Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка інтернет-магазину IoT-пристроїв “DigiDive” з використанням React, Nest, Prisma, MySQL

Виконав: студент IV курсу, групи CHc-41
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Головняк М.Р.
(підпис) (прізвище та ініціали)

Керівник _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Липак Г.І.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«_8_» _червня_ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)
за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)
Студенту Головняку Максиму Руслановичу
(прізвище, ім'я, по батькові)
1. Тема роботи Розробка інтернет-магазину IoT-пристроїв "DigiDive" з використанням React, Nest, Prisma, MySQL
Керівник роботи Боднарчук Ігор Орестович, к.т.н., доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «*14*» *травня* 2026 року № *4/9-237*
2. Термін подання студентом завершеної роботи 26 червня 2026 р.
3. Вихідні дані до роботи Літературні та інтернет-джерела
4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз предметної області та постановка завдання на розробку інтернет-магазину "DigiDive". 1.1 Аналіз предметної області. 1.2 Вимоги до інтернет-магазину "DigiDive". 1.3 Пошук актантів та варіантів використання інтернет-магазину "DigiDive". 1.4 Варіанти використання функціональності інтернет-магазину "DigiDive". 1.5 Вибір та обґрунтування використовуваних технологій розробки інтернет-магазину "DigiDive". 1.6 Висновок до першого розділу. 2. Проєктування та реалізація інтернет-магазину "DigiDive". 2.1 Моделювання архітектури інтернет-магазину "DigiDive". 2.2 Перелік інформаційних сутностей та способів їх зберігання. 2.3 Проєктування концептуальної моделі даних. 2.4 Програмна реалізація інтернет-магазину "DigiDive". 2.4.1 Реалізація та структура серверної частини інтернет-магазину "DigiDive". 2.4.2 Реалізація та структура клієнтської частини інтернет-магазину "DigiDive". 2.5 Висновок до другого розділу. 3. Тестування та експлуатація інтернет-магазину "DigiDive". 3.1 Інтерфейс користувача та особливості експлуатації інтернет-магазину "DigiDive". 3.2 Тестування і верифікація інтернет-магазину "DigiDive". 3.3 План заходів для забезпечення захисту інтернет-магазину "DigiDive". 3.4 Висновок до третього розділу. 4. Безпека життєдіяльності, основи охорони праці. Висновки. Перелік джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Титульна сторінка. 2. Актуальність теми. 3. Мета та задачі дослідження. 4. Об'єкт і предмет дослідження. 5. Практичне значення отриманих результатів. 6. Аналіз предметної області. 7. Обґрунтування вибору технологій. 8. Архітектура інтернет-магазину "DigiDive". 9. Моделювання бази даних. 10. Особливості клієнтської частини. 11. Серверна логіка та безпека. 12. Верифікація та тестування. 13. Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Мариненко С.Ю., к.т.н., доц. каф. МТ	01.06.2026	08.06.2026

7. Дата видачі завдання 26 січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	26.01.2026	*Виконано*
2.	Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи	27.01.2026-16.02.2026	*Виконано*
3.	Виконання дослідження аналізу індустрії Інтернету	17.02.2026-10.05.2026	*Виконано*
	речей, вивчення специфіки ринку електронної		
	комерції, формування технічного обґрунтування		
	інструментарію розробки та моделювання взаємодії		
	користувачів із системою.		
	Розроблення архітектури інтернет-магазину,		
	структури бази даних, серверної логіки, адаптивного		
	інтерфейсу користувача, інтеграції платіжного шлюзу		
4.	Оформлення розділу «Аналіз предметної області та	11.05.2026-17.05.2026	*Виконано*
	постановка завдання на розробку інтернет-магазину		
	“DigiDive”»		
5.	Оформлення розділу «Проєктування та реалізація	18.05.2026-24.05.2026	*Виконано*
	інтернет-магазину “DigiDive”»		
6.	Оформлення розділу «Тестування та експлуатація	25.05.2026-31.05.2026	*Виконано*
	інтернет-магазину “DigiDive”»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	01.06.2026-08.06.2026	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони	01.06.2026-08.06.2026	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	09.06.2026-11.06.2026	*Виконано*
10.	Нормоконтроль	12.06.2026-15.06.2026	*Виконано*
11.	Перевірка на плагіат	16.06.2026	*Виконано*
12.	Попередній захист кваліфікаційної роботи	18.06.2026	*Виконано*
13.	Захист кваліфікаційної роботи	26.06.2026	

Студент _____ Головняк М.Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка інтернет-магазину IoT-пристроїв “DigiDive” з використанням React, Nest, Prisma, MySQL // Кваліфікаційна робота освітнього ступеня “Бакалавр” // Головняк Максим Русланович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно-інформаційних систем і програмної інженерії, кафедра комп’ютерних наук, група СНс-41 // Тернопіль, 2026 // С. 64, рис. – 28, табл. – 2, кресл. – 13, додат. – 5, бібліогр. – 39.

Ключові слова: інтернет-магазин, інтернет речей, react, nest, mySQL, prisma, typescript, stripe.

Кваліфікаційна робота присвячена розробленню інтернет-магазину IoT-пристроїв “DigiDive” із використанням сучасних вебтехнологій та клієнт-серверної архітектури.

