтською та серверною частинами додатку.
4. Реалізація фронтенду::
    - Розробити клієнтську частину додатку з використанням React.

- Забезпечити динамічне оновлення інтерфейсу без перезавантаження сторінки.
- Реалізувати основні функціональні компоненти, такі як каталог товарів, кошик, система користувачів та обробка замовлень.

5. Інтеграція платіжних систем:
    - Вибрати платіжні системи, які будуть підтримуватися інтернет-магазином.
    - Забезпечити безпечну інтеграцію з цими платіжними системами.
    - Реалізувати можливість здійснення платежів та повернення коштів.
6. Оптимізація додатку:
    - Оптимізувати продуктивність додатку для забезпечення швидкого відгуку на запити користувачів.
    - Виконати стрес-тестування для оцінки стійкості системи під навантаженням.

Унікальний функціонал:

1. Розширені функції аутентифікації та авторизації користувачів:
    - Реалізація системи реєстрації та входу користувачів з валідацією даних.
    - Підтримка ролей користувачів (адміністратор, покупець) з обмеженням доступу до адміністративних функцій для звичайних користувачів.
2. Розширені можливості для користувачів:
    - Користувачі можуть реєструватися та входити в систему, створювати та зберігати списки бажань.

- Функціонал відгуків та рейтингів дозволяє користувачам залишати відгуки про придбані товари, що допомагає іншим покупцям робити обґрунтований вибір.

Унікальні функціональні можливості проекту, такі як інтерактивна панель адміністратора, система обробки замовлень з реальним часом оновлення та розширені можливості для користувачів, роблять його універсальним і зручним як для користувачів, так і для адміністраторів. Ці особливості забезпечують високу продуктивність, масштабованість та зручність використання системи, що є важливими чинниками для успішного функціонування інтернет-магазину.

2 ДОСЛІДЖЕННЯ ЗАСОБІВ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ВЕБ-ЗАСТОСУНКУ

2.1 Огляд мов програмування

Вибір мови програмування для веб-застосунків є критично важливим рішенням, яке залежить від багатьох факторів, включаючи вимоги проекту, досвід розробників, продуктивність, масштабованість та інші. Ось огляд деяких з найпопулярніших мов програмування, що використовуються для розробки веб-застосунків:

Python — Python є популярним вибором для бекенд-розробки завдяки своїй простоті та читабельності коду.

Переваги:

- Висока читабельність і простота вивчення.
- Велика кількість бібліотек і фреймворків (Django, Flask).
- Підтримка різних парадигм програмування (об'єктно-орієнтоване, функціональне).

Недоліки:

- Відносно низька продуктивність у порівнянні з мовами, що компілюються.
- Менша підтримка асинхронного програмування в порівнянні з JavaScript..

PHP — PHP залишається популярним вибором для розробки серверної частини веб-застосунків, особливо завдяки своїй інтеграції з веб-серверами.

Переваги:

- Простота інтеграції з веб-серверами (наприклад, Apache).
- Широка підтримка баз даних.

- Велика спільнота і багато готових рішень (CMS, фреймворки).

Недоліки:

- Історично, погана репутація через хаотичний код.
- Менша сучасність в порівнянні з новішими мовами.

Ruby — Ruby використовується в основному з фреймворком Ruby on Rails, який надає багато інструментів для швидкої розробки.

Переваги:

- Висока продуктивність розробки завдяки конвенціям фреймворка.
- Читабельний і лаконічний код.
- Велика кількість вбудованих інструментів.

Недоліки:

- Менша продуктивність у порівнянні з деякими іншими мовами.
- Складність масштабування великих проектів.

Java — Java широко використовується для створення великих і масштабованих веб-застосунків завдяки своїй стабільності і продуктивності.

Переваги:

- Висока продуктивність і стабільність.
- Підтримка багатопоточності.
- Велика кількість фреймворків (Spring, Hibernate).

Недоліки:

- Більш складний і об'ємний код у порівнянні з іншими мовами.
- Високий поріг входу для новачків.

JavaScript - JavaScript є найпопулярнішою мовою програмування для фронтенд-розробки. Вона дозволяє створювати інтерактивні елементи на веб-сторінках.

Переваги:

- Підтримується всіма сучасними веб-браузерами.

- Велика кількість бібліотек і фреймворків (React, Angular, Vue.js).
- Швидкість виконання на клієнтській стороні.
- Підтримка асинхронного програмування (Promises, async/await).

Недоліки:

- Безпека коду, оскільки він виконується на клієнтській стороні.
- Необхідність налаштування середовища для серверного використання (Node.js).

Кожна мова програмування має свої переваги та недоліки, і вибір конкретної мови залежить від вимог проекту та навичок команди. Важливо враховувати не тільки поточні потреби, але й можливість масштабування та підтримки проекту в майбутньому.

2.2 Аналіз сучасних фреймворків для створення веб-застосунків

Фреймворки є важливим інструментом для розробників, оскільки вони надають структуру для організації коду, а також пропонують інструменти та бібліотеки, що прискорюють процес розробки. Нижче наведено огляд найсучасніших і найпопулярніших фреймворків для створення веб-застосунків, розділених на фронтенд та бекенд фреймворки.

React — це бібліотека JavaScript для створення користувацьких інтерфейсів, розроблена компанією Facebook. Вона використовує компонентний підхід, що дозволяє створювати повторно використовувані компоненти, і забезпечує високу продуктивність завдяки віртуальному DOM. React має велику спільноту та широкий вибір бібліотек і інструментів, а також підтримку React Native для створення мобільних застосунків. Проте, для повноцінної роботи React потребує додаткових бібліотек, таких як Redux для управління станом, і вимагає постійного оновлення знань через стрімкий розвиток.

Angular — це повноцінний фреймворк для розробки веб-застосунків, розроблений Google. Він надає повний набір інструментів для розробки, включаючи двостороннє зв'язування даних, роутінг і вбудовані сервіси. Angular забезпечує високу продуктивність і масштабованість, а також чітку структуру проекту та дотримання принципів MVC (Model-View-Controller). Однак, він має високий поріг входу для новачків і великий обсяг коду порівняно з іншими фреймворками.

Vue.js — це прогресивний фреймворк JavaScript для побудови користувацьких інтерфейсів. Він відрізняється легкістю у вивченні та використанні, можливістю поступового інтегрування в проекти, високою продуктивністю і гнучкістю. Vue.js також має вбудовані інструменти для управління станом і роутінгу (Vuex, Vue Router). Проте, у порівнянні з React та Angular, Vue.js має меншу спільноту і обмежені ресурси для великих корпоративних проектів.

Svelte — це сучасний фронтенд-фреймворк, який відрізняється від традиційних фреймворків тим, що більшість роботи виконується на етапі компіляції, а не в браузері. Це забезпечує високу продуктивність і мінімальний розмір кінцевого коду.

Next.js — це фреймворк для React, який надає інструменти для серверного рендерингу та створення статичних сайтів. Він підтримує інкрементальну генерацію статичних сторінок, автоматичне розділення коду і багато інших сучасних можливостей.

Astro — це новий фреймворк для створення статичних сайтів, який дозволяє використовувати будь-який фронтенд-фреймворк (React, Vue, Svelte тощо) для створення компонентів. Astro компілює сайт у максимально легкий та швидкий статичний контент.

2.3 Огляд архітектурних патернів

Архітектурні патерни є важливими концепціями в розробці програмного забезпечення, які допомагають розробникам організувати код і забезпечити його підтримуваність, масштабованість і гнучкість. Нижче наведено огляд деяких з найбільш поширених архітектурних патернів, що використовуються при створенні веб-застосунків.

Монолітна архітектура - Монолітна архітектура передбачає побудову всього застосунку як єдиного цілісного коду, де всі компоненти взаємодіють один з одним всередині одного проекту. Це може бути зручно для невеликих проектів, де простота управління важлива. Однак, зростання застосунку може призвести до проблем з масштабованістю та підтримкою, оскільки зміни в одному компоненті можуть впливати на інші компоненти.

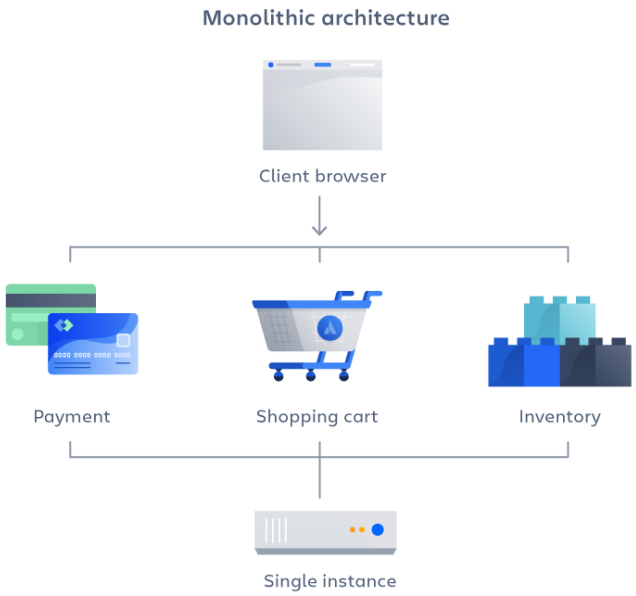

Рисунок 5 — *Схема монолітної архітектури*

Мікросервісна архітектура - Мікросервісна архітектура розділяє застосунок на невеликі, незалежні сервіси, кожен з яких виконує одну конкретну функцію і взаємодіє з іншими сервісами через чітко визначені інтерфейси (API). Це забезпечує легшу масштабованість і підвищує стійкість системи, оскільки кожен сервіс може розгортатися і оновлюватися незалежно від інших. Однак, мікросервіси можуть збільшити складність управління через необхідність координації між численними сервісами.

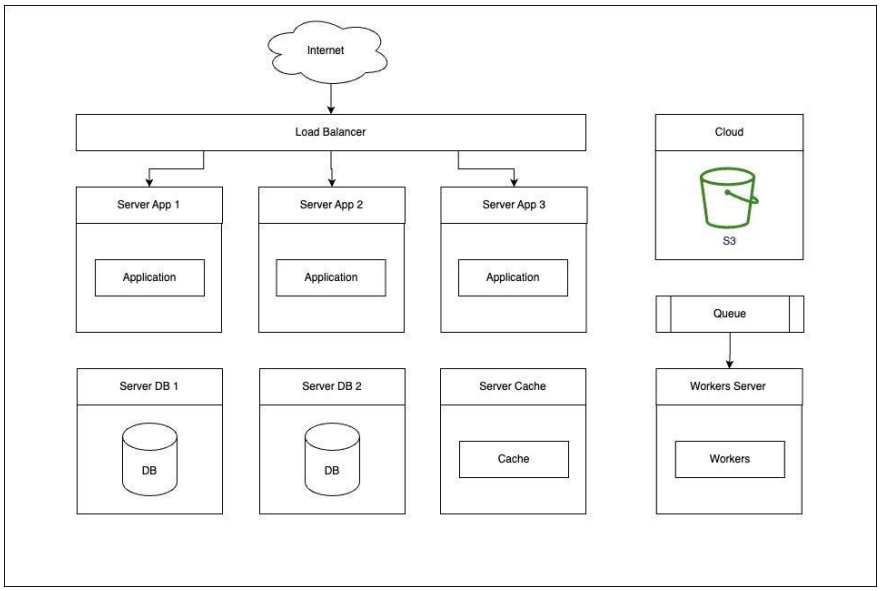

Рисунок 6 — *Схема Мікросервісної архітектури*

Архітектура MVC - MVC є популярним архітектурним патерном, який розділяє застосунок на три основні компоненти: модель (Model), представлення (View) і контролер (Controller). Модель відповідає за дані та логіку бізнесу, представлення відповідає за відображення даних користувачам, а контролер обробляє вхідні запити і координує взаємодію між моделлю і представленням. Такий підхід забезпечує чіткий поділ відповідальностей і покращує підтримуваність коду.

