МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА КОМПОНЕНТІВ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ПІДТРИМКИ ІНТЕРНЕТ МАГАЗИНУ»**

Виконав: студент 4 курсу, групи 6.1210-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
О.О. Новіков
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, доцент, к.т.н. Мухін В.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Новікову Олегу Олеговичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка компонентів інформаційної системи для підтримки інтернет магазину

керівник роботи Мухін Віталій Вікторович, к.т.н, доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Проєктування та реалізація серверної та вебчастини програмного забезпечення.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

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

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.01.2024	
2.	Збір вихідних даних.	25.01.2024	
3.	Обробка методичних та теоретичних джерел.	16.02.2024	
4.	Розробка першого та другого розділу.	29.03.2024	
5.	Розробка третього та четвертого розділу.	26.04.2024	
6	Розробка п’ятого розділу.	20.05.2024	
7.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
8.	Захист кваліфікаційної роботи.	20.06.2024	

Студент _____ О.О. Новіков
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.В. Мухін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка компонентів інформаційної системи для підтримки інтернет магазину»: 44 с., 24 рис., 2 табл., 20 джерел, 2 додатки.

БАЗА ДАНИХ, ІНТЕРФЕЙС ДОСТУПУ, MONGO, NODE JS, REACT JS.

Об’єкт дослідження – процес продажу сувенірної продукції, оформлення замовлень, документування облікових та статистичних даних.

Мета роботи: розробка клієнтської та серверної частини інформаційної системи продажу сувенірів.

Методи дослідження – аналіз предметної області, модулювання предметної області, об’єктно-орієнтовний підхід.

Результатом роботи є розроблений програмний модуль, що дозволяє створювати запити на продукції, переглядати інформацію про замовлення та проводити оплату.

SUMMARY

Bachelor’s qualifying paper “Development of an Information System Components for the Online Shop Supporting”: 44 pages, 24 figures, 2 tables, 20 references, 2 supplements.

ACCESS INTERFACE, DATABASE, MONGO, NODE JS, REACT JS.

The object of the study is the process of selling souvenir products, order processing, documenting accounting and statistical data.

The aim of the study is development of the client and server parts of the souvenir sales information system.

The methods of research are domain analysis, domain modeling, object-oriented approach.

The result of the work is a developed software module that allows creating product requests, viewing order information, and making payments.

ЗМІСТ

ВСТУП

За останніми даними, кількість користувачів інтернету стрімко зростає, а частка онлайн-продажів у великих містах досягає 40% [1]. Фахівці підкреслюють, що ця тенденція є стійкою. Кожного року кількість інтернет-магазинів збільшується, оскільки вони є вигідними та зручними для споживачів, дозволяючи їм економити час і гроші. Інтернет-магазини працюють цілодобово, а автоматизовані системи дозволяють продавати товари без участі продавця. Серед переваг також відсутність необхідності тримати товар на складі заздалегідь, що знижує витрати на зберігання [13]. Достатньо узгодити умови з постачальниками та замовляти товари по мірі отримання замовлень.

Електронний магазин – це система, що функціонує на основі технологій електронної комерції. Вона виконує ті ж функції, що й традиційний магазин: презентація товарів (послуг) покупцям, обробка замовлень, продаж та доставка товарів [12].

Основні складнощі у створенні інтернет-магазину виникають на перетині інтернет-технологій та традиційної комерційної діяльності. У традиційній торгівлі покупці звикли оглядати товар, оцінювати його якість і характеристики. В електронній комерції ця можливість відсутня. Візуальної інформації часто недостатньо, адже на рішення впливають емоційні та психологічні фактори. Також можуть виникати проблеми з доставкою товарів, особливо якщо їхня вартість невелика [5].

Метою цієї кваліфікаційної роботи є створення інтернет-магазину для продажу сувенірної продукції. Ця система буде працювати в межах певного підприємства та служитиме основною платформою для комерційної діяльності.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Електронна комерція та електронний бізнес стали ключовими чинниками конкурентоспроможності компаній по всьому світу. Вони є як наслідком, так і рушійною силою глобалізації. Сфера електронної комерції та електронного бізнесу сприяє прозорості цін, прискорює інноваційні процеси та підтримує зростання компаній. Сьогодні це повинно стати основою будь-якої бізнес-стратегії, орієнтованої на майбутнє [13].

Електронна комерція має низку важливих переваг перед традиційними видами торгівлі, серед яких [8]:

– точність;
– швидкість;
– безперервність;
– комунікація з клієнтом;
– гнучкість;
– знижені транзакційні витрати;
– короткі терміни доставки.

Ефективні відносини з управління та доставки, які відповідають вимогам клієнтів, сьогодні важко уявити без електронної комерції, і в майбутньому це стане ще важливішим. Електронна комерція підтримує не лише великі компанії, а й малі та середні підприємства, надаючи їм можливість швидко стати глобальними гравцями. У приватному та бізнес-секторах транзакції через Інтернет або мобільні пристрої стали незамінними [7].

Незалежно від того, що ви бажаєте придбати, продати, застрахувати чи дізнатися – зараз майже всі види товарів і послуг представлені в онлайн-торгівлі. Від автомобілів до продуктів – майже все можна купити, не виходячи з дому [8].

1.2 Ідентифікація проблем та розробка актуальних рішень

Однією з основних проблем сучасних інтернет-магазинів є низька швидкість роботи. Це стосується як програмної частини вебдодатку, так і швидкості доставки та обробки замовлень. Нещодавно Amazon провели масштабне дослідження, яке показало, що при збільшенні часу відгуку сторінки на кожні 100 мілісекунд втрачається 1% продажів [9]. Багато бізнесменів-початківців навіть не підозрюють, що їх сайт працює повільно, і не розуміють, що це є головною причиною проблем [10].

Швидкість роботи сайтів є критично важливою у сучасній електронній комерції, оскільки більшість користувачів заходять на сайти з мобільних пристроїв. На жаль, мобільний зв’язок не завжди встигає за швидкістю сучасного дротового Інтернету. Тому для забезпечення доступу до сайту для якомога більшої кількості користувачів потрібно оптимізувати швидкість роботи як front-end, так і back-end частини [11].

Крім того, успіх продажів залежить від швидкості обробки та доставки замовлень. Якщо швидкість роботи сайту буде прийнятною, але обробка та доставка замовлень затягуватиметься, є ймовірність, що клієнти звернуться до іншого магазину. Цю тенденцію задають великі компанії, такі як Amazon, Aliexpress, або українські сайти як Rozetka [12].

Велика кількість публікацій на цю тему свідчить про активну роботу у сфері, проте знайти абсолютно нові рішення досить складно. Більшість існуючих рішень індивідуалізовані та використовуються для просування власних продуктів, оскільки сфера інтернет-комерції постійно розвивається і варіативна [13].

Для вирішення проблем інтернет-комерції у сфері друкованої продукції, необхідно створити програмний модуль, який відповідатиме вимогам оптимізації вебсервісів, забезпечуватиме безпеку передачі даних користувачів та буде зручним для кінцевих користувачів [14].

Для вирішення проблеми інтернет-комерції у сфері друкованої продукції, необхідно створити програмний модуль, який буде відповідати вимогам оптимізації вебсервісів, буде безпечним для передачі даних користувача у мережі та буде зручним для кінцевого користувача.

1.3 Постановка задачі

Аналіз предметної області виявив потребу у створенні програмного модуля для продажу друкованої продукції через швидкий та адаптивний інтернет-магазин.

Функціональні можливості для відвідувачів інтернет-магазину включають:

– авторизацію;
– реєстрацію;
– перегляд списку товарів;
– замовлення продукції на друк;
– додавання товарів до кошика;
– видалення товарів з кошика.

Функціональні можливості для покупців інтернет-магазину включають:

– авторизація;
– реєстрація;
– перегляд профілю;
– перегляд списку товарів;
– перегляд історії замовлень;
– оформлення замовлення на покупку;
– оплата замовленого товару;
– додавання товарів до кошика;
– видалення товарів з кошика.

Функціональні можливості для адміністратора інтернет-магазину включають:

– авторизація;
– перегляд списку товарів у системі;
– перегляд списку замовлень у системі;
– зміна статусу замовлення;
– видалення товарів з системи;
– додавання товарів до системи;
– представлення знижок на товари.

Завдяки цим функціональним можливостям буде реалізовано вебсервіс.

2 РОЗРОБКА ВИМОГ ДО СИСТЕМИ

2.1 Розробка системних вимог до інформаційної системи

Програмний модуль для продажу друкованої продукції має включати такі компоненти:

– вебзастосунок;
– серверну частину;
– базу даних.

Вебзастосунок повинен забезпечувати весь необхідний функціонал модулю. Серверна та клієнтська частини повинні бути розроблені як окремі проєкти, з API-інтерфейсом для комунікації між ними. Всі дані, що передаються між компонентами, мають бути зашифровані для забезпечення захищеності зв’язку [14].

Серверна частина відповідатиме за обробку та відповідь на запити від вебклієнта, міститиме бізнес-логіку та запити до бази даних. Для підвищення швидкодії системи, перевірка коректності введених даних користувачем повинна виконуватися як на стороні вебклієнта, так і на сервері [15].

Проєкт має бути розміщений у хмарному сховищі для забезпечення безперервного та безпечного доступу до даних і для забезпечення надійності зберігання. Для бази даних слід використовувати NoSQL, що забезпечить масштабованість та гнучкість реалізації. Доступ до бази даних повинен здійснюватися лише через запити з серверної частини програмного модуля [6].

2.2 Функціональні вимоги інформаційної системи

Програмний модуль має містити такі функції для відвідувачів інтернет-магазину:

– авторизація;

– реєстрація;
– перегляд списку товарів;
– замовлення продукції на друк;
– зміна вмісту кошика.