У першому розділі кваліфікаційної роботи було проведено аналіз предметної області Інтернету речей та систем електронної комерції, визначено основні вимоги до інтернет-магазину “DigiDive” та обґрунтовано вибір технологій для її реалізації.

У другому розділі кваліфікаційної роботи спроєктовано архітектуру інтернет-магазину та концептуальну модель даних, а також визначено структуру клієнтської і серверної частин інтернет-магазину “DigiDive”.

У третьому розділі кваліфікаційної роботи представлено результати роботи інтернет-магазину “DigiDive”, проведено тестування та верифікацію, а також розглянуто заходи щодо забезпечення безпеки веб-додатку.

Об’єкт дослідження: процес розроблення та функціонування вебдодатків електронної комерції.

Предмет дослідження: методи та технології розроблення і тестування клієнт-серверних вебдодатків електронної комерції.

ANNOTATION

Development of the "DigiDive" IoT Device Online Store Using React, Nest, Prisma, and MySQL // Qualification work of the educational level «Bachelor» // Holovniak Maksym Ruslanovych // Ternopil Ivan Pulyu National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, group SNs-41 // Ternopil, 2026 // P. 64, fig. – 28, tabl. – 2, chair. – 13, annexes. – 5, references – 39.

Keywords: internet store, internet of things, react, nest, mySQL, prisma, typescript, stripe.

The qualification work is dedicated to the development of an IoT devices online store “DigiDive” using modern web technologies and a client-server architecture.

The goal of the work is to design and implement a functional, scalable, and secure web application for selling IoT devices.

The first section of the qualification paper considered the IoT and e-commerce domain, defined key requirements for the “DigiDive” store, and justified the choice of technologies.

In the second section of the qualification work, it is considered the architecture of the online store and the conceptual data model, as well as the structure of the client and server parts of the “DigiDive” web application.

The third section of the qualification work presents the results of the practical implementation of the “DigiDive” online store, including system testing and verification, as well as measures aimed at ensuring web application security.

The object of research is the process of development and operation of electronic commerce web applications.

The subject of research is the methods and technologies for developing and testing client-server electronic commerce web applications.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (англ. Application Programming Interface) – програмний інтерфейс для взаємодії між клієнтом та сервером.

DTO (англ. Data Transfer Object) – об'єкт для передачі даних та валідації вхідних запитів.

ER (англ. Entity-Relationship) – модель сутність–зв’язок для проєктування бази даних.

HTTP (англ. Hypertext Transfer Protocol) – протокол передачі гіпертекстових ресурсів у мережі Інтернет.

HTTPS (англ. Hypertext Transfer Protocol Secure) – захищений протокол передачі даних із шифруванням трафіку.

IoT (англ. Internet of Things) – Інтернет речей, мережа взаємопов'язаних фізичних пристроїв.

JWT (англ. JSON Web Token) – формат передачі даних для автентифікації та авторизації між клієнтом і сервером.

ORM (англ. Object-Relational Mapping) – технологія зв'язку бази даних із об'єктно-орієнтованим кодом.

REST (англ. Representational State Transfer) – архітектурний стиль взаємодії компонентів через HTTP-методи.

SPA (англ. Single Page Application) – односторінковий вебзастосунок, що працює без перезавантаження.

URL (англ. Uniform Resource Locator) – уніфікований локатор ресурсів в мережі Інтернет.

СУБД – система управління базами даних, що забезпечує створення, зберігання та обробку інформації в MySQL.

ЗМІСТ

ВСТУП

Актуальність теми. Внаслідок глобалізації, швидкого розвитку цифрової економіки та широкої автоматизації побутових та промислових процесів технології Інтернету речей (IoT) поширилися та стали важливою частиною сучасного інформаційного середовища. Зростаючий попит на інтелектуальні пристрої, датчики та системи керування вимагає створення спеціалізованих високопродуктивних веб-платформ для їх ефективного впровадження та обслуговування. Традиційні методи створення систем електронної комерції не завжди забезпечують гнучкість, необхідну при роботі зі складною ієрархією продуктів та високим навантаженням на серверну інфраструктуру. Тому проектування та впровадження інтернет-магазину IoT-пристроїв “DigiDive” є актуальним напрямком сучасних досліджень у галузі програмної інженерії.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня “Бакалавр” є підвищення ефективності процесів електронної комерції та покращення взаємодії користувачів із каталогом IoT-пристроїв шляхом розроблення інтернет-магазину “DigiDive” на основі клієнт-серверної архітектури.

Для досягнення поставленої мети потрібно виконати ряд завдань, зокрема:

– проаналізувати сучасний стан досліджень у сфері Інтернету речей та систем електронної комерції, а також розробити вимоги до вебзастосунку;

– обґрунтувати вибір сучасних інструментів розробки на основі екосистеми TypeScript (React, NestJS, Prisma, MySQL);

– розробити концептуальну модель даних та змоделювати трирівневу архітектуру між клієнтським та серверним частинами вебдодатку;

– реалізувати серверну та клієнтську частини системи з використанням сучасних методів розробки вебдодатків;

– протестувати та верифікувати REST API та механізми маршрутизації на стороні клієнта;

– розробити та впровадити заходи щодо захисту даних, автентифікації користувачів та інтеграції платіжних сервісів.