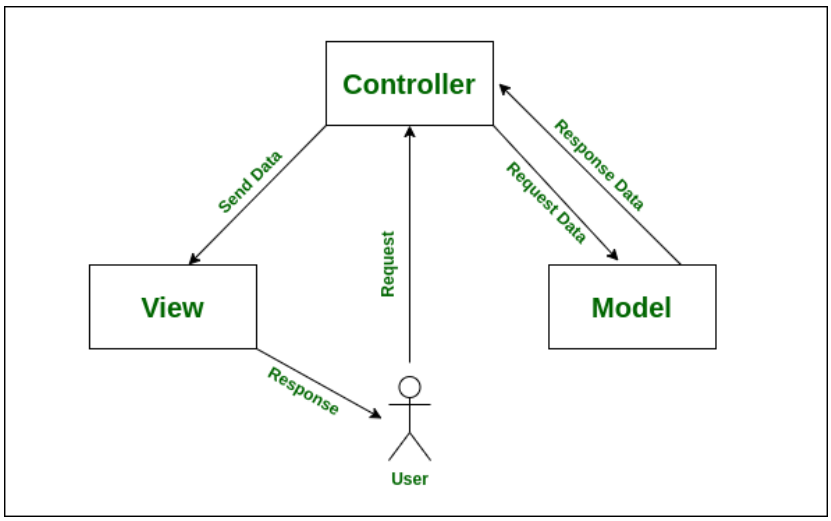

Рисунок 7 — *Схема архітектури MVC*

Архітектура MVVM (Model-View-ViewModel) - MVVM є варіацією патерну MVC, який особливо популярний у розробці клієнтських застосунків. В цій архітектурі ViewModel слугує посередником між моделлю та представленням, забезпечуючи двостороннє зв'язування даних і автоматичне оновлення інтерфейсу користувача. Це сприяє більшій гнучкості та покращує тестованість коду.

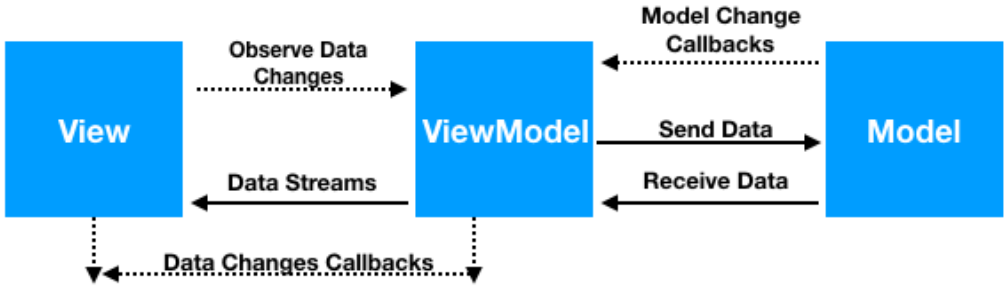

Рисунок 8 — *Схема архітектури MVVM*

Архітектура SPA (Single Page Application) — SPA (односторінковий застосунок) — це архітектурний підхід, при якому вся функціональність веб-застосунку реалізується на одній веб-сторінці. Всі необхідні ресурси завантажуються один раз, а динамічний контент оновлюється за допомогою JavaScript без перезавантаження сторінки. Це забезпечує швидку та плавну роботу користувацького інтерфейсу, але може збільшити складність розробки та оптимізації початкового завантаження.

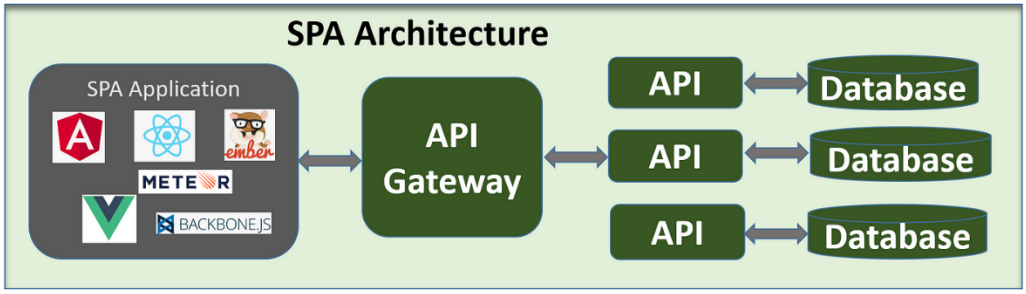

Рисунок 9 — *Схема архітектури SPA*

Архітектура PWA (Progressive Web App) - PWA поєднує в собі переваги веб- та мобільних застосунків, забезпечуючи швидку роботу, офлайн-режим та можливість встановлення на пристрій як нативного застосунку. PWA використовує сучасні веб-технології, такі як Service Workers і маніфести застосунків, для забезпечення високої продуктивності та покращеного користувацького досвіду.

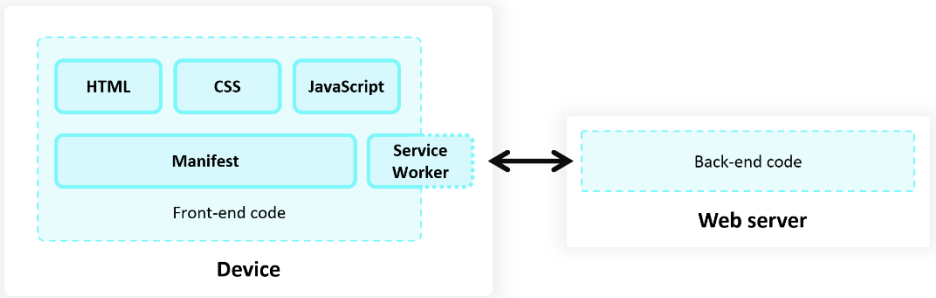

Рисунок 10 — *Схема архітектури PWA*

У моєму проекті фронтенд використовує компонентний підхід з React, Redux для управління станом і React Router для маршрутизації, що відповідає патерну MVVM (Model-View-ViewModel). На бекенді використовується архітектура MVC (Model-View-Controller) з Express.js, де маршрути обробляють запити, контролери керують логікою, а моделі взаємодіють з базою даних. Використання REST API для обміну даними між фронтендом і бекендом додає структурованість і гнучкість.

2.4 Огляд управлінь залежностями

Управління залежностями є важливим аспектом розробки програмного забезпечення, що дозволяє ефективно використовувати зовнішні бібліотеки та фреймворки, підтримувати їх актуальність і забезпечувати сумісність з вашим проектом. В цьому розділі розглянемо основні інструменти та підходи до управління залежностями у фронтенд та бекенд розробці.

npm (Node Package Manager) — це стандартний менеджер пакетів для JavaScript, який використовується для управління залежностями у проектах на базі Node.js. npm дозволяє легко встановлювати, оновлювати та видаляти пакети, а також керувати їх версіями.

Основні команди npm включають:

- npm install — встановлює залежності, вказані у файлі package.json.
- npm update — оновлює встановлені пакети до останніх версій.
- npm uninstall — видаляє пакет із проекту.

Yarn — це альтернативний менеджер пакетів для JavaScript, який забезпечує швидшу і безпечнішу установку залежностей. Yarn використовує файл yarn.lock для фіксації версій встановлених пакетів, що дозволяє уникнути проблем з сумісністю між розробниками.

Основні команди Yarn включають:

- yarn install — встановлює всі залежності, вказані у файлі package.json та yarn.lock.
- yarn add <package> — додає новий пакет до проекту і автоматично оновлює yarn.lock.
- yarn remove <package> — видаляє пакет з проекту і оновлює yarn.lock.

pip - це стандартний менеджер пакетів для Python, який використовується для встановлення та управління додатковими бібліотеками та модулями. pip дозволяє легко встановлювати, оновлювати та видаляти пакети, а також керувати їх версіями.

Основні команди pip включають:

- pip install <package> — встановлює пакет.
- pip freeze > requirements.txt — генерує файл requirements.txt з поточними версіями всіх встановлених пакетів.
- pip install -r requirements.txt — встановлює всі пакети, вказані у файлі requirements.txt.

Управління залежностями є критично важливим аспектом розробки програмного забезпечення, який дозволяє ефективно використовувати зовнішні бібліотеки і фреймворки, підтримувати їх актуальність і забезпечувати сумісність з вашим проектом. Використання інструментів, таких як npm, Yarn, Webpack для JavaScript проектів, а також pip і Virtualenv для Python проектів, дозволяє автоматизувати процес управління залежностями і уникнути багатьох потенційних проблем з сумісністю і оновленнями.

2.5 Огляд управлінь залежностями

Системи керування базами даних (СКБД) є ключовими компонентами сучасних інформаційних систем, що забезпечують зберігання, управління та

обробку даних. Вибір СКБД залежить від вимог проекту, обсягу даних, продуктивності та інших факторів. Ось огляд деяких з найпопулярніших СКБД, що використовуються сьогодні.

MySQL — це одна з найпопулярніших реляційних СКБД, що використовується для створення веб-застосунків. Вона забезпечує високу продуктивність, надійність та зручність у використанні. MySQL підтримує стандарт SQL для роботи з базами даних і широко використовується в поєднанні з іншими технологіями, такими як PHP і Node.js.

PostgreSQL — це потужна реляційна СКБД з відкритим вихідним кодом, яка підтримує широкий спектр сучасних можливостей, таких як складні запити, транзакції, і тригери. PostgreSQL відома своєю надійністю та масштабованістю, що робить її популярною для великих і складних проектів.

SQLite — це легка реляційна СКБД, яка зберігає всю базу даних у одному файлі. Вона ідеально підходить для невеликих проектів, мобільних додатків і вбудованих систем. SQLite проста у налаштуванні та використанні, але має обмежену продуктивність у порівнянні з іншими реляційними СКБД.

MongoDB — це одна з найпопулярніших NoSQL баз даних, що використовує документо-орієнтований підхід до зберігання даних. Вона забезпечує високу гнучкість і масштабованість, підтримуючи динамічні схеми та реплікацію даних. MongoDB часто використовується в поєднанні з Node.js для створення веб-застосунків.