Для покупців інформаційної системи:

– авторизація;
– реєстрація;
– перегляд профілю;
– перегляд історії замовлень;
– замовлення продукції на друк;
– оформлення замовлення на покупку;
– оплата замовленого товару;
– перегляд списку товарів.

Для адміністраторів інформаційної системи:

– авторизація;
– перегляд списку товарів у системі;
– перегляд списку замовлень;
– додавання товарів у систему;
– видалення товарів з системи.

2.3 Нефункціональні вимоги інформаційної системи

Розроблюваний програмний модуль повинен відповідати деяким вимогам.

Безпека:

– паролі всіх користувачів мають зберігатися у зашифрованому вигляді;
– база даних має бути ізольована від будь-яких запитів, окрім запитів із серверної частини.

Супроводжуваність:

– реалізація програмного модуля повинна забезпечувати легке масштабування системи без зміни існуючих компонентів.

Доступність:

– система повинна коректно працювати у популярних браузерах Safari, Chrome, Firefox.

Вимоги до користувацького інтерфейсу інформаційної системи:

– інтерфейс повинен бути виконаний у єдиному стилі та повторно використовувати компоненти;
– інтерфейс повинен бути реалізований в темних тонах для забезпечення комфортного перегляду вебсторінок.

У програмній частині модуля мають бути використані такі технології:

– база даних – MongoDB;
– серверна частина – Node.js Express [17];
– вебчастина – React.js.

Всі частини системи повинні взаємодіяти через API інтерфейс.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Схема проєктування програмного забезпечення

Система має три групи користувачів: відвідувач магазину, покупець та адміністратор. На рисунку 3.1 представлено схему, яка описує функціональність та поведінку цих користувачів.

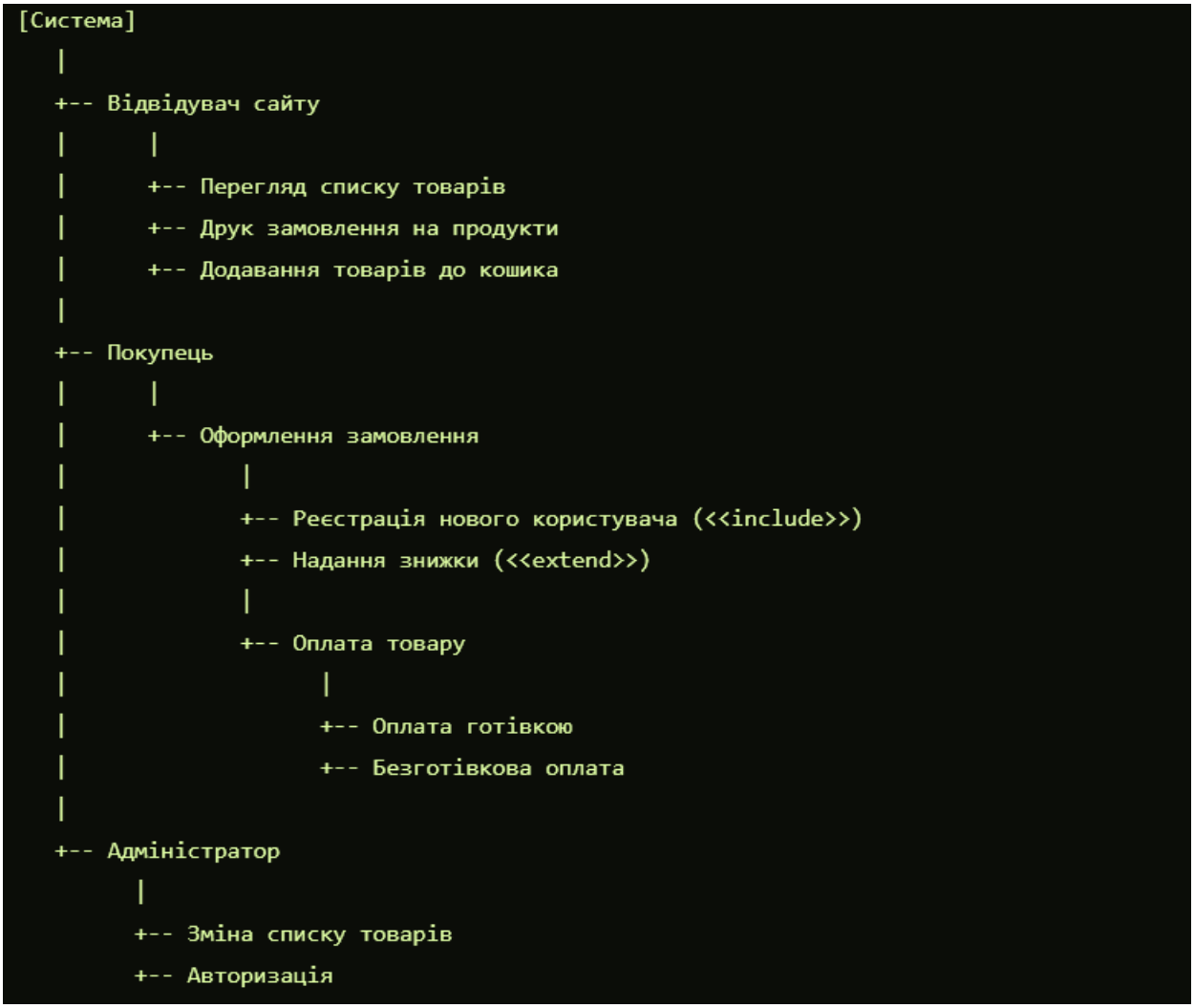

Рисунок 3.1 – Схема функціональності та поведінки користувачів

До функціональних можливостей відвідувача інтернет-магазину входять наступні можливості:

– перегляд каталогу товарів;
– замовлення товару;
– додавання товарів до кошику;
– видалення товарів з кошику.

До функціональних можливостей покупця входять наступні можливості:

– авторизація на сайті;
– реєстрація на сайті;
– доступ до історії замовлень;
– замовлення товару;
– оплата створеного замовлення.

До функціональних можливостей адміністратора входять наступні можливості:

– авторизація на сайті;
– видалення товару з системи;
– додавання товару до системи;
– представлення знижки на товар.

3.2 Проєктування архітектури програмного забезпечення

Складові системи включають такі елементи: вебдодаток, серверну частину та хмарну базу даних.

Для побудови архітектури вебдодатку було обрано фреймворк React [20] як основну технологію, а також ряд технологій для створення інтерфейсів і запитів для взаємодії з сервером [4].

Архітектура серверної частини базується на NoSQL базі даних MongoDB Atlas [6] та сервері, реалізованому на Node.js Express [18].

Сервер на Node.js [2] характеризується високою продуктивністю і легко масштабується. Зв’язок між сервером та базою даних забезпечується за допомогою бібліотеки Mongoose [19].

Проєкт представлений у вигляді діаграм розгортання. Схему розгортання проєкту можна побачити на рисунку 3.2.

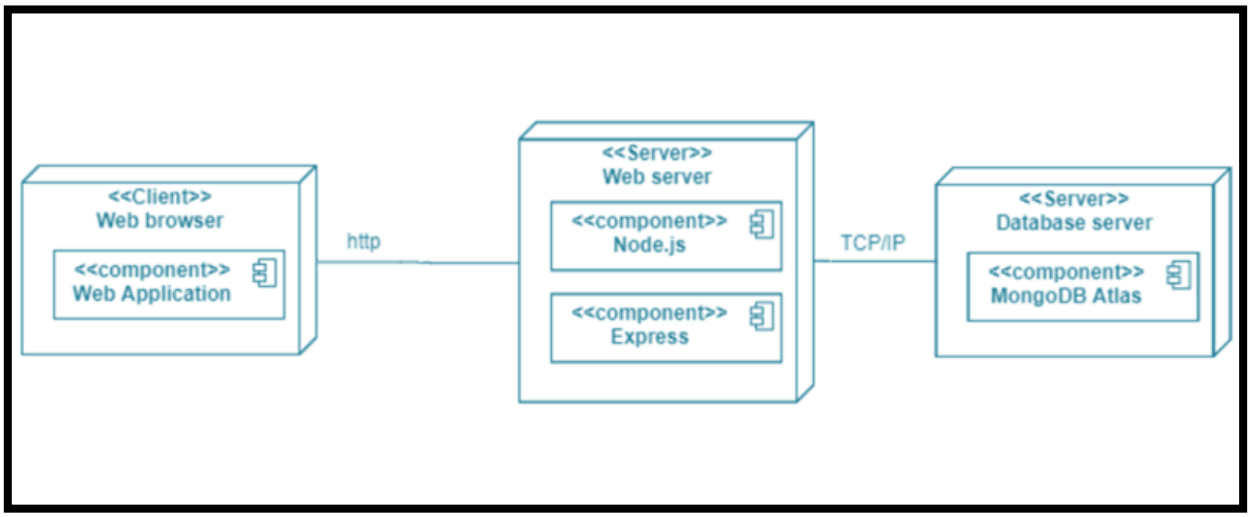

Рисунок 3.2 – Діаграма розгортання

На діаграмі показано взаємозв'язок компонентів програмної системи. Всі частини взаємодіють через TCP/IP канали за допомогою API. Дані обробляються на сервері відповідно до моделей даних, зберігаються в базі даних або передаються до клієнтської частини системи.

3.3 Проєктування структури зберігання даних

Базу даних у системі виконує MongoDB, яка є документо-орієнтованою базою даних. Дані зберігаються у вигляді документів, які мають структуру, схожу на файли JSON. Цей тип баз даних не потребує нормалізації, що дозволяє легко масштабувати базу та підтримувати цілісність даних без порушення роботи системи при зміні структури бази даних [3].

Середовище MongoDB Atlas забезпечує безперервний доступ до бази даних і захист від несанкціонованого доступу. Для цього необхідно точно визначити потенційних користувачів системи та дозволені IP-адреси для входу в систему. Структуру бази даних наведено на рисунку 3.3.