Практичне значення одержаних результатів. Практичне значення роботи полягає у створенні повнофункціонального клієнт-серверного інтернет-магазину “DigiDive”, готового до подальшого використання та розгортання. Запропонована архітектура є модульною та масштабованою і може бути використана як основа для розробки комерційних рішень у сфері електронної комерції. Реалізовані підходи для організації безпеки, автентифікації та обробки платежів дозволяють знизити вартість розробки подібних систем, забезпечуючи при цьому високу продуктивність та надійність вебдодатків.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ НА РОЗРОБКУ ІНТЕРНЕТ-МАГАЗИНУ “DIGIDIVE”

1.1 Аналіз предметної області

Сучасний розвиток інформаційних технологій та поширення концепції Інтернету речей сприяли активному впровадженню автоматизованих систем у різні сфери людської діяльності. Технології Інтернету речей використовуються для створення систем безпеки, відеоспостереження, керування освітленням, контролю мікроклімату, автоматизації житлових та комерційних приміщень, а також багатьох інших інженерних рішень. У зв’язку зі зростанням популярності таких систем зростає попит на спеціалізоване обладнання, що продається через платформи електронної комерції [1].

Особливістю ринку IoT-пристроїв є складна структура взаємодії між різними продуктами та окремими категоріями товарів. На відміну від традиційних споживчих товарів, обладнання автоматизації часто використовується як частина єдиної системи, компоненти якої повинні бути сумісні один з одним. До таких компонентів належать системи безпеки, обладнання для інтелектуального освітлення, клімат-контроль, мультимедійне обладнання та інші елементи автоматизованої інфраструктури [2]. У зв'язку з цим виникає потреба у впорядкованому представленні товарів та зручних механізмах їх пошуку.

Для ефективної організації каталогу необхідно використовувати багаторівневу структуру категорій, яка дозволяє логічно групувати товари за призначенням та характеристиками. Це спрощує навігацію користувача та забезпечує швидкий доступ до потрібних продуктів. Крім того, важливою функцією є реалізація зв’язку між категоріями та брендами, що дозволяє відображати лише релевантні варіанти фільтрації відповідно до обраного розділу каталогу.

Ще однією особливістю ринку IoT-пристроїв є його динамічність та залежність від швидкого оновлення технологічних рішень [3]. Виробники часто випускають нові версії обладнання, що призводить до постійної зміни асортименту та поступової заміни старіших моделей. У таких умовах важливого значення набуває не лише актуальність товарної пропозиції, але й здатність інформаційних систем адаптуватися до змін у структурі каталогу без втрати даних.

Проведений аналіз предметної області свідчить про доцільність розробки інтернет магазину “DigiDive” для продажу IoT-пристроїв. Він повинен забезпечувати ефективне управління каталогом товарів, підтримувати гнучкі механізми пошуку та фільтрації, враховувати актуальний стан складських залишків і гарантувати цілісність даних. Реалізація зазначених функціональних можливостей дозволить підвищити зручність роботи користувачів та оптимізувати процес підбору обладнання для побудови автоматизованих систем.

1.2 Вимоги до інтернет-магазину “DigiDive”

У межах розроблення інтернет-магазину IoT-пристроїв “DigiDive” сформуємо систему вимог, яка визначатиме необхідні функції, властивості та обмеження. Передусім визначимо функціональні вимоги, які описують основні процеси роботи. Інтернет-магазин “DigiDive” дозволить збирати та зберігати інформацію про IoT-пристрої, вставляти їх структуроване представлення в базу даних, а також обробляти запити користувачів на пошук та фільтрацію товарів.

Результати запитів будуть відображатись у зручному для аналізу та вибору форматі. Вебдодаток формуватиме картки товарів, встановлюватиме зв’язки між товарами, брендами та ієрархічною структурою категорій, виконуватиме розрахунок загальної вартості замовлення в кошику та підтримуватиме процес оформлення замовлень. Передбачимо механізм динамічного фільтрування виробників відповідно до обраного розділу каталогу,

а також автоматичне оновлення стану кошика у разі зміни наявності товарів. Для адміністрування інтернет-магазином “DigiDive” передбачимо захищений інтерфейс, який дозволятиме керувати товарами, оновлювати ціни, складські залишки та мультимедійні дані.

Нефункціональні вимоги визначатимуть якісні характеристики вебдодатку. Інтернет-магазин забезпечуватиме надійне зберігання даних, коректну обробку інформації про користувачів, товари та замовлення, а також стабільну роботу в умовах високого навантаження [4].

Особлива увага буде приділена продуктивності, а саме час відгуку на операції пошуку, фільтрації та сортування буде мінімальним, а оновлення інтерфейсу виконуватиметься без повного перезавантаження сторінок. Це буде досягнуто за допомогою механізмів кешування даних. Крім того інтернет-магазин “DigiDive” забезпечить безпеку обробки даних, зокрема шляхом шифрування конфіденційних даних та використання токенізованої автентифікації користувачів.

Окрему групу становитимуть експлуатаційні та технічні вимоги. Вебдодаток працюватиме у сучасних браузерах і не вимагатиме встановлення додаткового програмного забезпечення на стороні користувача. Інтерфейс інтернет-магазину “DigiDive” буде адаптивним та забезпечить коректне відображення на різних пристроях, включаючи персональні комп’ютери, планшети та мобільні телефони.