Cassandra — це розподілена NoSQL база даних, розроблена для обробки великих обсягів даних на багатьох серверах. Вона забезпечує високу доступність і відмовостійкість, використовуючи реплікацію даних між вузлами. Cassandra підходить для застосунків, що потребують високої швидкості запису і читання даних.

Redis — це NoSQL база даних, яка використовується як кеш або сховище даних у пам'яті. Вона підтримує структури даних, такі як рядки, хеші, списки та

множини. Redis забезпечує дуже високу швидкість доступу до даних і часто використовується для оптимізації продуктивності веб-застосунків.

Google Cloud Firestore — це гнучка, масштабована хмарна база даних NoSQL, розроблена Google. Вона забезпечує реальний час синхронізації даних і легку інтеграцію з іншими сервісами Google Cloud. Firestore підходить для мобільних і веб-застосунків, що потребують швидкого доступу до даних і високої доступності.

У моєму проекті використовується MongoDB як база даних. MongoDB є однією з найпопулярніших NoSQL баз даних, яка забезпечує високу гнучкість і масштабованість. Вона використовує документо-орієнтований підхід до зберігання даних, що дозволяє зберігати дані у форматі JSON-подібних документів. MongoDB часто використовується в поєднанні з Node.js для створення веб-застосунків, що відповідає структуру вашого MERN-стека (MongoDB, Express.js, React, Node.js).

2.6 Огляд API

API (Application Programming Interface) є ключовим елементом у розробці веб-застосунків, що дозволяє різним програмним компонентам взаємодіяти між собою. API визначає набір правил і протоколів для побудови і інтеграції програмного забезпечення. У цьому розділі розглянемо основні типи API, їх характеристики та приклади використання**.**

REST (Representational State Transfer) — це архітектурний стиль для створення веб-сервісів, що використовують HTTP протокол. REST API базуються на принципах простоти, масштабованості і продуктивності. Вони використовують стандартні HTTP методи, такі як GET, POST, PUT, DELETE, для взаємодії з ресурсами.

GraphQL — це запитова мова для API, розроблена Facebook, яка дозволяє клієнтам точно визначати, які дані їм потрібні. На відміну від REST, де кожен кінцевий пункт повертає фіксований набір даних, GraphQL дозволяє клієнтам запитувати тільки ті поля, які їм потрібні, що підвищує ефективність використання мережі.

SOAP (Simple Object Access Protocol) — це протокол для обміну структурованими повідомленнями між додатками через різні мережі. SOAP API використовують XML для форматування повідомлень і можуть працювати через різні транспортні протоколи, такі як HTTP, SMTP та інші. SOAP забезпечує високий рівень безпеки і надійності, але є більш складним у використанні порівняно з REST.

Для мого веб-додатку я вибрав REST API через його простоту, гнучкість і сумісність з різними клієнтськими додатками. Використання REST API дозволяє легко інтегрувати бекенд з фронтендом, забезпечуючи ефективну взаємодію між компонентами. Також у проекті використовується JWT (JSON Web Token) для автентифікації та авторизації користувачів, що забезпечує високий рівень безпеки доступу до ресурсів. Для зберігання даних обрано MongoDB, яка забезпечує гнучкість і масштабованість, необхідні для сучасних веб-застосунків.

2.7 Висновок

Для розробки цього веб-додатку було обрано сучасні та ефективні засоби, що забезпечують високу продуктивність, масштабованість і безпеку.

Фронтенд - React було обрано для створення користувацького інтерфейсу завдяки його компонентному підходу, який сприяє розробці повторно використовуваних і добре структурованих компонентів. Redux використовується для управління станом додатку, що дозволяє ефективно обробляти глобальний стан і спрощує управління даними. React Router забезпечує маршрутизацію в

додатку, дозволяючи легко керувати навігацією між різними сторінками та компонентами.

Бекенд - Для серверної частини використовується Node.js з Express.js, що дозволяє створювати швидкі та легкі RESTful API. Це рішення забезпечує високу продуктивність і гнучкість у розробці бекенду. JWT (JSON Web Token) було обрано для автентифікації та авторизації користувачів, що гарантує безпеку доступу до ресурсів додатку.

База даних - Для зберігання даних використовується MongoDB, яка є потужною NoSQL базою даних. Вона забезпечує гнучкість у зберіганні даних завдяки документно-орієнтованій структурі, що дозволяє легко масштабувати додаток при зростанні кількості користувачів і обсягу даних.

API - Для інтеграції з зовнішніми сервісами обрано REST API, що забезпечує простоту та гнучкість у взаємодії між різними компонентами додатку.

Менеджер пакетів - Для управління залежностями використовується npm (Node Package Manager), який дозволяє легко встановлювати, оновлювати та видаляти пакети, а також керувати їх версіями. Використання npm спрощує процес розробки та забезпечує стабільність проекту завдяки фіксації версій пакетів у файлі package.json.

3. ПРОЕКТУВАННЯ ТА РОЗРОБКА ВЕБ-ДОДАТКУ

3.1 Опис предметної області

Проект спрямований на розробку веб-додатку для електронної комерції, зокрема інтернет-магазину, який дозволяє користувачам переглядати, шукати, додавати до кошика та купувати різні товари онлайн. Основна мета полягає у створенні зручного, інтуїтивного і швидкого інтерфейсу для користувачів, що сприяє позитивному досвіду використання і підвищенню лояльності клієнтів.

Користувачі мають можливість переглядати каталог товарів, структурований за категоріями, брендами, цінами та іншими атрибутами. Пошукова функція допомагає швидко знаходити товари за ключовими словами, а фільтри та сортування дозволяють звузити результати пошуку. Кожен товар має детальну сторінку з описом, фотографіями, ціною, рейтингами та відгуками від інших покупців.

Додаток включає функціонал кошика, де користувачі можуть додавати обрані товари, переглядати їх список, змінювати кількість або видаляти товари. Після завершення вибору товарів користувачі переходять до оформлення замовлення, вводячи інформацію для доставки, вибираючи спосіб оплати та підтверджуючи замовлення.

Проект також включає адміністративну панель для управління асортиментом товарів, замовленнями та користувачами. Адміністратори можуть додавати нові товари, редагувати інформацію про існуючі, видаляти непотрібні товари, а також оновлювати запаси та ціни. Управління замовленнями включає перегляд усіх замовлень, оновлення їх статусів, а також обробку повернень і відшкодувань. Управління користувачами дозволяє адміністраторам переглядати список

зареєстрованих користувачів, редагувати їх інформацію, змінювати права доступу та блокувати облікові записи.

Для забезпечення безпеки даних користувачів у проекті використовується автентифікація та авторизація за допомогою JWT (JSON Web Token), що гарантує безпечний доступ до облікових записів та персональних даних. Інтеграція з зовнішніми сервісами через REST API забезпечує простоту і гнучкість у взаємодії між різними компонентами додатку, включаючи платіжні системи та логістичні сервіси.

3.2 Архітектура системи

Архітектура системи веб-додатку для електронної комерції побудована з використанням сучасних підходів і технологій, що забезпечують гнучкість, масштабованість та ефективність. Система складається з фронтенд і бекенд частин, які взаємодіють між собою через API. Далі наведено детальний опис архітектури системи та архітектурних підходів, які використовуються, зокрема MVC.

Фронтенд частина додатку побудована на основі React, що забезпечує створення компонентів, які можна повторно використовувати, і дозволяє ефективно керувати станом додатку за допомогою Redux. Використовується React Router для маршрутизації, що дозволяє легко керувати навігацією між різними сторінками додатку. Всі компоненти фронтенду об'єднані у єдину структуру, яка забезпечує зручний і швидкий користувацький інтерфейс.

Фронтенд взаємодіє з бекендом через REST API, що дозволяє надсилати запити для отримання даних про товари, користувачів та замовлення, а також для передачі даних про нові замовлення або зміни в існуючих даних. Використовується архітектурний підхід MVVM (Model-View-ViewModel), який сприяє чіткому розділенню логіки представлення та бізнес-логіки.

Бекенд частина системи побудована на Node.js з використанням Express.js для створення RESTful API. Використовується архітектурний підхід MVC (Model-View-Controller), який забезпечує чітке розділення логіки програми на три основні компоненти:

- Model (Модель): Цей компонент відповідає за взаємодію з базою даних MongoDB. Моделі визначають структуру даних, використовувану в додатку. Вони містять логіку для збереження, оновлення, видалення і отримання даних з бази. У вашому проекті, моделі знаходяться в каталозі models і включають такі файли, як userModel.js, productModel.js і orderModel.js. Кожна модель описує схему даних для відповідного об'єкта, використовуючи Mongoose, бібліотеку для роботи з MongoDB у Node.js.
- View (Подання): У контексті бекенд архітектури, поданням є форматовані відповіді API, які відправляються клієнту. Ці відповіді зазвичай знаходяться у форматі JSON і містять дані, запитані клієнтом. Подання у вашій системі не включає традиційні HTML або шаблони, оскільки вони обробляються на фронтенді. Натомість, View в вашому додатку представлено об'єктами JSON, що повертаються клієнту після обробки запиту.
- Controller (Контролер): Контролери обробляють вхідні HTTP-запити, взаємодіють з моделями і формують відповіді, які відправляються клієнту. Контролери містять бізнес-логіку і забезпечують зв'язок між моделями і представленням. У вашому проекті контролери розташовані в каталозі controllers і включають такі файли, як userController.js, productController.js і orderController.js. Наприклад, productController.js містить функції для обробки запитів, пов'язаних з продуктами, такі як отримання списку продуктів, додавання нового продукту, оновлення інформації про продукт і видалення продукту.

Для забезпечення безпеки використовується JWT (JSON Web Token), що дозволяє надійно керувати сесіями користувачів і захищати доступ до ресурсів додатку. JWT використовується для захисту маршрутів, які потребують автентифікації, забезпечуючи, що тільки авторизовані користувачі мають доступ до певних ресурсів або дій.

База даних - Як основна база даних використовується MongoDB, яка забезпечує гнучкість у зберіганні даних завдяки документно-орієнтованій структурі. MongoDB дозволяє зберігати дані у форматі JSON-подібних документів, що ідеально підходить для веб-додатків, де дані можуть мати складну структуру і часто змінюватися.

Інтеграції - система також може інтегруватися з зовнішніми сервісами через REST API для розширення функціональності, наприклад, для обробки платежів або відстеження доставки замовлень. Такі інтеграції забезпечують більшу гнучкість і розширюють можливості додатку без необхідності розробки власних рішень для кожної додаткової функції.