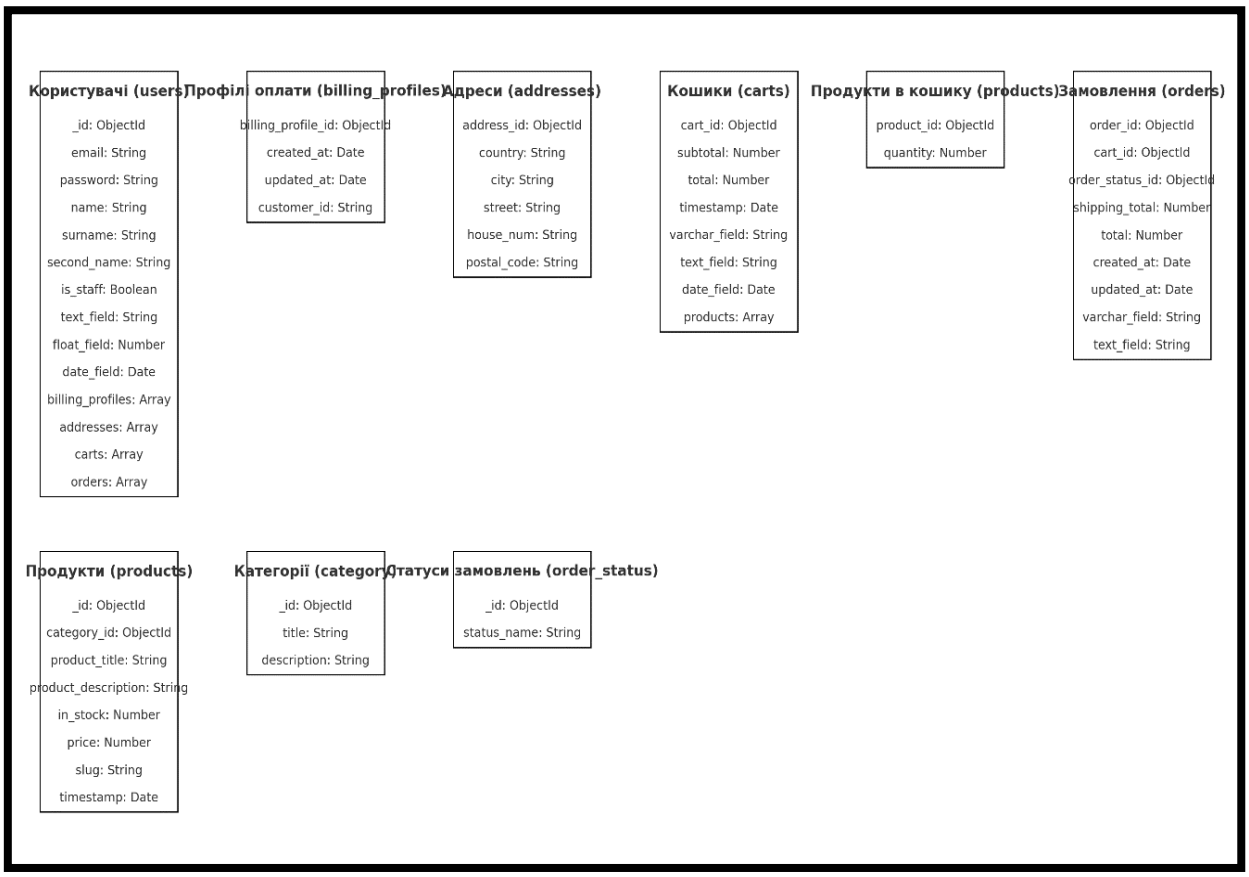

Рисунок 3.3 – Структура бази даних

На представленій діаграмі показано структуру та співвідношення елементів бази даних системи. Вона містить 9 моделей: користувачі, кошик, замовлення, товари, адреса доставки, категорії товарів, статуси замовлень, платіжні дані та товари в кошику.

Усі компоненти вебчастини повинні лише отримувати запити до бази даних через серверну частину системи, що дозволяє їм отримувати ці дані за допомогою API-сервера та додавати рівень абстракції.

3.4 Створення UI / UX дизайну системи

При створенні інтерфейсу користувача необхідно враховувати користувацький досвід та дотримуватися основних вимог до проєктування інтерфейсів.

Зокрема, вимоги до проєктування включають:

– структурованість усіх елементів інтерфейсу;
– логічне групування всіх елементів інтерфейсу;
– вирівнювання всіх елементів інтерфейсу;
– виконання всіх елементів інтерфейсу та сторінок в єдиному стилі.

Для оформлення програмної системи було вирішено використовувати кольори сірого та чорного відтінків з додаванням білих елементів для контрасту.

На рисунку 3.4 наведено скриншот головної сторінки сайту.

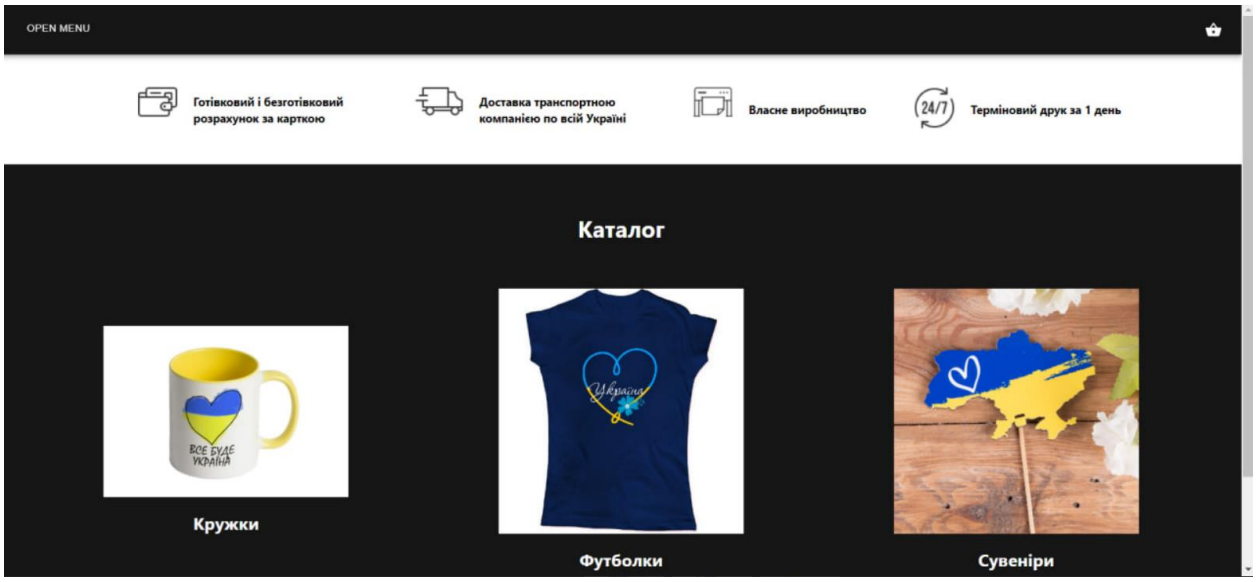

Рисунок 3.4 – Головна сторінка сайту

На головній сторінці сайту розміщені всі основні товари та інформація, яка може знадобитися користувачеві, а також коротка інформація про діяльність компанії та товари, які компанія пропонує для продажу.

Основним елементом навігації по магазину є навігаційна панель у верхній частині будь-якої сторінки. Ця панель містить усі основні посилання: сторінки товарів, поділених за групами, кошик, посилання на головну сторінку та посилання на профіль користувача, де користувач може авторизуватися або зареєструватися за потреби.

На рисунку 3.5 наведено скриншот сторінки товарів групи «Кружки».

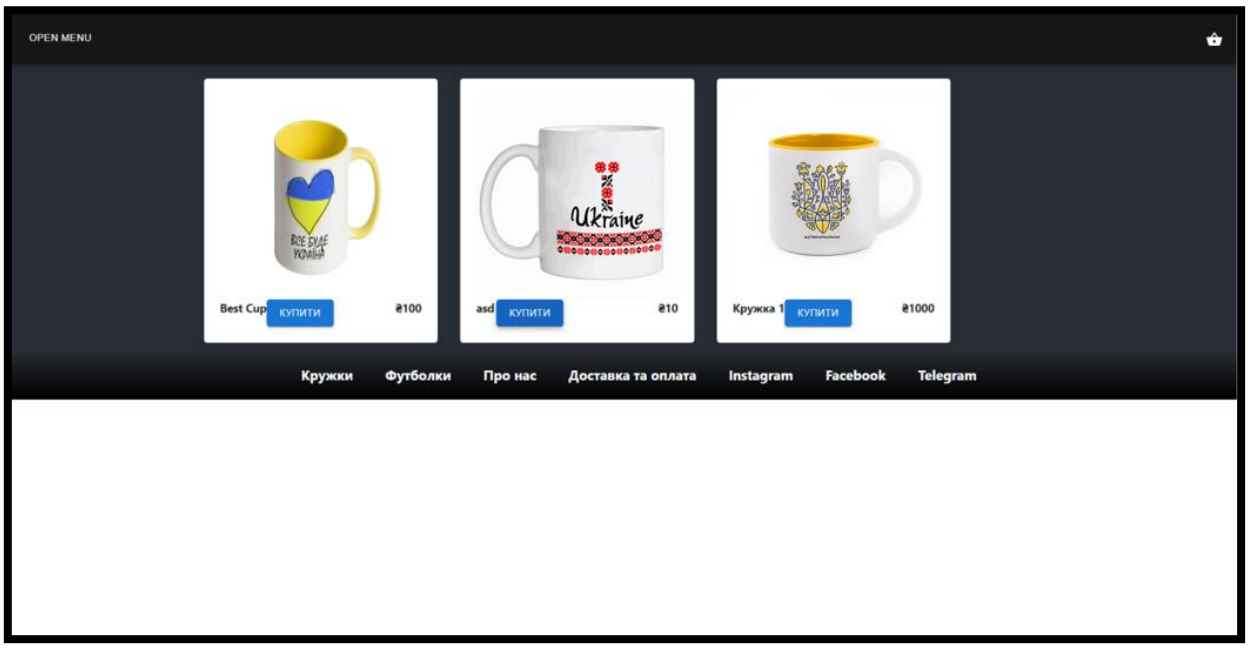

Рисунок 3.5 – Сторінка товарів групи «Кружки»

На сторінці груп товарів, їх список відображається у вигляді карток, які містять інформацію про товар, його ціну та назву. На рисунку 3.6 наведено скриншот сторінки кошика сайту.