Архітектурно вебдодаток буде реалізований у клієнт-серверній моделі, взаємодіятиме з реляційною базою даних через рівень доступу до даних, забезпечуватиме захищений обмін інформацією через безпечні канали зв’язку та підтримуватиме стабільну роботу навіть під час короткочасних перебоїв у роботі мережі.

1.3 Пошук актантів та варіантів використання інтернет-магазину “DigiDive”

Визначимо основних акторів та їхні ролі в інтернет-магазині IoT-пристроїв “DigiDive”. Взаємодія з вебплатформою здійснюватиметься трьома типами користувачів, а саме неавторизованим користувачем, авторизованим та адміністратором. Кожен із акторів виконує дії в рамках визначених повноважень та можливостей інтерфейсу.

Неавторизований користувач має доступ до публічної частини вебдодатку, а саме зможе переглядати каталог та інформацію про певні товари.

Авторизований користувач, окрім основних можливостей перегляду, зможе створювати персональний список вибраних товарів, добавляти в кошик, оформлювати замовлення та переглядати особисту історію замовлень.

Адміністратор матиме повний контроль над інтернет-магазином, включаючи наповнення каталогу товарів, створення категорій та брендів, відстеження оформлених замовлень та перегляд статистики інтернет-магазину.

На основі ролей акторів сформуємо реєстр варіантів використання, що описує типові сценарії взаємодії користувача з інтернет-магазином “DigiDive”. У таблиці 1.1 представлено реєстр варіантів використання.

Таблиця 1.1 – Реєстр варіантів використання

№	Назва варіанта	Актори	Короткий опис
1	**2**	**3**	**4**
1	Реєстрація та авторизація	Неавторизований користувач	Створення облікового запису та вхід у систему
2	Перегляд та фільтрація каталогу	Усі користувачі	Перегляд товарів, сортування та фільтрація
3	Керування кошиком	Авторизований користувач	Додавання товарів до кошика, зміна їх кількості

Продовження таблиці 1.1

1	2	3	4
4	Керування списком обраних товарів	Авторизований користувач	Додавання товарів до списку обраного
5	Оформлення замовлення	Авторизований користувач	Заповнення контактних даних та адреси доставки
6	Перегляд історії замовлень	Авторизований користувач	Перегляд замовлень та їх статусів
7	Керування товарами	Адміністратор	Створення, редагування та архівація товарів
8	Керування деревом категорій	Адміністратор	Створення, редагування та видалення категорій
9	Керування брендами	Адміністратор	Додавання та оновлення інформації про виробників товарів
10	Керування замовленнями	Адміністратор	Перегляд замовлень та зміна їх статусів
11	Перегляд статистики	Адміністратор	Загальний дохід, кількість замовлень, користувачів та продуктів

Ці варіанти використання формують основу функціональної поведінки інтернет-магазину та показують, як інтернет-магазин IoT-пристроїв “DigiDive” реагує на дії різних типів користувачів та забезпечує логіку роботи.

Крім того, описані варіанти використання дозволяють детально описати функціональні вимоги та чітко зрозуміти обсяг робіт, що реалізуються на етапі розроблення.

1.4 Варіанти використання функціональності інтернет-магазину “DigiDive”

Для опису функціональних можливостей інтернет-магазину IoT-пристроїв “DigiDive” сформуємо перелік основних варіантів використання, що ілюструють взаємодію користувачів із вебплатформою. Кожен із варіантів використання описує окремий сценарій роботи та дозволяє визначити дії, пов’язані з різними типами користувачів [5]. Такий підхід сприяє кращому розумінню структури інтернет-магазину “DigiDive”, розмежуванню ролей акторів та формуванню вимог до функціональності вебдодатку.

Також визначено варіанти використання, які доступні адміністраторам вебзастосунку. До них належать керування товарами, категоріями та брендами, відстеження замовлень користувачів, а також перегляд статистичних даних інтернет-магазину. Реалізація зазначених функцій забезпечує підтримку актуальності каталогу, контроль процесу продажів та ефективне адміністрування. На рисунку 1.1 представлено UML-діаграму варіантів використання для інтернет-магазину IoT-пристроїв “DigiDive”.

Рисунок 1.1 – UML-діаграма варіантів використання для інтернет-магазину IoT-пристроїв “DigiDive”

Розгляд зазначених сценаріїв дозволяє визначити функціональні можливості кожного актора, встановити межі взаємодії із вебдодатком та забезпечити чіткий розподіл прав доступу відповідно до вимог інтернет-магазину IoT-пристроїв “DigiDive”.

1.5 Вибір та обґрунтування використовуваних технологій розробки інтернет-магазину “DigiDive”

Для розроблення інтернет-магазину IoT-пристроїв “DigiDive” обрано сучасний стек технологій, який забезпечуватиме ефективність, масштабованість та стабільну роботу, а також зручність подальшої підтримки та розвитку.

На фронтенд-частині використовується TypeScript у поєднанні з React. Такий підхід дозволяє реалізувати компонентну архітектуру інтерфейсу з чіткою типізацією даних, що зменшує кількість потенційних помилок під час розроблення та підвищує надійність коду [6]. Крім того, TypeScript спрощує роботу зі складними структурами даних, особливо з ієрархією категорій, товарами та параметрами фільтрації, що є критично важливим в галузі Інтернет речей.