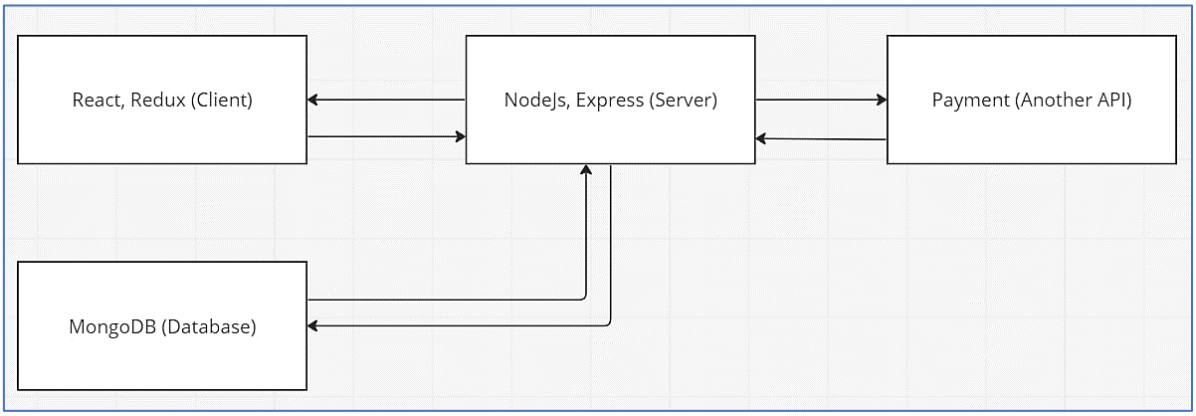

Рисунок 11 — *Архітектура системи*

Архітектура системи веб-додатку для електронної комерції забезпечує чітке розділення фронтенд і бекенд частин, що підвищує гнучкість і

масштабованість проекту. Використання сучасних технологій і підходів, таких як React, Redux, Node.js, Express.js, MongoDB і JWT, дозволяє створити ефективний, безпечний і зручний у використанні додаток, який може легко інтегруватися з зовнішніми сервісами і масштабуватися у відповідь на зростання кількості користувачів і обсягів даних. Архітектурні підходи MVVM на фронтенді та MVC на бекенді сприяють чіткому розділенню відповідальностей, полегшуючи підтримку та розвиток системи.

3.3 Фукціональні вимоги системи

Функціональні вимоги системи визначають, які можливості і функції повинні бути реалізовані у веб-додатку для електронної комерції. Вимоги базуються на основних потребах користувачів та адміністративного персоналу, забезпечуючи повний спектр функціональності для ефективного використання і управління системою.

1. Реєстрація та аутентифікація користувачів
    - Реєстрація нових користувачів: Система повинна дозволяти новим користувачам створювати обліковий запис, вказуючи ім'я, електронну адресу, пароль та інші необхідні дані.
    - Авторизація користувачів: Користувачі повинні мати можливість входити у свої облікові записи за допомогою електронної адреси та пароля.
    - Відновлення пароля: Система повинна забезпечувати можливість відновлення пароля через електронну пошту у випадку його втрати.
2. Редагування профілю
    - Користувачі повинні мати можливість оновлювати свої особисті дані, такі як ім'я, адреса доставки, контактна інформація тощо.

- Перегляд історії замовлень: Користувачі повинні мати доступ до історії своїх замовлень, включаючи деталі кожного замовлення.

3. Каталог товарів

- Перегляд товарів: Користувачі повинні мати можливість переглядати каталог товарів, організований за категоріями, брендами та іншими атрибутами.
- Пошук товарів: Система повинна забезпечувати функціональність пошуку товарів за ключовими словами.
- Фільтрація та сортування: Користувачі повинні мати можливість фільтрувати та сортувати товари за ціною, популярністю, рейтингом та іншими параметрами.
- Деталі товару: Користувачі повинні мати можливість переглядати детальну інформацію про товар, включаючи опис, зображення, ціну, наявність, рейтинги та відгуки.

4. Кошик та оформлення замовлення

- Додавання товарів до кошика: Користувачі повинні мати можливість додавати товари до кошика для подальшого оформлення замовлення.
- Редагування кошика: Користувачі повинні мати можливість змінювати кількість товарів у кошику або видаляти товари з нього.
- Оформлення замовлення: Користувачі повинні мати можливість оформити замовлення, вказуючи адресу доставки, обираючи спосіб оплати та підтверджуючи замовлення.
- Підтвердження замовлення: Після оформлення замовлення користувачі повинні отримувати підтвердження замовлення з усіма деталями.

5. Платежі

- Інтеграція з платіжними системами: Система повинна підтримувати інтеграцію з платіжними сервісами, такими як PayPal, для обробки онлайн-платежів.
- Безпечна обробка платежів: Всі транзакції повинні бути безпечними та захищеними, використовуючи протоколи шифрування.
- Оформлення замовлення: Користувачі повинні мати можливість оформити замовлення, вказуючи адресу доставки, обираючи спосіб оплати та підтверджуючи замовлення.
- Підтвердження замовлення: Після оформлення замовлення користувачі повинні отримувати підтвердження замовлення з усіма деталями.

6. Управління замовленнями

- Обробка замовлень: Адміністратори повинні мати можливість переглядати, обробляти та оновлювати статус замовлень (наприклад, "в обробці", "відправлено", "доставлено").
- Повідомлення про замовлення: Користувачі повинні отримувати повідомлення про статус їхніх замовлень через електронну пошту.

7. Адміністративні функції

- Управління товарами: Адміністратори повинні мати можливість додавати нові товари, редагувати існуючі, змінювати ціни та видаляти товари з каталогу.
- Управління користувачами: Адміністратори повинні мати можливість переглядати інформацію про користувачів, редагувати її, блокувати або видаляти облікові записи.
- Управління відгуками: Адміністратори повинні мати можливість модерації відгуків користувачів, видаляючи неприйнятні або некоректні відгуки.

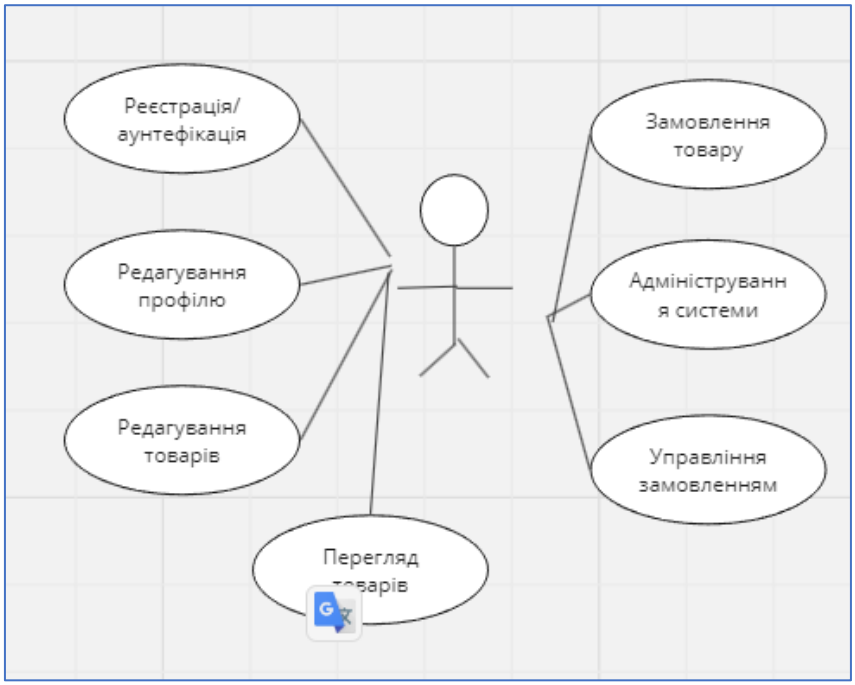

Рисунок 12 — *UseCase діаграма*

Отже функціональні вимоги системи забезпечують всебічне покриття основних потреб як кінцевих користувачів, так і адміністративного персоналу. Вони включають реєстрацію та аутентифікацію користувачів, управління профілем, роботу з каталогом товарів, кошиком та замовленнями, обробку платежів, адміністративні функції та можливості для звітності та аналітики. Це дозволяє створити повноцінний і зручний у використанні веб-додаток для електронної комерції.

3.4 Вимоги до апаратного та програмного забезпечення

Для ефективного функціонування веб-додатку для електронної комерції необхідно забезпечити відповідні апаратні та програмні ресурси як на стороні сервера, так і на стороні клієнта. Вимоги до апаратного та програмного забезпечення включають необхідні конфігурації серверів, клієнтських пристроїв,

а також інструменти та платформи, що використовуються для розробки, тестування та підтримки системи.

Вимоги:

- Процесор: Багатоядерний процесор (мінімум 4 ядра) для забезпечення обробки численних запитів одночасно.
- Оперативна пам'ять: Мінімум 16 ГБ оперативної пам'яті для забезпечення ефективної роботи серверних процесів і обробки запитів.
- Мережева карта: Гігабітна мережева карта для забезпечення високої швидкості передачі даних між сервером та клієнтами.
- Операційна система: Windows 10 або новіша, macOS 10.15 або новіша, або сучасні дистрибутиви Linux.
- Веб-браузер: Останні версії Google Chrome, Mozilla Firefox, Microsoft Edge або Safari.
- Місце на диску: Мінімум 256 ГБ SSD.
- Веб-браузер: Останні версії Google Chrome, Mozilla Firefox, Microsoft Edge або Safari.

Вимоги до апаратного та програмного забезпечення для веб-додатку з електронної комерції включають потужні серверні ресурси для обробки численних запитів та зберігання даних, а також сучасні клієнтські пристрої для забезпечення зручної та швидкої роботи з додатком. Крім того, важливими є програмні інструменти для розробки, тестування та підтримки додатку, які забезпечують ефективність розробки та високу якість кінцевого продукту.

3.5 Модулі та алгоритми

Алгоритми:

У цьому проекті використовуються кілька ключових алгоритмів, які забезпечують функціонування веб-застосунку та взаємодію користувачів з системою:

1. Алгоритм аутентифікації та авторизації

   Цей алгоритм забезпечує безпечний вхід користувачів у систему та розподіл прав доступу залежно від їх ролей.

   Ключові етапи:

   - Користувач вводить свої облікові дані та надсилає їх на сервер.
   - Сервер перевіряє облікові дані у базі даних.
   - Якщо облікові дані коректні, сервер генерує токен доступу (JWT) та надсилає його користувачеві.
   - Користувач зберігає токен та використовує його для доступу до захищених ресурсів.
   - Сервер перевіряє токен на кожному запиті до захищених ресурсів для підтвердження прав доступу.

2. Алгоритм управління кошиком

   Цей алгоритм забезпечує додавання, видалення та оновлення продуктів у кошику користувача.

   Ключові етапи:

   - Користувач вибирає продукт та додає його до кошика.
   - Сервер отримує запит та оновлює інформацію про кошик у базі даних.
   - Користувач може змінювати кількість продуктів або видаляти їх з кошика.

- Сервер синхронізує зміни з базою даних та оновлює інтерфейс користувача.

3. Алгоритм управління замовленнями

Цей алгоритм обробляє процес створення, оновлення та видалення замовлень користувачів.

Ключові етапи:

- Користувач оформлює замовлення, вибираючи продукти з кошика.
- Сервер зберігає замовлення у базі даних та надсилає підтвердження користувачеві.
- Адміністратор та продавець мають можливість переглядати, оновлювати та керувати статусом замовлень.
- Користувач отримує сповіщення про зміни статусу замовлення.

4. Алгоритм управління відгуками

Цей алгоритм дозволяє користувачам залишати відгуки про продукти та управляти ними.