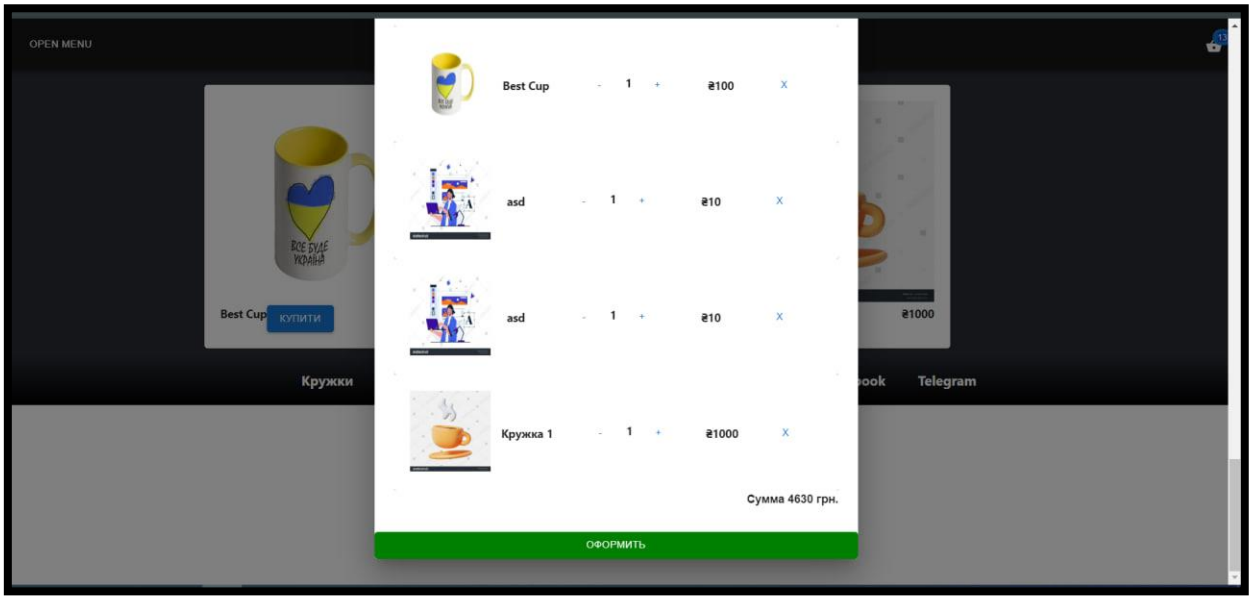

Рисунок 3.6 – Сторінка кошику

На сторінці кошика представлена інформація про всі товари, які обрав користувач, з можливістю редагування кількості або видалення товарів з кошика. На рисунку 3.7 наведено скриншот сторінки списку товарів.

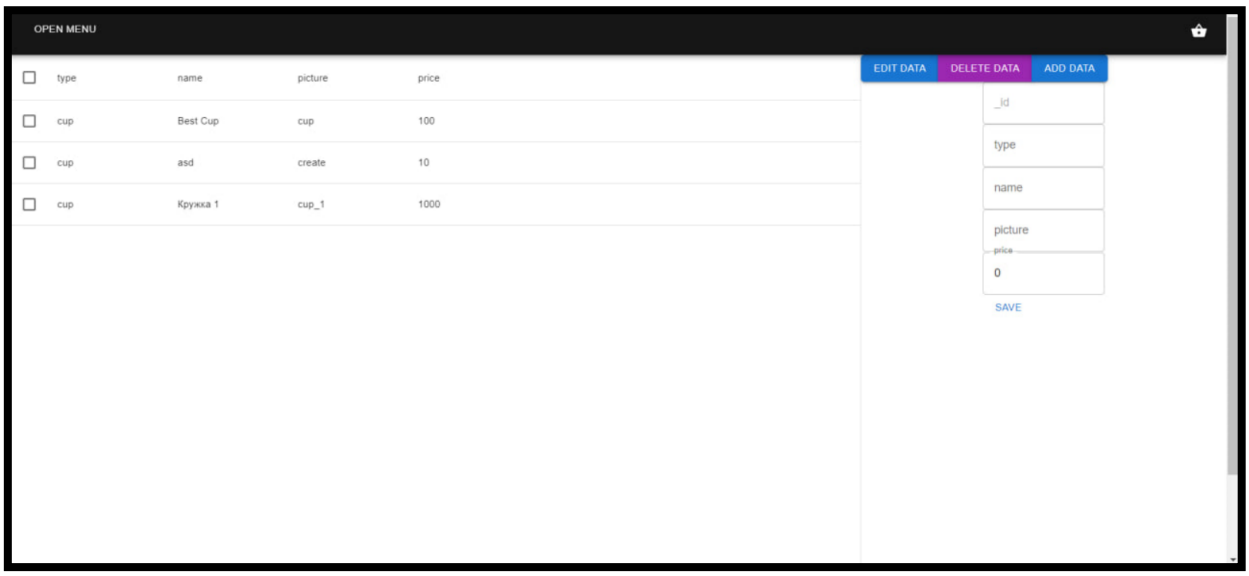

Рисунок 3.7 – Сторінка списку товарів

На сторінці списку товарів відображається інформація про всі наявні товари для продажу. Доступ до цієї сторінки має виключно адміністратор. З цієї сторінки адміністратор може додавати або видаляти товари, обираючи їх з таблиці. Також адміністратор має доступ до сторінки з усіма замовленнями. Це можна побачити на рисунку 3.8.

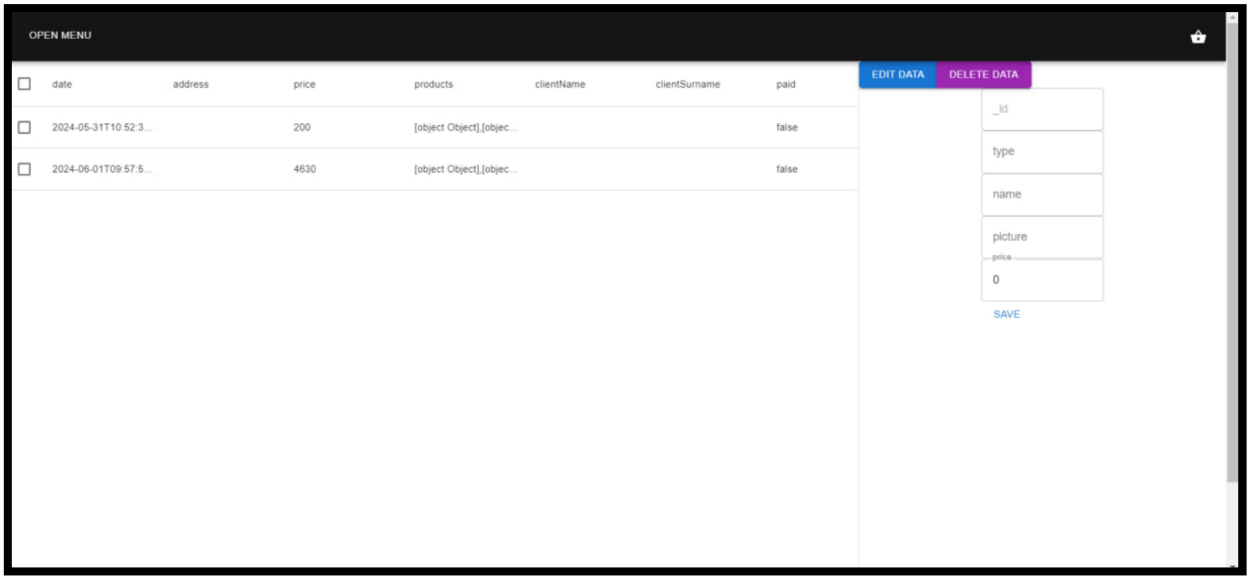

Рисунок 3.8 – Сторінка списку замовлень

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Реалізація серверної частини програмного забезпечення

Серверну частину було створено на базі платформи Node.js, а для створення серверної частини обрано фреймворк Express.js. На основі Express було розроблено API інтерфейс програми [7]. Архітектурна модель має дворівневу структуру. Структуру проєкту серверної частини можна побачити на рисунку 4.1.