Візуальна складова та графічний інтерфейс користувача будуються на основі бібліотеки компонентів Chakra UI. Вибір цього інструменту ґрунтується на можливості швидкого створення адаптивного дизайну. Він надає набір готових модульних елементів, що дозволило уніфікувати дизайн-систему інтернет-магазину, скоротити час розроблення та забезпечити коректне відображення інтерфейсу на мобільних та десктопних пристроях.

Для керування станом інтернет-магазину “DigiDive” використовується Redux Toolkit. Він забезпечує централізоване зберігання та обробку стану, включаючи дані про товари, кошик користувача, обрані товари та параметри фільтрів. Це дозволяє забезпечити узгоджену взаємодію між різними компонентами інтерфейсу та спрощує масштабування клієнтської частини вебзастосунку.

Оплата замовлення в інтернет-магазині “DigiDive” реалізується завдяки інтеграції міжнародного платіжного сервісу Stripe. Ця платформа була обрана для забезпечення безпечного процесу проведення безготівкових розрахунків за допомогою банківських карток. Інтеграція Stripe API дозволяє передати обробку фінансових даних надійному провайдеру з дотриманням стандарту PCI DSS, а механізм вебхуків забезпечує миттєве оновлення статусів замовлень після оплати.

Додатково використовується RTK Query, який відповідає за взаємодію з серверною частиною через API. Цей інструмент забезпечує автоматичне кешування запитів, синхронізацію серверних даних та зменшення кількості повторних HTTP-запитів, що позитивно впливає на продуктивність інтерфейсу та швидкість завантаження сторінок каталогу [7].

Серверна частина реалізується з використанням фреймворку NestJS, що дозволяє створити модульну архітектуру застосунку з чітким розділенням бізнес-логіки, контролерів та сервісів. Такий підхід підвищує читабельність коду, спрощує масштабування та підтримку інтернет-магазину “DigiDive”. Вбудовані механізми валідації та захисту маршрутів допомагають підвищити безпеку вебдодатку.

Для роботи з базою даних використовується Prisma ORM, яка забезпечує типобезпечну взаємодію з реляційною СУБД. Це дозволяє формувати запити на рівні високорівневих моделей даних, зменщує ризик помилок та спрощує роботу зі складними зв’язками між сутностями, такими як товари, категорії, користувачі та замовлення.

MySQL використовується як система керування базами даних, яка забезпечує надійне зберігання інформації, підтримку реляційних зв’язків та цілісність даних [8]. Вона ефективно обробляє великі обсяги даних, що є важливим для інтернет-магазину з динамічно змінним каталогом товарів та великою кількістю транзакцій.

1.6 Висновок до першого розділу

В першому розділі кваліфікаційної роботи було проведено аналітичний огляд предметної області, визначено основні вимоги до інтернет магазину IoT-пристроїв “DigiDive” та основних акторів. обґрунтовано вибір технологій для реалізації проєкту. Аналіз індустрії Інтернету речей (IoT) виявив специфічні виклики, які пов’язані з типом продукту та ієрархічною структурою каталогу товарів. Встановлено, що універсальні рішення для електронної комерції не дозволяють ефективно вибирати пристрої Інтернет речей, що підтверджує актуальність розроблення інтернет-магазину IoT-пристроїв “DigiDive”.

Сформульовано функціональні, нефункціональні та технічні вимоги до вебдодатку. Визначено трьох акторів (гість, користувач, адміністратор) та побудовано UML-діаграму варіантів використання, що дозволило формалізувати основні бізнес-процеси інтернет-магазину.

Обґрунтовано вибір технологічного стеку, що включає React і TypeScript на стороні клієнта з використанням Redux Toolkit та RTK Query, а також NestJS, Prisma ORM і MySQL на стороні сервера. Для створення інтерфейсу використано Chakra UI, а для обробки платіжних операцій застосовано Stripe. Використані технології забезпечать продуктивність, безпеку та масштабованість інтернет-магазину IoT-пристроїв “DigiDive”.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ІНТЕРНЕТ-МАГАЗИНУ “DIGIDIVE”

2.1 Моделювання архітектури інтернет-магазину “DigiDive”

Проєктування архітектури інформаційної системи є одним із ключових етапів розроблення програмного забезпечення, оскільки архітектурні рішення визначають надійність, масштабованість та ефективність подальшої роботи системи [9]. Для інтернет-магазину IoT-пристроїв “DigiDive” обрано трирівневу клієнт-серверну архітектуру, яка розподіляє завдання між рівнем представлення, рівнем бізнес-логіки та рівнем даних.

Перший рівень представлений клієнтською частиною, реалізованою у вигляді односторінкового вебдодатку з використанням React та TypeScript. Цей рівень відповідає за відображення інтерфейсу користувача та забезпечення взаємодії з вебплатформою. Для централізованого керування станом інтернет-магазину “DigiDive” використовується Redux Toolkit, а обмін даними із серверною частиною здійснюється за допомогою RTK Query.

Другий рівень архітектури реалізує бізнес-логіку та побудований на основі фреймворку NestJS, що працює на платформі Node.js. Серверна частина має модульну структуру, що дозволяє розділяти функції між окремими компонентами вебдодатку. Для кожної бізнес-сутності, такої як користувачі, товари, категорії, бренди і замовлення, передбачено окремі модулі, які містять контролери, сервіси та допоміжні компоненти.