Ключові етапи:

- Користувач залишає відгук про продукт.
- Сервер перевіряє відгук та зберігає його у базі даних.
- Адміністратор або продавець можуть модерувати відгуки, оновлювати або видаляти їх за необхідністю.
- Відгуки відображаються на сторінці продукту для інших користувачів.

5. Алгоритм отримання всіх товарів з фільтрацією

Цей алгоритм забезпечує отримання списку всіх товарів з можливістю фільтрації за різними критеріями, такими як категорія, бренд, ціна тощо.

Ключові етапи:

- Користувач вибирає критерії фільтрації на веб-інтерфейсі.

- Сервер отримує запит з параметрами фільтрації.
- Сервер формує запит до бази даних з урахуванням заданих критеріїв.
- Результати запиту повертаються на сервер та відправляються користувачеві.
- Веб-інтерфейс відображає відфільтрований список товарів.

6. Алгоритм створення мерчантів

Цей алгоритм дозволяє адміністраторам додавати нових мерчантів у систему.

Ключові етапи:

- Адміністратор заповнює форму з даними мерчанта.
- Сервер отримує запит на створення нового мерчанта.
- Сервер перевіряє дані та зберігає інформацію про мерчанта у базі даних.
- Сервер відправляє відповідь адміністратору про успішне створення мерчанта.

7. Алгоритм створення брендів

Цей алгоритм дозволяє адміністраторам додавати нові бренди у систему.

Ключові етапи:

- Адміністратор заповнює форму з даними бренду.
- Сервер отримує запит на створення нового бренду.
- Сервер перевіряє дані та зберігає інформацію про бренд у базі даних.
- Сервер відправляє відповідь адміністратору про успішне створення бренду.

Перелік модулів (server):

1.models/contact.js: Модель для зберігання контактних даних користувачів.

2.models/merchant.js: Модель для зберігання інформації про продавців.

3.models/user.js: Модель для зберігання інформації про користувачів.

4.models/review.js: Модель для зберігання відгуків користувачів про продукти.

5.models/product.js: Модель для зберігання інформації про продукти.

6.models/wishlist.js: Модель для зберігання інформації про бажання користувачів.

7.models/order.js: Модель для зберігання інформації про замовлення.

8.models/brand.js: Модель для зберігання інформації про бренди.

9.models/category.js: Модель для зберігання інформації про категорії продуктів.

10.routes/contact.js: Роут для обробки запитів, пов'язаних з контактними даними.

11.routes/merchant.js: Роут для обробки запитів, пов'язаних з продавцями.

12.routes/user.js: Роут для обробки запитів, пов'язаних з користувачами.

13.routes/review.js: Роут для обробки запитів, пов'язаних з відгуками.

14.routes/product.js: Роут для обробки запитів, пов'язаних з продуктами.

15.routes/wishlist.js: Роут для обробки запитів, пов'язаних з бажаннями користувачів.

16.routes/order.js: Роут для обробки запитів, пов'язаних з замовленнями.

17.routes/brand.js: Роут для обробки запитів, пов'язаних з брендами.

19.routes/category.js: Роут для обробки запитів, пов'язаних з категоріями продуктів.

Перелік модулів (client):

20.components/Manager/AddMerchant.js: Компонент для додавання продавців.

21.components/Manager/EditMerchant.js: Компонент для редагування інформації

22.про продавців.

23.components/Manager/ListMerchant.js: Компонент для відображення списку

24.продавців.

25.components/Manager/AddProduct.js: Компонент для додавання продуктів.

26.components/Manager/EditProduct.js: Компонент для редагування продуктів.

27.components/Manager/ListProduct.js: Компонент для відображення списку продуктів.

28.components/Manager/AddCategory.js: Компонент для додавання категорій.

29.components/Manager/EditCategory.js: Компонент для редагування категорій.

30.components/Manager/ListCategory.js: Компонент для відображення списку категорій.

31.components/Manager/AddBrand.js: Компонент для додавання брендів.

32.components/Manager/EditBrand.js: Компонент для редагування брендів.

33.components/Manager/ListBrand.js: Компонент для відображення списку брендів.

34.components/Store/ProductList.js: Компонент для відображення списку продуктів у магазині.

35.components/Store/ProductDetails.js: Компонент для відображення детальної інформації про продукт.

36.components/Store/Cart.js: Компонент для управління кошиком користувача.

37.components/Store/Checkout.js: Компонент для оформлення замовлення.

38.components/Common/Input.js: Загальний компонент для введення даних.

39.components/Common/Button.js: Загальний компонент для кнопок.

40.components/Common/LoadingIndicator.js: Компонент для відображення індикатора завантаження.

41.components/Common/NotFound.js: Компонент для відображення сторінки 404.

Детальний опис основних модулів:

Server

1. keys.js

Файл keys.js містить конфігураційні параметри для всього застосунку. Він експортує об'єкт з різними налаштуваннями, такими як URL-адреси API, рядки підключення до бази даних, налаштування JWT та облікові дані для різних сторонніх сервісів.

Основні функції:

- Конфігурація застосунку (ім'я, API URL, клієнтський URL)
- Порт для сервера
- Налаштування бази даних (URL MongoDB)
- Налаштування JWT (секретний ключ і строк дії токена)
- Налаштування сторонніх сервісів (Mailchimp, Mailgun, Google, Facebook, AWS)

Лістинг 1: *Конфігурація JWT*

```
const config = {
  application: {
    appName: 'EcommerceApp',
    apiEndpoint: `${process.env.API_BASE_URL}`,
    frontendURL: process.env.FRONTEND_URL
  },
  authentication: {
    jwtKey: process.env.SECRET_KEY,
    jwtExpiration: '7 days'
  },
  serverPort: process.env.SERVER_PORT || 3000,
  dbConfig: {
    connectionString: process.env.DATABASE_URI
  },
};

module.exports = config;
```

2.passport.js

Файл passport.js налаштовує стратегію автентифікації JWT за допомогою Passport.js. Він використовує passport-jwt для автентифікації користувачів на основі JWT токенів, витягнутих із заголовка авторизації.

Основні функції:

- Витягування JWT токена з заголовка авторизації
- Налаштування стратегії JWT з перевіркою користувача в базі даних
- Ініціалізація Passport.js у застосунку

Лістинг 2: *Налаштування стратегії JWT*

```
const passport = require('passport');
const { Strategy: JwtStrategy, ExtractJwt } =
require('passport-jwt');
const mongoose = require('mongoose');

const config = require('./config');
const User = mongoose.model('User');

const jwtSecret = config.authentication.jwtKey;
const jwtOptions = {};
jwtOptions.jwtFromRequest =
ExtractJwt.fromAuthHeaderAsBearerToken();
jwtOptions.secretOrKey = jwtSecret;

passport.use(
  new JwtStrategy(jwtOptions, function (jwtPayload, callback)
{
    User.findById(jwtPayload.id)
      .then(function (user) {
        if (user) {
          return callback(null, user);
```

```
        }
        return callback(null, false);
      })
      .catch(function (error) {
        return callback(error, false);
      });
  })
);

module.exports = async function (application) {
  application.use(passport.initialize());
};
```

3.template.js

Файл template.js містить різні шаблони листів, які використовуються в застосунку для таких дій, як скидання пароля, реєстрація акаунту, реєстрація продавця тощо.

Основні функції:

- Створення шаблонів для скидання пароля, підтвердження скидання пароля, реєстрації продавця, привітання продавця, реєстрації акаунту, підписки на розсилку, контактних повідомлень, заявки продавця, деактивації акаунту продавця, підтвердження замовлення.

Лістинг 3: *Налаштування стратегії JWT*

```
exports.composeResetEmail = (hostAddress, token) => {
  const subject = 'Instructions for Password Reset';
  const intro = 'You have requested to reset your
password.\n\n';
  const instructions = 'Follow the link below or copy it into
your browser to complete the process:\n\n';
```

```
  const link = `http://${hostAddress}/reset-
password/${token}\n\n`;
  const outro = 'If you did not request a password reset,
please ignore this email and your password will stay the
same.\n';

  const emailBody = {
    subject: subject,
    body: `${intro}${instructions}${link}${outro}`
  };
  return emailBody;
};
```

4.auth.js

Файл auth.js налаштовує автентифікацію JWT за допомогою Passport.js.

Основні функції:

- Аутентифікація користувача за допомогою JWT токена.

Лістинг 4: *Аутентифікація за допомогою Passport.js*

```
const passportService = require('passport');

const initJwtAuth = function () {
  const options = { session: false };
  return passportService.authenticate('jwt', options);
};

const jwtMiddleware = initJwtAuth();

module.exports = function (app) {
  app.use(jwtMiddleware);
};
```

5.Файл models/contact.js

Файл contact.js містить схему для зберігання контактних повідомлень користувачів.

Основні функції:

- Зберігання даних про контактні повідомлення.

Лістинг 5: *Схема контактного повідомлення*

```
const mongoose = require('mongoose');
const { Schema } = mongoose;

// Define schema for storing contact information
const ContactSchemaDefinition = {
  fullName: {
    type: String,
    required: true,
    trim: true
  },
  email: {
    type: String,
    required: true
  },
  messageContent: {
    type: String,
    required: true,
    trim: true
  },
  updatedTimestamp: {
    type: Date,
    default: Date.now
  },
  creationTimestamp: {
```

```
    type: Date,
    default: Date.now
  }
};

// Create contact schema
const ContactSchema = new Schema(ContactSchemaDefinition);

// Define model for contact
const ContactModel = mongoose.model('Contact', ContactSchema);

// Function to create a new contact
const createNewContact = async (data) => {
  const contact = new ContactModel(data);
  return await contact.save();
};

// Function to find a contact by email
const findContactByEmail = async (email) => {
  return await ContactModel.findOne({ email });
};

// Function to update a contact by ID
const updateContactById = async (id, data) => {
  return await ContactModel.findByIdAndUpdate(id, data, { new:
true });
};

// Function to delete a contact by ID
const deleteContactById = async (id) => {
  return await ContactModel.findByIdAndDelete(id);
```

```
};

// Export functions and model
module.exports = {
  createNewContact,
  findContactByEmail,
  updateContactById,
  deleteContactById,
  ContactModel
};
```

6.Файл models/merchant.js

Файл merchant.js містить схему для зберігання даних про продавців.

Основні функції:

- Зберігання даних про продавців, включаючи статус, контактні дані та бренд.

Лістинг 6: *Схема продавця*

```
const Mongoose = require('mongoose');
const { MERCHANT_STATUS } = require('../constants');
const { Schema } = Mongoose;
// Merchant Schema
const MerchantSchema = new Schema({
  name: {
    type: String,
    trim: true
  },
  email: {
    type: String
  },
  phoneNumber: {
    type: String
```

```
},
brandName: {
  type: String
},
business: {
  type: String,
  trim: true
},
isActive: {
  type: Boolean,
  default: false
},
brand: {
  type: Schema.Types.ObjectId,
  ref: 'Brand',
  default: null
},
status: {
  type: String,
  default: MERCHANT_STATUS.Waiting_Approval,
  enum: [
    MERCHANT_STATUS.Waiting_Approval,
    MERCHANT_STATUS.Rejected,
    MERCHANT_STATUS.Approved
  ]
},
updated: Date,
created: {
  type: Date,
  default: Date.now
}
```

```
});

module.exports = Mongoose.model('Merchant', MerchantSchema);
```

7.Файл models/order.js

Файл order.js містить схему для зберігання даних про замовлення.