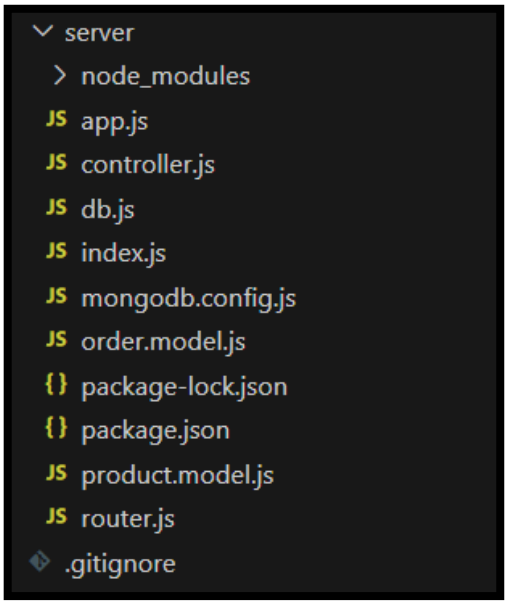

Рисунок 4.1 – Структура проєкту серверної частини

Запуск сервера здійснюється за допомогою файлу **app.js**, який містить код для запуску сервера. Підключення до бази даних відбувається за допомогою файлу **mongodb.config.js**, який містить налаштування підключення до бази даних та всі файлові залежності проєкту. Файли **order.model.json** та **product.model.json** містять моделі об’єктів системи. Ці моделі використовуються як схеми для MongoDB та містять інформацію про поля документа, посилання на інші таблиці та значення за замовчуванням (якщо необхідно).

На рисунку 4.2 наведено приклад коду реалізації моделі для замовлення.

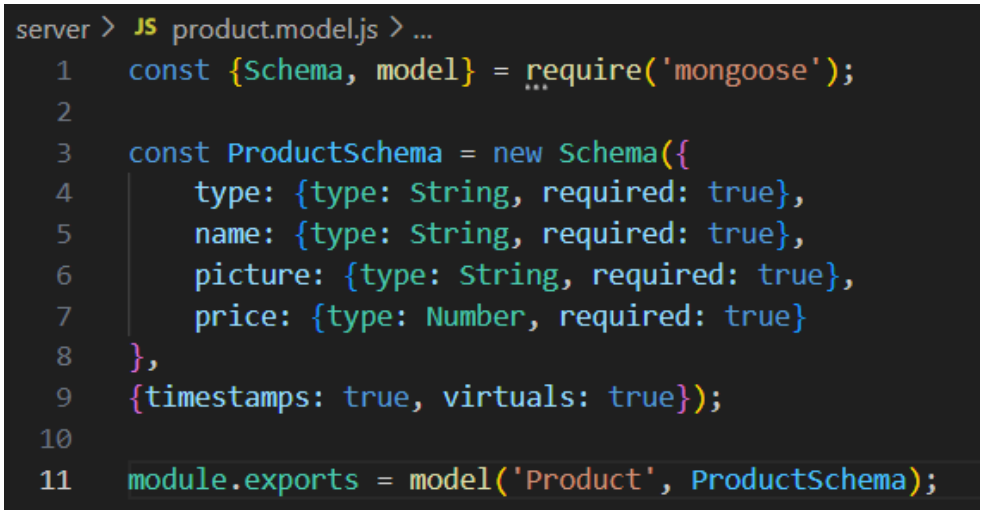

Рисунок 4.2 – Реалізація моделі для замовлення

Цей код створює модель продукту для MongoDB за допомогою Mongoose:

– імпортує об’єкти Schema та model з Mongoose [19];
– створює схему ProductSchema з полями type, name, picture та price, всі вони обов’язкові;
– додає автоматичні поля createdAt та updatedAt;
– експортує модель Product на основі цієї схеми.

Ця модель використовується для взаємодії з колекцією продуктів у MongoDB.

У файлі **controller.js** міститься обробка кожної моделі. На рисунку 4.3 наведено приклад коду реалізації контролера для замовлення:

Функція createOrder виконує наступне:

– створює нове замовлення з даних, отриманих з тіла запиту (адреса, ціна, продукти, ім’я клієнта, прізвище, статус оплати, статус відправки, email, місто);
– зберігає це замовлення в базі даних;
– якщо збереження успішне, повертає дані замовлення зі статусом 200;
– у разі помилки повертає повідомлення про помилку зі статусом 500.

```
126  createOrder = async (req, res) => {
127    const order = new Order({
128      date: new Date(),
129      address: req.body.address,
130      price: req.body.price,
131      products: req.body.products,
132      clientName: req.body.name,
133      clientSurname: req.body.surname,
134      paid: req.body.paid,
135      sent: req.body.sent,
136      email: req.body.email,
137      city: req.body.city,
138    });
139
140    order
141      .save()
142      .then((data) => {
143        res.status(200).json(data);
144      })
145      .catch((err) => {
146        res.status(500).json({
147          message: "Fail!",
148          error: err.message,
149        });
150      });
151  };
```

Рисунок 4.3 – Програмний код контролеру «OrderController»

У файлі **router.js** зберігаються всі маршрути для серверної частини проєкту.

На рисунку 4.4 наведено приклад коду реалізації роутера.

```
server > JS router.js > ...
1   const express = require('express')
2
3   const router = express.Router();
4
5   let product = require('./controller');
6
7   router.post('/api/product', product.createProduct);
8   router.get('/api/product/:id', product.getProduct);
9   router.get('/api/products', product.products);
10  router.put('/api/product', product.updateProduct);
11  router.delete('/api/product/:id', product.deleteProduct);
12
13  router.post('/api/order', product.createOrder);
14  router.get('/api/order/:id', product.getOrder);
15  router.get('/api/orders', product.orders);
16  router.put('/api/order', product.updateOrder);
17  router.delete('/api/order/:id', product.deleteOrder);
18
19  module.exports = router;
```

Рисунок 4.4 – Програмний код роутеру «router»

Цей файл налаштовує маршрутизацію для продуктів та замовлень у Express.js додатку:

– імпортує Express та створює новий роутер;
– імпортує контролери з файлу ./controller;
– визначає маршрути для CRUD операцій з продуктами та замовленнями;
– експортує роутер для використання в додатку.

На рисунку 4.5 наведено частину коду для реалізації додавання продукту.

Функція createProduct додає новий продукт у базу даних. Ось як вона працює:

– створює новий продукт з даних, отриманих з тіла запиту (req.body);
– зберігає цей продукт у базі даних;
– якщо збереження успішне, повертає дані продукту зі статусом 200.

```
const Product = require("./product.model");
const Order = require("./order.model");
// add new product
createProduct = async (req, res) => {
  const product = new Product({
    type: req.body.type,
    name: req.body.name,
    picture: req.body.picture,
    price: req.body.price,
  });

  product
    .save()
    .then((data) => {
      res.status(200).json(data);
    })
    .catch((err) => {
      res.status(500).json({
        message: "Fail!",
        error: err.message,
      });
    });
};
```

Рисунок 4.5 – Код реалізації додавання продукту

Функція createProduct додає новий продукт у базу даних. Вона:

– створює новий об’єкт продукту з даних запиту;
– зберігає продукт у базі даних;

– повертає успішну відповідь зі статусом 200 і даними продукту або помилку зі статусом 500 у разі невдачі.

4.2 Реалізація вебчастини програмного забезпечення

Клієнтська частина була реалізована на основі фреймворку React.js, що працює на платформі Node.js.

Використання цього фреймворку значно спрощує роботу з DOM-елементами сторінки та сприяє підвищенню продуктивності за рахунок використання простих шаблонів для відображення даних у мережі. На рисунку 4.6 представлено структуру проєкту клієнтської частини.

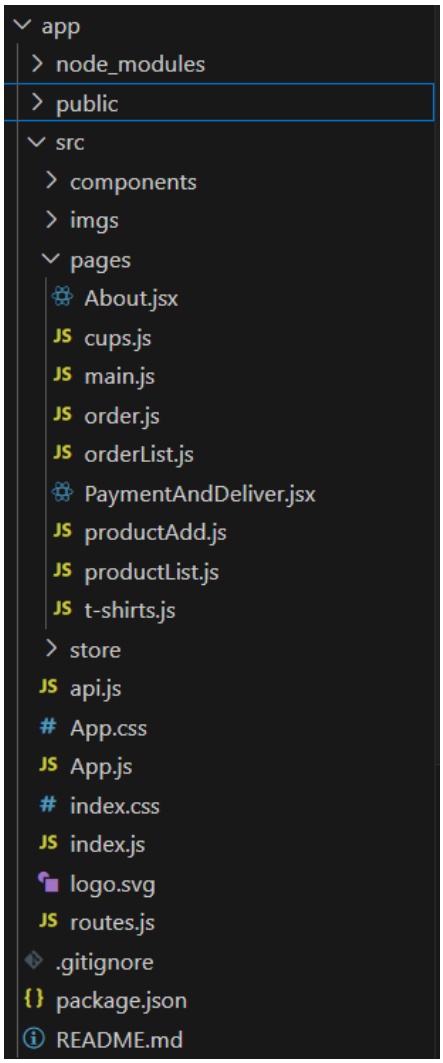

Рисунок 4.6 – Структура проєкту клієнтської частини

У файлі **App.js** міститься інформація про початкову сторінку та відомості про налаштування проєкту. Інформація про залежності проєкту міститься у файлі **index.js**, який є основним для запуску проєктів у майбутньому. Усі маршрути клієнтської частини включені у файл **routes.js**. На рисунку 4.7 наведено приклад коду реалізації роутера вебпрограми.

app > src > JS routes.js > ...

```
1   import React from "react";
2   import { BrowserRouter, Routes, Route } from "react-router-dom";
3   import { MainPage } from "./pages/main";
4   import { ProductList } from "./pages/productList";
5   import { ProductAdd } from "./pages/productAdd";
6   import { Cups } from "./pages/cups";
7   import { Shirts } from "./pages/t-shirts";
8   import { Order } from "./pages/order";
9   import { OrderList } from "./pages/orderList";
10  import { About } from "./pages/About";
11  import { PaymentAndDeliver } from "./pages/PaymentAndDeliver";
12
13  export const useRoutes = () => {
14    return (
15      // <BrowserRouter>
16        <Routes>
17          <Route path="/" exact={true} component={MainPage} />
18          <Route path="/cups" component={Cups} />
19          <Route path="/t-shirts" component={Shirts} />
20          <Route path="/products" component={ProductList} />
21          <Route path="/product/new" component={ProductAdd} />
22          <Route path="/order" component={Order} />
23          <Route path="/orders" component={OrderList} />
24          <Route path="/about" component={About} />
25          <Route path="/ship&pay" component={PaymentAndDeliver} />
26        </Routes>
27      // </BrowserRouter>
28    );
29  };
30
```

Рисунок 4.7 – Програмний код контролеру «userController»

Ця функція useRoutes налаштовує маршрутизацію в React-додатку за допомогою react-router-dom, визначаючи маршрути для різних сторінок, таких як головна сторінка, список продуктів, додавання продукту, список замовлень, сторінка про нас та сторінка оплати і доставки.