Контролери відповідають за приймання HTTP-запитів та взаємодію з клієнтською частиною через REST API. Основна бізнес-логіка зосереджена у сервісах, які обробляють дані, перевіряють згідно бізнес-правил та виконують необхідні операції з базою даних. Для забезпечення коректності вхідних даних використовуються механізми валідації, а контроль доступу до функціональності системи реалізується за допомогою рольової моделі та JWT-автентифікації.

Третій рівень архітектури відповідає за зберігання та обробку даних. Для цього використовується СУБД MySQL у поєднанні з інструментом Prisma ORM. Використання Prisma дозволяє здійснювати типобезпечну взаємодію з базою даних та спрощує роботу зі складними зв’язками між сутностями вебзастосунку [10]. Крім того, ORM забезпечує підтримку міграцій та централізоване керування структурою бази даних.

СУБД MySQL використовується для зберігання інформації про користувачів, товарів, категорій, брендів, замовлень та інших об’єктів предметної області. Реляційна модель даних забезпечує цілісність даних та підтримку зв’язків між окремими сутностями, що є важливим для належного функціонування інтернет-магазину. На рисунку 2.1 представлено архітектурну модель інтернет-магазину IoT-пристроїв “DigiDive”.

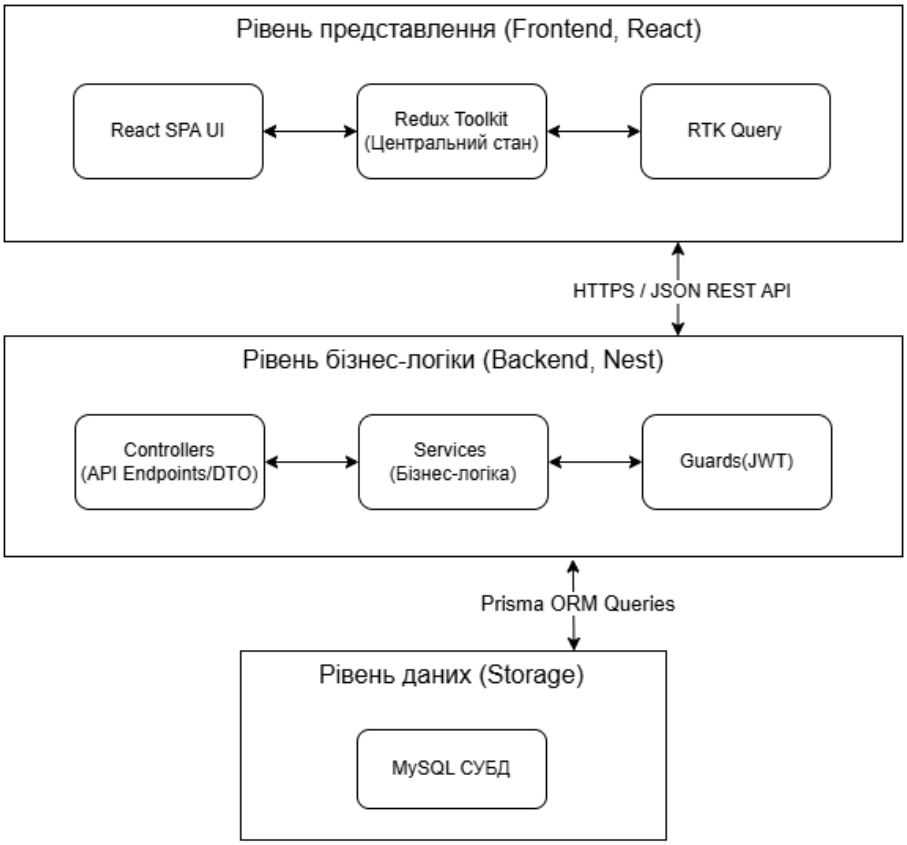

Рисунок 2.1 – Архітектурна модель інтернет-магазину IoT-пристроїв “DigiDive”

Вищезазначений архітектурний підхід забезпечує чітке розподілення обов’язків між окремими компонентами інтернет-магазину IoT-пристроїв “DigiDive”, спрощуючи супровід та подальший розвиток. Завдяки відокремленню клієнтської частини від серверної бізнес-логіки зміни в інтерфейсі користувача не впливають на основні механізми обробки даних та виконання бізнес-процесів.

Підвищення продуктивності інтернет-магазину досягається завдяки використанню асинхронної моделі обробки запитів, реалізованої засобами Node.js та NestJS [11]. Крім того, використання RTK Query на стороні клієнта дозволяє використовувати механізми кешування даних та зменшувати кількість повторних звернень до сервера. Це зменшує час відгуку інтернет-магазину “DigiDive” та допомагає краще використовувати ресурси сервера.

Отже, обраний набір архітектурний рішень формує основу для стабільної роботи інтернет-магазину IoT-пристроїв “DigiDive”, дозволяє розширити його функціональність та підтримує його подальше масштабування зі зростанням кількості користувачів і обсягу даних.