Основні функції:

- Зберігання даних про замовлення, включаючи користувача та товари у замовленні.

Лістинг 7: *Схема замовлення*

```
const mongoose = require('mongoose');
const { MERCHANT_STATUS } = require('../constants');
const { Schema } = mongoose;

// Schema definition for merchant details
const MerchantSchema = new Schema({
  name: {
    type: String,
    trim: true,
    required: true
  },
  email: {
    type: String,
    required: true
  },
  phone: {
    type: String,
    required: true
  },
  brand: {
    type: String,
```

```
  required: true
},
businessType: {
  type: String,
  trim: true,
  required: true
},
isActive: {
  type: Boolean,
  default: false
},
brandRef: {
  type: Schema.Types.ObjectId,
  ref: 'Brand',
  default: null
},
status: {
  type: String,
  default: MERCHANT_STATUS.Waiting_Approval,
  enum: [
    MERCHANT_STATUS.Waiting_Approval,
    MERCHANT_STATUS.Rejected,
    MERCHANT_STATUS.Approved
  ]
},
lastUpdated: {
  type: Date,
  default: Date.now
},
createdOn: {
  type: Date,
```

```
    default: Date.now
  }
});

// Create Merchant model
const Merchant = mongoose.model('Merchant', MerchantSchema);

// Create a new merchant
const createMerchant = async (data) => {
  const merchant = new Merchant(data);
  return await merchant.save();
};

// Find a merchant by email
const getMerchantByEmail = async (email) => {
  return await Merchant.findOne({ email });
};

// Update merchant by ID
const updateMerchantById = async (id, data) => {
  return await Merchant.findByIdAndUpdate(id, data, { new:
true });
};

// Delete merchant by ID
const deleteMerchantById = async (id) => {
  return await Merchant.findByIdAndDelete(id);
};

// Get all merchants
const getAllMerchants = async () => {
```

```
  return await Merchant.find();
};

// Get merchant by ID
const getMerchantById = async (id) => {
  return await Merchant.findById(id);
};

// Export functions and model
module.exports = {
  createMerchant,
  getMerchantByEmail,
  updateMerchantById,
  deleteMerchantById,
  getAllMerchants,
  getMerchantById,
  Merchant
};
```

8.Файл models/product.js

Файл product.js містить схему для зберігання даних про продукти, використовуючи плагін mongoose-slug-generator.

Основні функції:

- Зберігання даних про продукти, включаючи назву, опис, кількість, ціну та бренд.

Лістинг 8: *Схема продукту*

```
const mongoose = require('mongoose');
const slug = require('mongoose-slug-generator');
const { Schema } = mongoose;

const slugOptions = {
```

```
  separator: '-',
  lang: 'en',
  truncate: 120
};

mongoose.plugin(slug, slugOptions);

// Define schema for products
const ProductSchema = new Schema({
  sku: {
    type: String,
    required: true
  },
  name: {
    type: String,
    trim: true,
    required: true
  },
  slug: {
    type: String,
    slug: 'name',
    unique: true
  },
  imageUrl: {
    type: String
  },
  imageKey: {
    type: String
  },
  description: {
    type: String,
```

```
    trim: true
  },
  quantity: {
    type: Number,
    required: true
  },
  price: {
    type: Number,
    required: true
  },
  isTaxable: {
    type: Boolean,
    default: false
  },
  isActive: {
    type: Boolean,
    default: true
  },
  brandReference: {
    type: Schema.Types.ObjectId,
    ref: 'Brand',
    default: null
  },
  updatedAt: {
    type: Date,
    default: Date.now
  },
  createdAt: {
    type: Date,
    default: Date.now
  }
```

```
});

// Create Product model
const Product = mongoose.model('Product', ProductSchema);

// Function to create a new product
const createProduct = async (data) => {
  const product = new Product(data);
  return await product.save();
};

// Function to find a product by slug
const getProductBySlug = async (slug) => {
  return await Product.findOne({ slug });
};

// Function to update product by ID
const updateProductById = async (id, data) => {
  return await Product.findByIdAndUpdate(id, data, { new: true
});
};

// Function to delete product by ID
const deleteProductById = async (id) => {
  return await Product.findByIdAndDelete(id);
};

// Export functions and model
module.exports = {
  createProduct,
  getProductBySlug,
```

```
  updateProductById,
  deleteProductById,
  Product
};
```

9.Файл routes/contact.js

Файл contact.js містить контролер для обробки контактних повідомлень від користувачів.

Основні функції:

- Додавання нового контактного повідомлення
- Відправка підтвердження на email користувача

Лістинг 9: *Додавання нового контактного повідомлення*

```
const express = require('express');
const router = express.Router();
const ContactModel = require('../../models/contact');
const mailgunService = require('../../services/mailgun');

router.post('/add', async (req, res) => {
  try {
    const { name, email, message } = req.body;

    if (!email) {
      return res.status(400).json({ error: 'You must enter an
email address.' });
    }

    if (!name) {
      return res.status(400).json({ error: 'You must enter
description & name.' });
    }
```

```
    if (!message) {
      return res.status(400).json({ error: 'You must enter a
message.' });
    }

    const existingContact = await ContactModel.findOne({ email
});

    if (existingContact) {
      return res.status(400).json({ error: 'A request already
existed for the same email address' });
    }

    const newContact = new ContactModel({
      name,
      email,
      message
    });

    const savedContact = await newContact.save();

    await mailgunService.sendEmail(email, 'contact');

    res.status(200).json({
      success: true,
      message: `We received your message, we will reach you on
your email address ${email}!`,
      contact: savedContact
    });
  } catch (error) {
```

```
    return res.status(400).json({
      error: 'Your request could not be processed. Please try
again.'
    });
  }
});

module.exports = router;
```

10.Файл routes/merchant.js

Файл merchant.js містить контролер для обробки даних продавців.

Основні функції:

- Додавання нового продавця
- Пошук продавців
- Оновлення статусу активності продавця
-

Лістинг 10: *Додавання нового продавця*

```
const express = require('express');
const router = express.Router();
const { MERCHANT_STATUS, ROLES } = require('../../constants');
const MerchantModel = require('../../models/merchant');
const emailService = require('../../services/mailgun');

// Handler for new merchant registration
router.post('/create', async (req, res) => {
  try {
    const { merchantName, businessInfo, contactNumber,
emailAddress, brandLabel } = req.body;

    // Ensure all required fields are filled
```

```
    if (!merchantName || !emailAddress || !contactNumber || !businessInfo) {
      return res.status(400).json({ error: 'Merchant name, email address, contact number, and business information are required.' });
    }

    // Check if the merchant already exists in the database
    const foundMerchant = await MerchantModel.findOne({ emailAddress });
    if (foundMerchant) {
      return res.status(400).json({ error: 'This email address is already registered with another merchant.' });
    }

    // Create a new merchant entry
    const merchantData = new MerchantModel({
      merchantName,
      emailAddress,
      businessInfo,
      contactNumber,
      brandLabel
    });
    const createdMerchant = await merchantData.save();

    // Send a registration confirmation email
    await emailService.sendEmail(emailAddress, 'Welcome Aboard!', 'merchant-registration-success');

    // Return a success response
    res.status(201).json({
```

```
      success: true,
      message: `Registration complete! We will reach out to you
at ${contactNumber}.`,
      merchant: createdMerchant
    });
  } catch (error) {
    // Handle any errors during the process
    return res.status(500).json({
      error: 'An unexpected error occurred while processing
your registration. Please try again later.'
    });
  }
});

module.exports = router;
```

Client

12. components/Manager/ListProduct.js

Компонент для відображення списку продуктів.

Цей компонент відображає список всіх продуктів, дозволяючи адміністраторам переглядати їх та керувати ними.

Основні функції:

- Завантажує список продуктів з сервера.
- Відображає інформацію про кожен продукт.

Лістинг 11: *Відображення списку товарів*

```
/**
 *
 * ProductList
 *
```

```
 */

import React from 'react';

import { Link } from 'react-router-dom';

const ProductList = props => {
  const { products } = props;

  return (
    <div className='p-list'>
      {products.map((product, index) => (
        <Link
          to={`/dashboard/product/edit/${product._id}`}
          key={index}
          className='d-flex flex-row align-items-center mx-0
mb-3 product-box'
        >
          <img
            className='item-image'
            src={`${
              product && product.imageUrl
                ? product.imageUrl
                : '/images/placeholder-image.png'
            }`}
          />
          <div className='d-flex flex-column justify-content-
center px-3 text-truncate'>
            <h4 className='text-truncate'>{product.name}</h4>
            <p className='mb-2 text-
truncate'>{product.description}</p>
```

```
          </div>
        </Link>
      ))}
    </div>
  );
};

export default ProductList;
```

13. components/Manager/AddProduct.js

Компонент для додавання продуктів.

Цей компонент дозволяє адміністраторам додавати нові продукти. Він містить форму для введення даних про продукт та обробляє події зміни та відправки форми.

Основні функції:

- Обробляє зміну значення у полях форми.
- Відправляє форму для створення нового продукту.
- Лістинг 12: *Відображення додавання параметру*

```
import React, { useState } from 'react';
import { useDispatch } from 'react-redux';
import { addProduct } from '../../actions/productActions';

const AddProduct = () => {
  const [formData, setFormData] = useState({ name: '', description: '', price: '', quantity: '', brand: '', category: '' });
  const dispatch = useDispatch();

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

```
};

const handleSubmit = (e) => {
  e.preventDefault();
  dispatch(addProduct(formData));
};

return (
  <form onSubmit={handleSubmit}>
    <input
      type="text"
      name="name"
      value={formData.name}
      onChange={handleInputChange}
      placeholder="Enter product name"
      required
    />
    <textarea
      name="description"
      value={formData.description}
      onChange={handleInputChange}
      placeholder="Enter product description"
      required
    />
    <input
      type="number"
      name="price"
      value={formData.price}
      onChange={handleInputChange}
      placeholder="Enter product price"
      required
```