На всіх сторінках інтерфейсу є окремі компоненти для повторного використання, де це необхідно. Для полегшення підтримки проєкту, сторінки та компоненти були перенесені в різні каталоги. Необхідні компоненти можна знайти в каталозі **components**. У цьому каталозі знаходяться всі компоненти, що використовуються в проєкті, їх назви відповідають завданням і типам користувачів. Наприклад, у каталозі **navbar** містяться тільки компоненти навігаційної панелі.

Кожна сторінка містить шаблон сторінки і необхідні компоненти. На рисунку 4.8 наведено приклад програмного рішення для головної сторінки сайту.

app > src > pages > JS productList.js > ...

```
1   import React, { useState, useEffect } from 'react';
2   import { DataGrid } from '@mui/x-data-grid';
3   import { Button, ButtonGroup, Container, TextField } from '@mui/material';
4   import api from '../api'
5
6   let rows = [
7   ]
8   const columns = [
9       { field: 'type', headerName: 'type', width: 150 },
10      { field: 'name', headerName: 'name', width: 150 },
11      { field: 'picture', headerName: 'picture', width: 150 },
12      { field: 'price', headerName: 'price', width: 150 },
13  ];
14
```

Рисунок 4.8 – Програмний код контролеру «userController»

Цей код імпортує необхідні модулі та компоненти для створення таблиці продуктів у React-додатку:

– імпортує React, хуки useState та useEffect з бібліотеки 'react;
– імпортує компонент DataGrid з бібліотеки @mui/x-data-grid для відображення таблиці даних;
– імпортує компоненти Button, ButtonGroup, Container та TextField з бібліотеки @mui/material для створення інтерфейсу користувача;
– імпортує api з локального файлу для взаємодії з API.

Для роботи з даними використовуються різні хуки життєвого циклу компонента, що дозволяють обробляти всі етапи його існування.

В результаті роботи було створено наступні сторінки інтерфейсу користувача:

– сторінка авторизації;
– сторінка реєстрації;
– головна сторінка сайту;
– сторінка перегляду профілю користувача;
– сторінка перегляду списку товарів по групах;
– сторінка перегляду списку товарів (для адміністратора);
– сторінка створення замовлення;
– сторінка перегляду кошика;
– сторінка оплати;
– сторінка перегляду інформації про компанію;
– сторінка перегляду договору користування сайтом;
– сторінка перегляду замовлення;
– сторінка перегляду списку замовлень (для адміністратора).

На рисунку 4.9 наведено приклад розробленої сторінки списку товарів групи «Кружки».

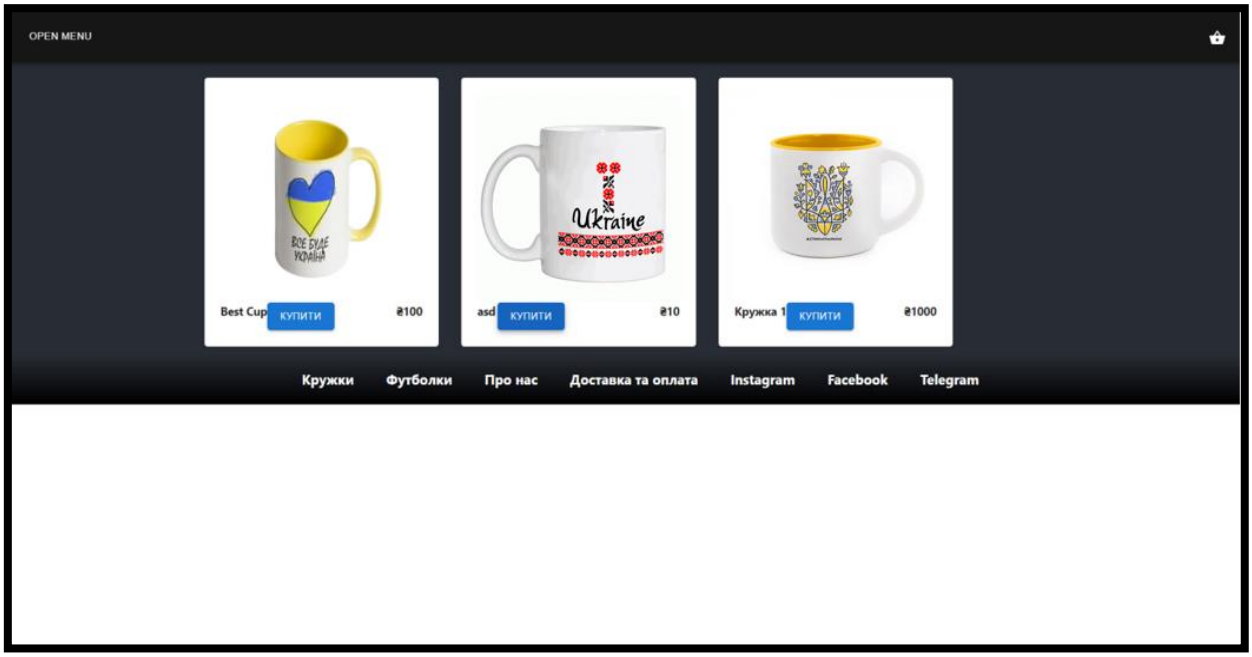

Рисунок 4.9 – Сторінка перегляду списку товарів по групам

Щоб зайти на сторінку списку товарів, користувач повинен перейти з головної сторінки за посиланням на обрану групу товарів або скористатися посиланням на обрану групу товарів з навігаційної панелі. На цій сторінці відображаються дані про основні товари обраної групи, такі як назва, ціна та зображення. Якщо користувач бажає придбати товар, він має натиснути кнопку «Buy», після чого цей товар з’явиться у кошику користувача, де він може продовжити оформлення замовлення або видалити товар, якщо передумає. На рисунку 4.10 наведено скриншот сторінки кошика.

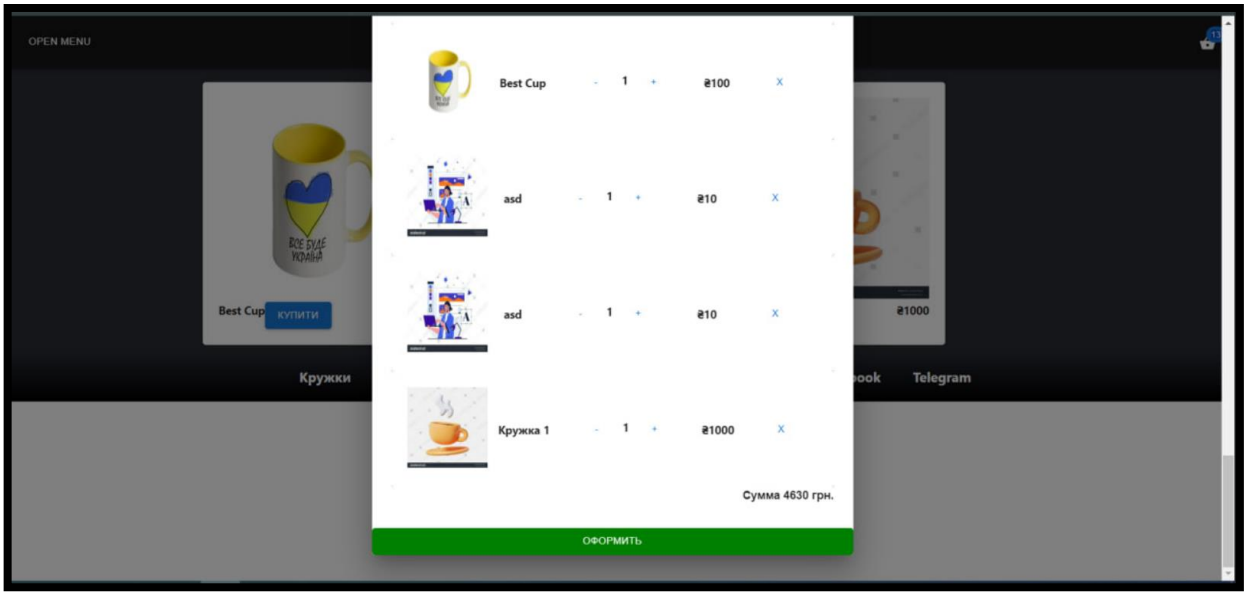

Рисунок 4.10 – Сторінка кошику користувача

На сторінці кошика відображається список товарів з такими даними, як зображення, назва, кількість, ціна з урахуванням кількості товарів, а також кнопка «Оформити», яка створює запит на формування нового замовлення та перенаправляє користувача до сторінки введення даних. Є можливість редагування кількості товару або видалення товару.

На рисунку 4.11 наведено скриншот сторінки оформлення замовлення. На сторінці оформлення замовлення користувач заповнює дані про своє ім’я, прізвище, номер телефону, місто та електронну пошту. Після натиснення кнопки «Оформити», ці дані передаються до бази даних, і в майбутньому з клієнтом зв’яжеться менеджер компанії для відправлення товару.