2.2 Перелік інформаційних сутностей та способів їх зберігання

У межах розроблення інтернет-магазину IoT-пристроїв “DigiDive” важливими етапами є визначення основних інформаційних сутностей та проєктування структури їх зберігання. Організація бази даних безпосередньо впливає на продуктивність виконання запитів, цілісність даних, а також на можливість виконання динамічної фільтрації та пошуку на стороні клієнта. Для забезпечення надійного зберігання інформації, логічної узгодженості та масштабованості інтернет-магазину “DigiDive” було обрано систему керування базами даних MySQL. Взаємодія серверної частини з базою даних реалізується за допомогою Prisma ORM, яка забезпечує об’єктно-реляційне відображення та типобезпечну роботу з моделями даних.

У рамках Prisma ORM опис інформаційних сутностей виконується за допомогою моделей, на основі яких автоматично створюються відповідні таблиці та поля у базі даних. Це дозволяє уніфікувати структуру даних, зменшити ймовірність помилок під час розробки та спростити підтримку схеми бази даних.

На основі аналізу бізнес-процесів предметної області та вимог до структури даних сформовано перелік основних інформаційних сутностей інтернет-магазину IoT-пристроїв “DigiDive”. У таблиці 2.1 представлено перелік інформаційних сутностей.

Таблиця 2.1 – Перелік інформаційних сутностей

Сутність	**Пояснення**	**Основні поля та первинний ключ**
User	Користувач системи	id (PK), email, password, role, createdAt, updatedAt
Product	Товар каталогу	id (PK), name, imagePath, description, price, stock, warrantyMonths, createdAt, updatedAt, isActive, categoryId, brandId
Brand	Виробник товару	id (PK), name
Category	Категорія товару	id (PK), name, imagePath, parentId
Favorite	Список обраних товарів	id (PK), userId, productId
Cart	Кошик користувача	id (PK), userId, productId, quantity, addedAt
Order	Замовлення користувача	id (PK), userId, totalPrice, deliveryFee, status, country, fullName, company, address, postCode, city, phone, createdAt, updatedAt
OrderItem	Конкретний елемент замовлення	id (PK), orderId, productId, quantity, price

Для підвищення продуктивності інтернет-магазину IoT-пристроїв “DigiDive” під час виконання пошукових та фільтраційних запитів на рівні MySQL передбачено використання індексів. Індексування за ключовими полями, такими як категорії та статуси товарів, дозволяє скоротити час виконання вибірок та підвищити ефективність обробки запитів у великих обсягах даних.

Цілісність даних забезпечується на рівні бази даних шляхом налаштування правил каскадного видалення та обмежень цілісності. Зокрема, для кошика та списку обраних товарів реалізовано каскадне видалення при видаленні користувача, тоді як для елементів замовлень застосовано обмеження видалення пов’язаних товарів, що запобігає виникненню застарілих записів та порушенню цілісності даних.

2.3 Проєктування концептуальної моделі даних

Важливим етапом розроблення інтернет-магазину IoT-пристроїв “DigiDive” є створення ER-діаграми бази даних, яка відображає структуру предметної області та взаємозв’язки між сутностями без прив’язки до конкретної системи керування базами даних [12]. Метою цього етапу є визначення основних сутностей інтернет-магазину, їхніх атрибутів та встановлення логічних зв’язків між ними. Це дозволяє забезпечити цілісність, узгодженість та масштабованість вебзастосунку. Побудова ER-діаграми виконується на основі аналізу бізнес-процесів електронної комерції з урахуванням специфіки роботи з IoT-пристроями.

Під час створення ER-діаграми особлива увага приділяється взаємозв’язкам між сутностями предметної області. Сутність товару пов’язується з категоріями та виробниками, утворюючи реляційні відношення типу “один-до-багатьох”. Важливою особливістю моделі є існування рекурсивного зв’язку сутності категорії із самою собою, що дозволяє

реалізувати ієрархічну деревоподібну структуру з необмеженою кількістю рівнів вкладеності.

Персоналізація взаємодії з користувачем реалізується через сутності кошика та списку обраних товарів, які пов’язують користувача з відповідними товарами. Транзакційна частина інтернет-магазину “DigiDive” моделюється за допомогою сутностей замовлення та їх елементів, що забезпечує збереження інформації про склад замовлення та фіксацію вартості товарів на момент здійснення покупки. На рисунку 2.2 представлено ER-діаграму бази даних для інтернет-магазину IoT-пристроїв “DigiDive”.