```
        />
        <input
          type="number"
          name="quantity"
          value={formData.quantity}
          onChange={handleInputChange}
          placeholder="Enter product quantity"
          required
        />
        <input
          type="text"
          name="brand"
          value={formData.brand}
          onChange={handleInputChange}
          placeholder="Enter product brand"
          required
        />
        <input
          type="text"
          name="category"
          value={formData.category}
          onChange={handleInputChange}
          placeholder="Enter product category"
          required
        />
        <button type="submit">Add Product</button>
      </form>
    );
};

export default AddProduct;
```

3.6 Проект інтерфейсу

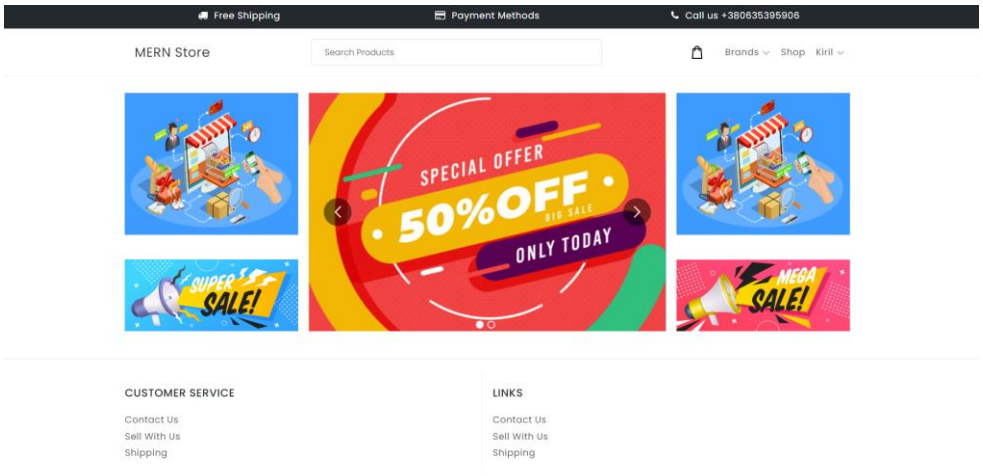

Рисунок 13 — *Головна сторінка*

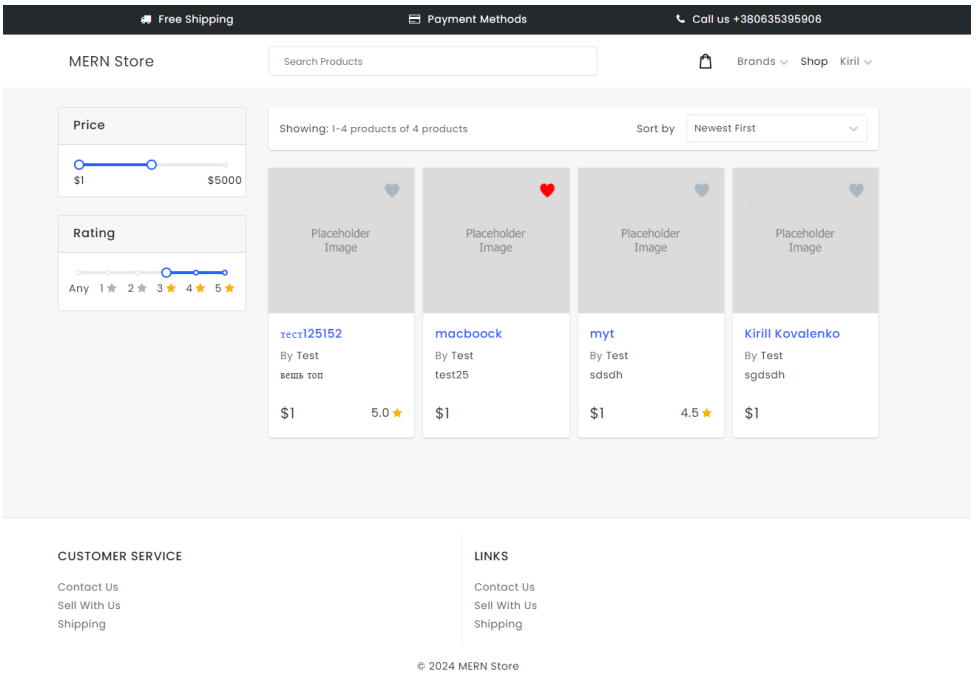

Рисунок 14 — *Сторінка каталогу товарів*

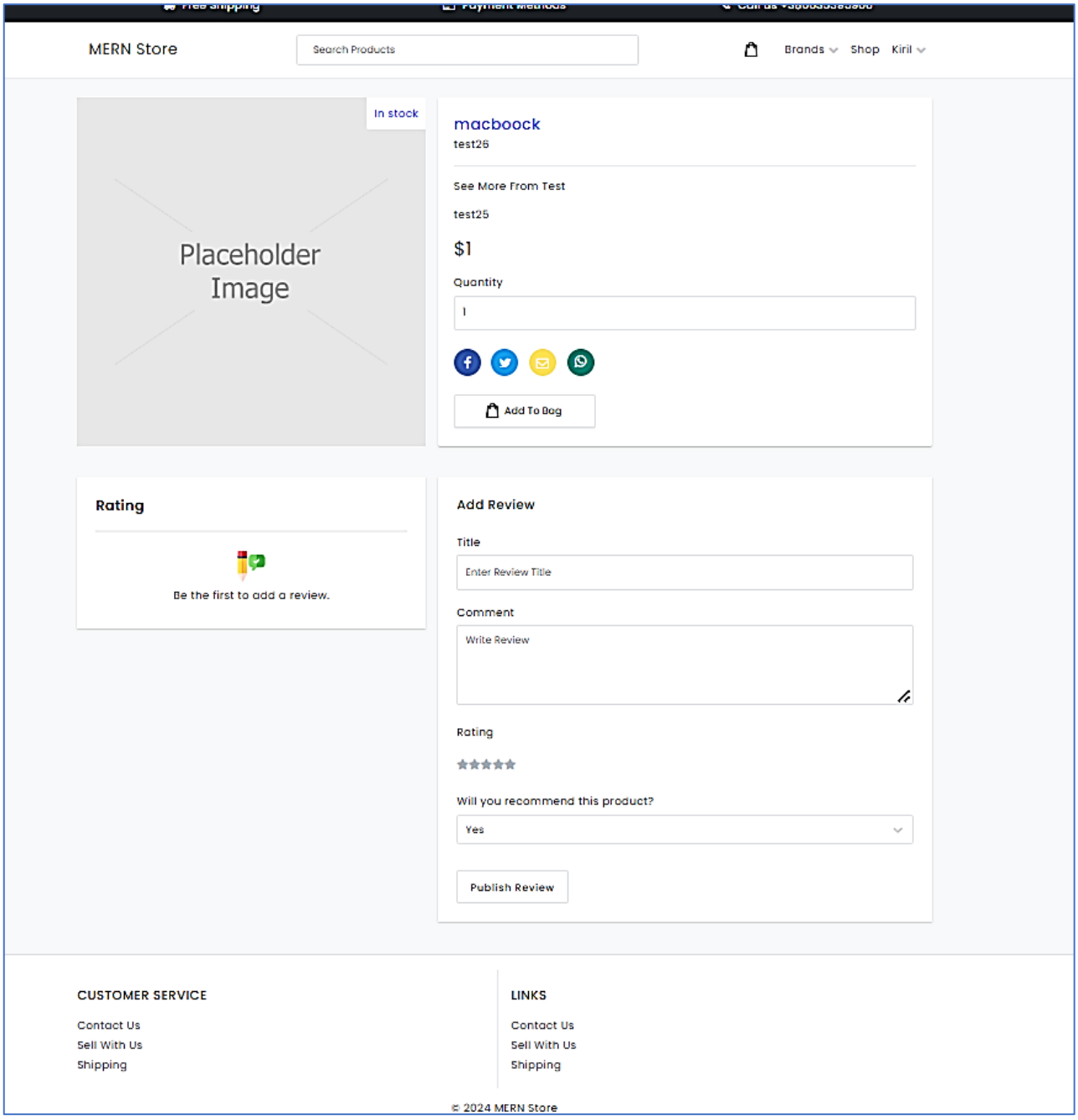

Рисунок 15 — *Сторінка детального огляду товару*

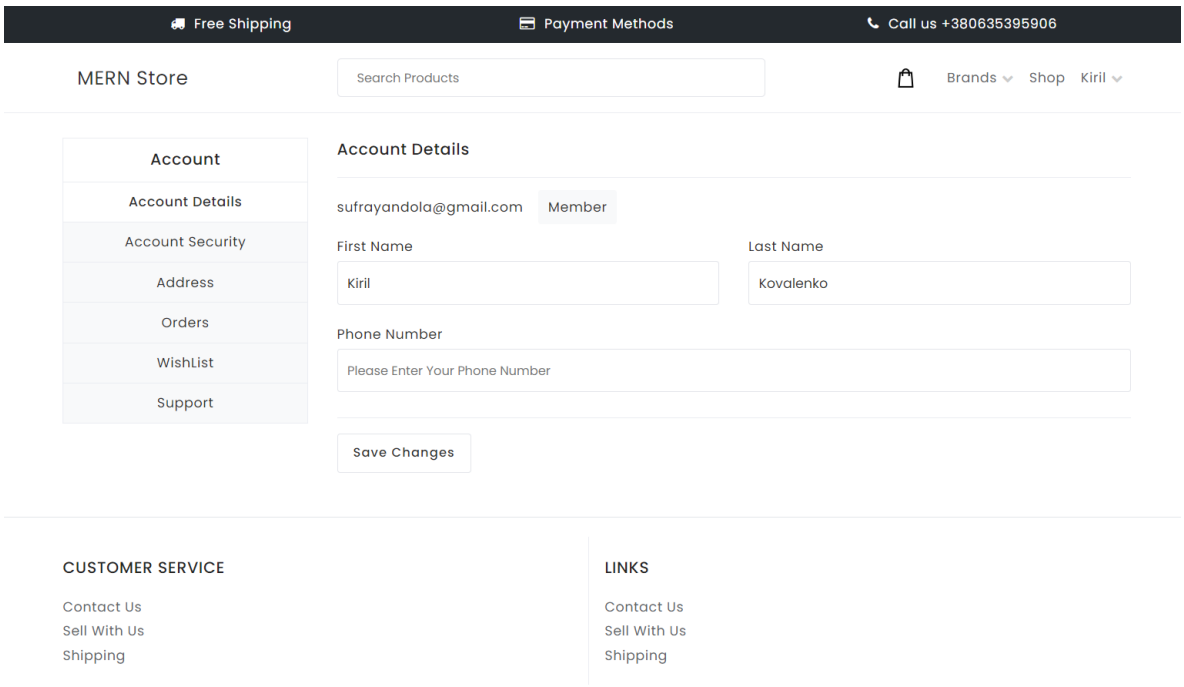

Рисунок 16 — *Сторінка особистого кабінету*

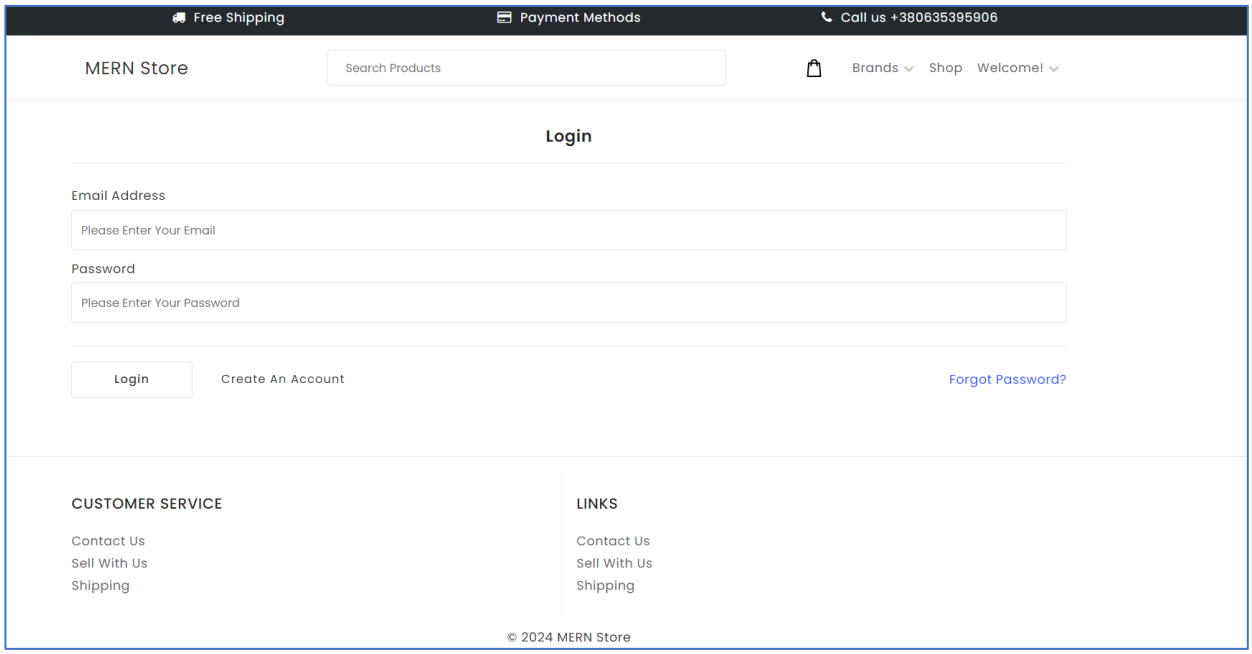

Рисунок 17 — *Сторінка авторизації*

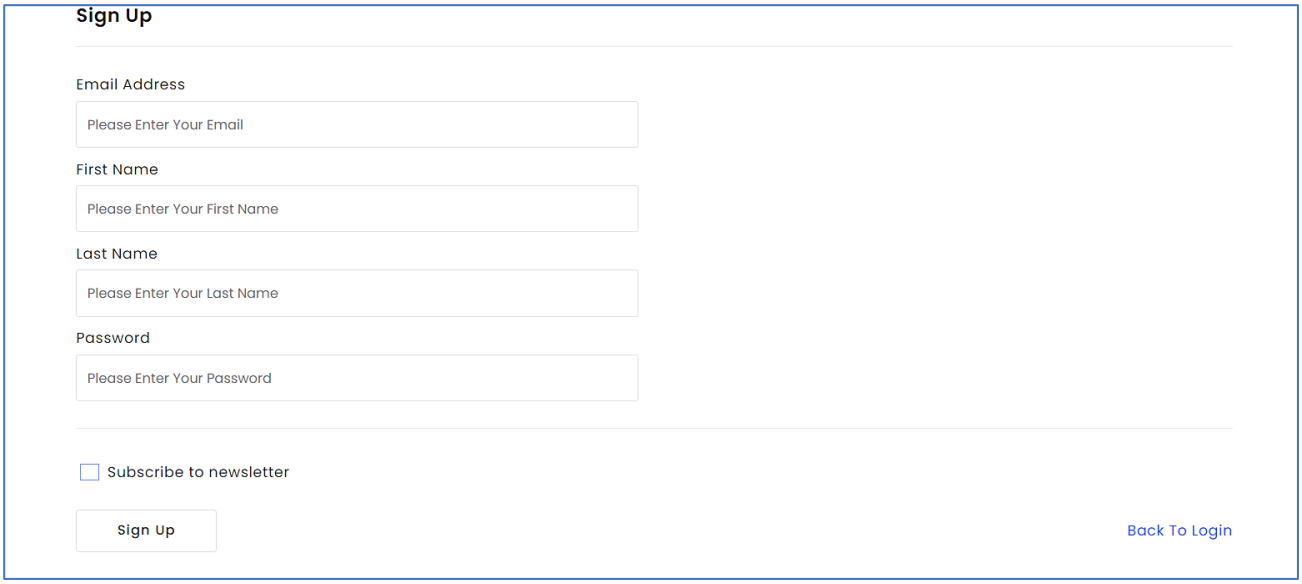

Рисунок 18 - *Сторінка регистрації*

3.7 Тестування

Для забезпечення надійної роботи онлайн-магазину на стеці MERN було проведено всебічне тестування. Основні аспекти тестування включали:

Тестування функціональності:

- Реєстрація та вхід користувача: Перевірка можливості створення нового облікового запису, входу та виходу з системи.
- Перегляд та пошук товарів: Тестування можливості перегляду списку товарів, використання фільтрів та пошукової функції.
- Кошик та замовлення: Перевірка функціональності додавання товарів до кошика, оформлення замовлення та перегляду історії замовлень.

- Управління обліковим записом: Перевірка можливості користувача редагувати свій профіль, додавати адреси доставки та переглядати історію замовлень.

Тестування безпеки:

- Аутентифікація та авторизація: Перевірка правильності роботи механізмів аутентифікації та авторизації для різних ролей користувачів (адміністратор, мерчант, покупець).

Захист від CSRF та XSS атак:

- Перевірка, чи захищений веб-застосунок від атак типу Cross-Site Request Forgery (CSRF) та Cross-Site Scripting (XSS).

Шифрування паролів:

- Перевірка правильності шифрування паролів користувачів.

Тестування продуктивності:

- Час завантаження сторінок: Вимірювання часу завантаження основних сторінок веб-застосунку для забезпечення швидкого доступу до інформації.
- Обробка великої кількості запитів: Перевірка здатності сервера обробляти велику кількість одночасних запитів без втрати продуктивності.
- Оптимізація запитів до бази даних: Перевірка ефективності запитів до бази даних, оптимізація індексів та зменшення кількості запитів.
- Тестування інтерфейсу користувача:

Зручність використання:

- Оцінка зручності навігації по веб-застосунку, зрозумілість інтерфейсу та його доступність.

Адаптивний дизайн:

- Перевірка коректного відображення інтерфейсу на різних пристроях (мобільних телефонах, планшетах, настільних комп'ютерах).

Тестування сумісності:

- Переглядачі: Перевірка коректної роботи веб-застосунку у різних браузерах (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari).
- Операційні системи: Тестування на різних операційних системах (Windows, macOS, Linux).

Проведене тестування дозволило виявити та виправити потенційні проблеми у роботі веб-застосунку, забезпечуючи його стабільність та надійність у використанні.

ВИСНОВКИ

1. Розробка веб-застосунку на стекові MERN**:** У ході даної роботи було успішно створено онлайн-магазин, який базується на стекові MERN (MongoDB, Express, React, Node.js). Це забезпечило потужну і гнучку архітектуру для побудови сучасного веб-застосунку.

2. Функціональність веб-застосунку:

- Веб-застосунок надає користувачам можливість:
- Переглядати та шукати продукти.