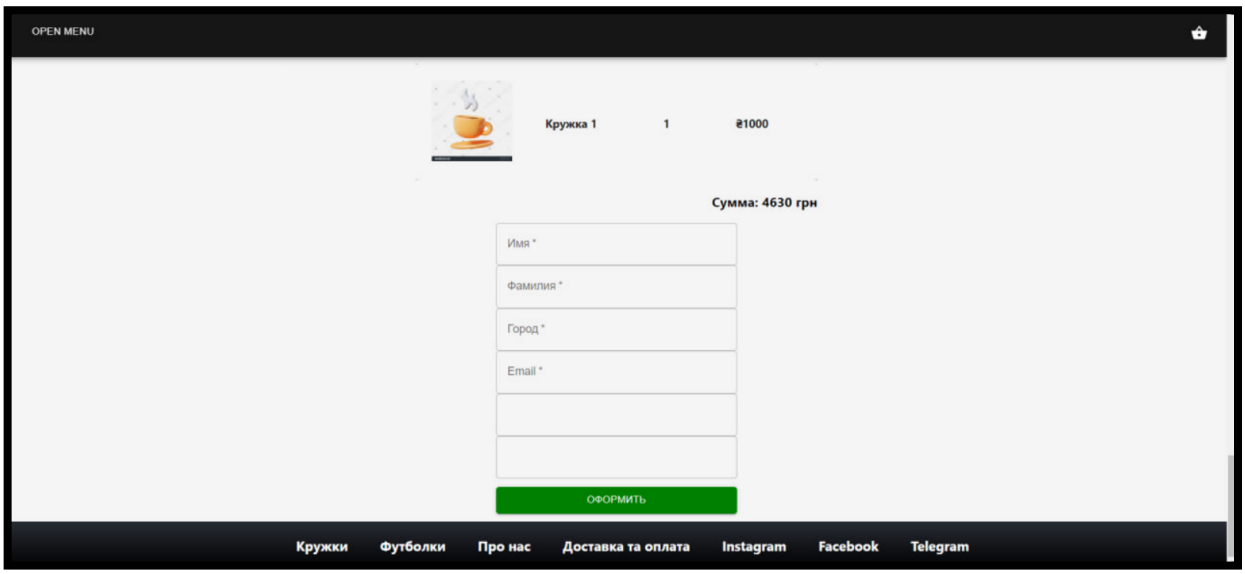

Рисунок 4.11 – Сторінка внесення даних для оформлення замовлення

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для тестування було обрано функціональне тестування та тестування API. Розроблено тест-кейс для перевірки працездатності модулів та їхньої функціональності. У таблиці 5.1 наведено приклад, який використовується для тестування функціональних можливостей.

Таблиця 5.1 – Приклад тест-кейсу

Кроки тесту	**Опис**	**Статус**
Натиснути кнопку додавання товару у кошик	Кнопка ініціює відправку даних до серверу	Пройдено
Отримати сповіщення біля посилання на кошик у навігаційній панелі	Відображення сповіщення біля надпису «Cart»	Пройдено
Зайти у кошик та переконатися у наявності товару у кошику	Відображення вибраного товару у кошику та його картинка, назва, кількість та ціна	Пройдено

При проведенні функціонального тестування всі тести були виконані та набули статусу «Пройдено».Програмний модуль виконує всі необхідні функції без будь-яких збоїв.

Тестування програмного модуля API проводилося для перевірки працездатності повідомлення системою про свій стан та справної роботи всіх маршрутів серверної частини програми. Для цього використовувався інструмент Postman. Для інформування інших частин програмної системи використовуються наступні статус коди:

– 200: операція успішна;
– 201: авторизація успішна;
– 401: не авторизовано;
– 404: результат не знайдено;
– 409: email вже використовується;
– 500: будь-яка помилка серверу.

Приклад результатів тестування API наведено у таблиці 5.2.

Таблиця 5.2 – Результати тестування API

Метод	**URL**	**Дані**	**Тип**	**Статус код**	**Дані відповіді**
GET	/api/product/	–	–	200	Array of objects
GET	/api/product/	–	–	500	message
POST	/api/product	Product object	valid	201	Array of objects
POST	/api/product	Product object	invalid	409	message
POST	/api/product	Product object	Valid/ invalid	500	message
PUT	/api/product	Product object + _id	valid	204	Array of objects
PUT	/api/product	Product object + _id	invalid	409	message
PUT	/api/product	Product object + _id	Valid/ invalid	500	Message
DELETE	/api/product/	_id	valid	204	Array of objects
DELETE	/api/product/	_id	invalid	409	message
DELETE	/api/product/	_id	Valid/ invalid	500	message

Продовження таблиці 5.2

Метод	**URL**	**Дані**	**Тип**	**Статус код**	**Дані відповіді**
POST	/api/order	Order object	valid	201	Array of objects
POST	/api/order	Order object	invalid	409	message
POST	/api/order	Order object	Valid/ invalid	500	message
PUT	/api/order	Order object + _id	valid	204	Array of objects
PUT	/api/order	Order object + _id	invalid	409	message
PUT	/api/order	Order object + _id	Valid/ invalid	500	message
DELETE	/api/order/	{_id}	valid	204	Array of objects
DELETE	/api/order/	{_id}	invalid	409	message
DELETE	/api/order/	{_id}	Valid/ invalid	500	message

За підсумками проведення тестування API, було підтверджено роботу всіх маршрутів та отримано відповідні статус коди для надсилання некоректних та валідних даних, а також отримання відповідних повідомлень або помилок.

ВИСНОВКИ

З кожним роком зростає кількість людей, які здійснюють покупки через інтернет, тому адаптація бізнесу до новітніх технологій стає все більш актуальною. Основними факторами, що впливають на користувацький досвід, є доступність та швидкість обробки й доставки замовлень. Для вирішення цієї проблеми необхідно було провести аналіз предметної галузі з метою визначення існуючих проблем та можливих шляхів їх вирішення [16].

Під час виконання кваліфікаційної роботи було проведено глибокий аналіз предметної галузі, що дозволило виявити ключові проблеми та фактори, які сприяють зростанню інтернет-продажів у конкретному бізнесі. Аналіз конкурентів дозволив визначити їх функціональні можливості, а також переваги та недоліки. Це допомогло сформувати вимоги до майбутньої системи, враховуючи сильні та слабкі сторони конкурентів.

Було визначено основні функції системи відповідно до поставленого завдання та її опису. Під час розробки програмного забезпечення були сформульовані загальні принципи роботи системи та описані вимоги до майбутньої системи.

У процесі створення програми були визначені основні компоненти та описана їх функціональність і поведінка. Також була спроєктована структура зберігання даних та інтерфейс користувача.

В результаті виконання кваліфікаційної роботи було визначено, описано та реалізовано основні вимоги до системи, а також розроблено програмний модуль для продажу сувенірної продукції.

ПЕРЕЛІК ПОСИЛАНЬ

1. Державна служба статистики України. URL: http://www.ukrstat.gov.ua/ (дата звернення: 26.01.2024).
2. Node.js v22.3.0 documentation. URL: https://nodejs.org/uk/docs/ (дата звернення: 20.02.2024).
3. Вебдизайн. URL: https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1%D0%B4%D0%B8%D0%B7%D0%B0%D0%B9%D0%BD (дата звернення: 26.02.2024).
4. Що таке вебсервер та як його вибрати. URL: https://sendpulse.ua/blog/what-is-a-web-server (дата звернення: 26.02.2024).
5. Commerce Trends 2023. URL: https://www.shopify.com/plus/commerce-trends (дата звернення: 27.02.2024).
6. What is NoSQL? URL: https://www.mongodb.com/nosql-explained (дата звернення: 20.04.2024).
7. What is ECOM Transaction? How It Works and Benefits. URL: https://razorpay.com/learn/what-is-ecom-transaction/ (дата звернення: 20.04.2024).
8. E-commerce Defined: Types, History, and Examples. URL: https://www.investopedia.com/terms/e/ecommerce.asp (дата звернення: 21.04.2024).
9. How does site speed impact conversion rates & sales. URL: https://acclaim.agency/blog/how-does-site-speed-impact-conversion-rates-sales (дата звернення: 22.04.2024).
10. Why does speed matter? URL: https://developers.google.com/web/fundamentals/performance/why-performance-matters (дата звернення: 24.04.2024).
11. Оптимізація швидкості роботи вебдодатків. URL: https://medium.com/@augustusphyras/web-performance-core-web-vitals-inp-1acfaba9700a (дата звернення: 25.04.2024).

12. Web Performance: Core Web Vitals & INP. URL: https://www.statista.com/topics/871/online-shopping/ (дата звернення: 25.04.2024).
13. eCommerce Technology: Top 10 Trends Shaping 2024’s Retail Landscape. URL: https://www.digitalsilk.com/digital-trends/ecommerce-technology/ (дата звернення: 25.04.2024).
14. OWASP Top Ten – URL: https://owasp.org/www-project-top-ten/ (дата звернення: 01.05.2024).
15. Production best practices: performance and reliability. URL: https://expressjs.com/en/advanced/best-practice-performance.html (дата звернення: 02.05.2024).
16. The Future of E-Commerce: Trends Shaping Global Trade. URL: https://www.linkedin.com/pulse/future-e-commerce-trends-shaping-global-trade-dan-wallace-brewster-qh5rc/ (дата звернення: 03.05.2024).
17. Support ExpressLRS. URL: https://github.com/ExpressLRS/ExpressLRS (дата звернення: 20.04.2024).
18. Офіційна документація Express. URL: https://github.com/expressjs/express (дата звернення: 20.04.2024).
19. Офіційна документація Mongoose. URL: https://github.com/Automattic/mongoose (дата звернення: 20.04.2024).
20. Офіційна документація React. URL https://uk.legacy.reactjs.org/docs/getting-started.html (дата звернення: 20.04.2024).

ДОДАТОК А

Реалізація окремих функцій серверної частини

Розглянемо реалізацію певного функціоналу серверної частини програмного забезпечення.

Наприклад, передбачено можливість видалення продуктів за їх **id**. Частину коду для реалізації цього функціоналу можна побачити на рисунку А.1.

```
// delete product by id
deleteOrder = async (req, res) => {
  let orderId = req.params.id;

  Product.findByIdAndRemove(orderId)
    .select("-__v -_id")
    .then((order) => {
      if (!order) {
        res.status(404).json({
          message: "Does Not exist a Order with id = " + orderId,
          error: "404",
        });
      }
      res.status(200).json({});
    })
    .catch((err) => {
      return res.status(500).send({
        message: "Error -> Can NOT delete a Order with id = " + orderId,
        error: err.message,
      });
    });
};
```

Рисунок А.1 – Код реалізації видалення продукту за «id»

Функція deleteOrder призначена для видалення замовлення з бази даних за його ID. Вона приймає два параметри: req (запит) та res (відповідь). Опишемо, що вона робить крок за кроком.

Перший. Отримує id замовлення з параметрів запиту (req.params.id) і зберігає його в змінній orderid.

Другий. Викликає метод findbyidandremove з моделі product, передаючи в нього orderid, щоб знайти та видалити замовлення з бази даних.

Третій. Використовує метод .select("-__v -_id") для виключення полів версії (__v) та ідентифікатора (_id) з результату.

Четвертий. Виконує .then для обробки знайденого і видаленого замовлення:

– якщо замовлення не знайдено, повертає статус 404 з повідомленням про помилку, що замовлення не існує;
– якщо замовлення знайдено і видалено, повертає пустий об’єкт зі статусом 200.

П’ятий. Виконує .catch для обробки можливих помилок під час видалення замовлення, повертаючи статус 500 з повідомленням про помилку.

Таким чином, ця функція дозволяє видалити замовлення з бази даних за вказаним ID і повертає відповідний результат або повідомлення про помилку в разі невдачі.

Передбачено можливість оновлення продуктів. Частина коду для реалізації зображена на малюнку А.2.

```
// update product
updateProduct = async (req, res) => {
  Product.findByIdAndUpdate(
    req.body._id,
    {
      type: req.body.type,
      name: req.body.name,
      picture: req.body.picture.name,
      price: req.body.price,
    },
    { new: true }
  )
    .select("-__v")
    .then((product) => {
      if (!product) {
        return res.status(404).send({
          message:
            "Error -> Can NOT update a product with id = " + req.params.id,
          error: "Not Found!",
        });
      }
      res.status(200).json(product);
    })
    .catch((err) => {
      return res.status(500).send({
        message: "Error -> Can not update a product with id = " + req.params.id,
        error: err.message,
      });
    });
};
```

Рисунок А.2 – Код реалізації оновлення продукту

Ця функція updateProduct призначена для оновлення даних продукту в базі даних. Вона приймає запит (req) та відповідь (res) як аргументи і виконує наступні дії.

Перша. Викликає метод findbyidandupdate з моделі product, передаючи в нього id продукту, який потрібно оновити (req.body._id), нові значення полів (type, name, picture, price) з тіла запиту (req.body), а також опцію { new: true }, що повертає оновлений документ.

Друга. Використовує метод .select("_ -v") для виключення поля версії (__v) з результату.

Третя. Виконує .then для обробки знайденого та оновленого продукту:

– якщо продукт не знайдено, повертає статус 404 з повідомленням про помилку;

– якщо продукт знайдено, повертає його в форматі json зі статусом 200.

Четверта. Виконує .catch для обробки можливих помилок під час оновлення продукту, повертаючи статус 500 з повідомленням про помилку.

Таким чином, ця функція дозволяє оновити дані про продукт у базі даних і повертає відповідний результат або повідомлення про помилку в разі невдачі.

На рисунку А.3 наведено частину коду для реалізації створення замовлення.

```
createOrder = async (req, res) => {
  const order = new Order({
    date: new Date(),
    address: req.body.address,
    price: req.body.price,
    products: req.body.products,
    clientName: req.body.name,
    clientSurname: req.body.surname,
    paid: req.body.paid,
    sent: req.body.sent,
    email: req.body.email,
    city: req.body.city,
  });

  order
    .save()
    .then((data) => {
      res.status(200).json(data);
    })
    .catch((err) => {
      res.status(500).json({
        message: "Fail!",
        error: err.message,
      });
    });
};
```

Рисунок А.3 – Код реалізації створення замовлення

Функція createOrder призначена для створення нового замовлення в базі даних. Вона приймає два параметри: req (запит) та res (відповідь). Опишемо, що вона робить крок за кроком.

Перший. Створює новий об’єкт order, використовуючи дані з тіла запиту (req.body). вказані поля:

– date: встановлює поточну дату;
– address: адреса замовлення (req.body.address);
– price: ціна замовлення (req.body.price);
– products: список продуктів у замовленні (req.body.products);
– clientname: ім’я клієнта (req.body.name);
– clientsurname: прізвище клієнта (req.body.surname);
– paid: статус оплати замовлення (req.body.paid);
– sent: статус відправки замовлення (req.body.sent);
– email: електронна пошта клієнта (req.body.email);
– city: місто клієнта (req.body.city).

Другий. Зберігає новий об’єкт замовлення в базі даних, використовуючи метод .save().

Третій. Виконує .then для обробки успішного збереження замовлення: якщо замовлення успішно збережено, повертає дані замовлення у відповідь зі статусом 200.

Четвертий. Виконує .catch для обробки можливих помилок під час збереження замовлення: якщо виникає помилка, повертає статус 500 з повідомленням про невдачу та деталями помилки.

Таким чином, ця функція дозволяє створити нове замовлення, зберегти його в базі даних та повернути відповідний результат або повідомлення про помилку в разі невдачі.

ДОДАТОК Б

Реалізація окремих функцій вебчастини

Розглянемо реалізацію певного функціоналу вебчастини програмного забезпечення.

На рисунку Б.1 можна побачити реалізацію функціоналу для додавання нового елемента до кошика.

```
const handleBuy = (product) => {
  if (localStorage.getItem("podarok-print-en")) {
    let stored = JSON.parse(localStorage.getItem("podarok-print-en"));
    stored.cart.cartItems.push({
      _id: `${product._id}`,
      type: `${product.type}`,
      name: `${product.name}`,
      picture: `${product.picture}`,
      quantity: 1,
      price: `${product.price}`,
    });
    localStorage.setItem("podarok-print-en", JSON.stringify(stored));
    dispatch(addCartAction(1));
  } else {
    let string = {
      cart: {
        cartItems: [
          {
            _id: `${product._id}`,
            type: `${product.type}`,
            name: `${product.name}`,
            picture: `${product.picture}`,
            quantity: 1,
            price: product.price,
          },
        ],
        price: 0,
      },
    };
    localStorage.setItem("podarok-print-en", JSON.stringify(string));
    dispatch(addCartAction(1));
  }
};
```

Рисунок Б.1 – Код реалізації додавання нового елемента до кошика

Функція handleBuy призначена для додавання продукту в кошик. Вона виконує наступні дії.

Перша. Перевіряє, чи існує в локальному сховищі (localstorage) елемент з ключем "podarok-print-en".

Друга. Якщо елемент існує:

– отримує цей елемент з локального сховища та розпарсює його в об’єкт stored;
– додає новий продукт до масиву cartitems у об’єкті stored.cart;
– оновлює локальне сховище, зберігаючи змінений об’єкт stored назад у вигляді рядка json;
– викликає функцію dispatch(addcartaction(1)), яка, ймовірно, оновлює стан додатка, сигналізуючи про додавання товару в кошик.

Третя. Якщо елемент не існує:

– створює новий об’єкт string, що містить структуру кошика з масивом cartitems, який включає новий продукт;
– зберігає цей об’єкт у локальному сховищі під ключем "podarok-print-en" у вигляді рядка json;
– викликає функцію dispatch(addcartaction(1)), щоб оновити стан додатка.

Ця функція ефективно додає новий продукт до кошика, зберігаючи інформацію в локальному сховищі браузера та оновлюючи стан додатка.

На рисунку Б.2 наведено реалізацію функціоналу для отримання усіх кружок.

```
export function Cups() {
  const [products, setProducts] = useState([]);
  const dispatch = useDispatch();

  useEffect(() => {
    async function load() {
      await api.getAllProducts().then((products) => {
        setProducts(products.data.filter((product) => product.type === "cup"));
      });
    }
    load();
  }, []);
```

Рисунок Б.2 – Код реалізації для завантаження та відображення продуктів

Функція Cups є React-компонентом, що завантажує та відображає продукти типу “cup”. Опишемо кроки, які виконує ця функція.

Перший. Ініціалізація стану:

– використовує хук usestate для створення стану products, який ініціалізується як порожній масив;
– задає функцію setproducts для оновлення стану products.

Другий. Ініціалізація диспетчера: використовує хук usedispatch для отримання функції dispatch, яка використовується для відправлення дій до redux-стору (хоча в цьому конкретному фрагменті коду вона не використовується).

Третій. Завантаження даних з api:

– використовує хук useeffect для виконання побічних ефектів. функція всередині useeffect виконується після першого рендеру компонента;
– всередині useeffect визначається асинхронна функція load, яка викликає api для отримання всіх продуктів;
– викликає метод api.getallproducts, який повертає обіцянку (promise);
– чекає на завершення обіцянки за допомогою await і потім обробляє результати;
– фільтрує продукти, щоб залишити тільки ті, які мають тип “cup”;
– оновлює стан products відфільтрованим масивом продуктів за допомогою setproducts.

Четвертий. Завантаження даних при монтуванні компонента:

– викликає функцію load для завантаження даних з api при першому рендері компонента;
– отже, ця функція завантажує список всіх продуктів з api, фільтрує їх, залишаючи тільки ті, що мають тип “cup”, та зберігає їх у стані компонента для подальшого відображення або обробки.