- Купувати товари.
- Взаємодіяти з продавцями.
- Залишати відгуки.
- Керувати кошиком та замовленнями.
- Використовувати систему автентифікації та авторизації для різних ролей користувачів.

3. Адміністрування та продавці:

- Користувачі можуть створювати облікові записи, додавати продукти до списку бажань.
- Продавці мають можливість додавати та редагувати свої товари.
- Адміністратори мають зручний інтерфейс для управління замовленнями та відгуками.

4. Тестування та порівняння:

- Проведено тестування працездатності створеного веб-застосунку. Результати показали високу продуктивність та надійність роботи системи. Функціональність застосунку було порівняно з наявними ринковими рішеннями, що підтвердило його конкурентоспроможність.

5. Висока продуктивність та надійність:

- Створений веб-застосунок відповідає сучасним вимогам та здатен ефективно обробляти запити користувачів, забезпечуючи стабільну роботу під час високих навантажень.

6. Покращення та розвиток в майбутньому:

- Розширення функціональності:
    - Планується додавання нових можливостей, таких як підтримка кількох мов, інтеграція з іншими платіжними системами та соціальними мережами, а також розвиток системи рекомендацій для користувачів.
- Покращення UI/UX:
    - Впровадження більш сучасних та інтуїтивно зрозумілих інтерфейсів для зручності користувачів.
- Оптимізація продуктивності:
    - Подальша оптимізація бази даних та серверної частини для забезпечення швидкої обробки запитів.
- Покращення безпеки:
    - Вдосконалення системи безпеки, включаючи захист від потенційних загроз та покращення процесів автентифікації та авторизації.
- Мобільні додатки:
    - Розробка мобільних додатків для iOS та Android, що дозволить користувачам зручно користуватися сервісом з мобільних пристроїв.
- Аналітика та звіти:
    - Впровадження системи аналітики для відстеження поведінки користувачів та формування детальних звітів для продавців та адміністраторів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Chodorow, K. (2013). MongoDB: The Definitive Guide (2nd ed.). O'Reilly Media.
2. Brown, E. (2014). Web Development with Node and Express: Leveraging the JavaScript Stack. O'Reilly Media.
3. Banks, A., & Porcello, E. (2017). Learning React: A Hands-On Guide to Building Web Applications Using React and Redux. O'Reilly Media.
4. Cantelon, M., Harter, M., Holowaychuk, T., & Rajlich, N. (2014). Node.js in Action. Manning Publications.
5. Gibson, A. (2017). Node.js 8 the Right Way: Practical, Server-Side JavaScript That Scales. Pragmatic Bookshelf.
6. Axios Documentation. Axios site. URL: https://axios-http.com/docs/intro (дата звернення 14.01.2024).
7. Duckett, J. (2014). *Web Development and Design Foundations with HTML5* (7th ed.). Pearson.
8. Node.js Documentation. Node.js site. URL: https://nodejs.org/docs/latest/api/ (дата звернення 14.01.2024).
9. Gepp, J., & Adams, D. (2019). *Testing Node.js Applications*. Packt Publishing.
10. Murach's Node.js by Joel Murach, Michael Urban.
11. Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node.
12. Fullstack React: The Complete Guide to ReactJS and Friends.
13. MongoDB Applied Design Patterns by Rick Copeland.
14. Express in Action: Writing, building, and testing Node.js applications by Evan Hahn.
15. Learning Node.js: A Hands-On Guide to Building Web Applications in JavaScript by Marc Wandschneider.
16. The Complete Node.js Developer Course (3rd Edition) by Andrew Mead.

Декларація

академічної доброчесності

здобувача ступеня вищої освіти ЗНУ

Я, Коваленко Кирило Максимович, студент 4 курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти ipz20bd-104@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Створення онлайн-магазину побутових товарів з використання React і NodeJS**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 17.06.2024 Підпис _____ Коваленко Кирило Максимович
(студент)

Дата 18.06.2024 Підпис _____ Попівщий Василь Іванович
(науковий керівник)