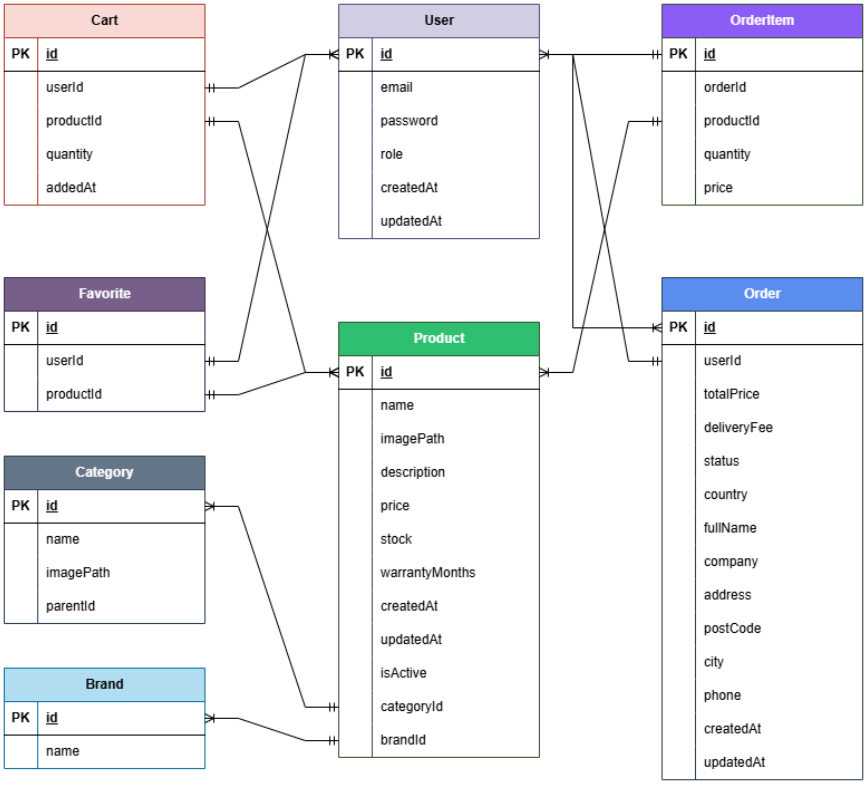

Рисунок 2.2 – ER-діаграма бази даних для інтернет-магазину IoT-пристроїв “DigiDive”

Представлена ER-діаграма слугуватиме основою для подальшого проєктування логічної та фізичної структури бази даних у СУБД MySQL. Встановлені зв’язки між сутностями визначають правила забезпечення

цілісності даних та поведінку інтернет-магазину IoT-пристроїв “DigiDive” при виконанні операцій додавання, оновлення та видалення записів.

Зокрема, зв’язки між користувачем, кошиком та списком обраного передбачають каскадне видалення пов’язаних записів у разі видалення облікового запису. Водночас зв’язок між товарами та елементами замовлень є обов’язковим, що унеможливлює видалення товару, якщо він використовується в існуючих замовленнях. Таким чином, розроблена ER-діаграма гарантує узгодженість даних, підтримує бізнес-логіку інтернет-магазину та створює основу для ефективної реалізації функцій вебзастосунку.

2.4 Програмна реалізація інтернет-магазину “DigiDive”

Етап програмної реалізації інформаційної системи полягає у трансформації спроєктованих архітектурних рішень, моделей даних та варіантів використання у функціонуючий програмний продукт. Реалізація здійснюється на основі розділеної клієнт-серверної архітектури (Decoupled Architecture), де фронтенд і бекенд функціонують як незалежні застосунки та взаємодіють один з одним через HTTPS за допомогою REST API.

Такий підхід чітко розділяє відповідальності між рівнями інтернет-магазину “DigiDive”, дозволяє незалежне розроблення інтерфейсу користувача та серверної логіки, спрощує масштабування окремих компонентів і підвищує загальну гнучкість вебзастосунку.

Інтернет магазин IoT-пристроїв “DigiDive” реалізується із використанням мови TypeScript на всіх рівнях, що забезпечує уніфіковану типізацію даних між клієнтською та серверною частинами. Це зменшує кількість помилок, пов’язаних з відмінностями в структурах даних, та полегшує подальше обслуговування та розширення функціональності інтернет-магазину [13].

Процес реалізації включає ініціалізацію проєктів, налаштування середовищ розробки, створення базової структури частин за допомогою CLI-інструментів, організацію модульної архітектури бекенду, а також

формування компонентної структури фронтенду та інтеграцію глобального стану застосунку. Деталі реалізації кожного рівня представлені у наступних підрозділах.

2.4.1 Реалізація та структура серверної частини інтернет-магазину “DigiDive”

Серверна частина інтернет-магазину IoT-пристроїв “DigiDive” базується на основі платформи Node.js з використанням фреймворку NestJS. Цей вибір зумовлений можливістю створення масштабованої, модульної та структурованої архітектури, яка відповідає вимогам до сучасних вебзастосунків електронної комерції.

Ініціалізація проєкту виконується за допомогою інструменту Nest CLI, який дозволяє автоматично генерувати базову структуру застосунку, файлову структуру, а також налаштовувати конфігурації TypeScript та необхідних залежностей. Використання CLI дозволяє значно пришвидшити початок розробки та забезпечує узгоджену структуру проєкту відповідно до рекомендацій фреймворку.

Для створення початкового шаблону застосунку використовується команда ініціалізації NestJS, після чого створюється базова архітектура серверної частини та підключаються основні конфігураційні модулі [14]. Після успішного виконання ініціалізації середовище автоматично генерує власну структуру проєкту, яка потім налаштовується відповідно до бізнес-логіки інтернет-магазину IoT-пристроїв “DigiDive”.

Крім того, варто зазначити, що отримана базова структура проєкту є відправною точкою для подальшого розширення серверної частини. Це дозволяє здійснювати поступову інтеграцію нових модулів, розширювати функціональність та розділяти бізнес-логіку між окремими компонентами. Такий підхід особливо важливий для вебдодатків електронної комерції, оскільки забезпечує можливість незалежного розроблення окремих частин

вебзастосунку без впливу на загальну архітектуру. На рисунку 2.3 представлено результат успішної ініціалізації серверної частини інтернет-магазину “DigiDive” за допомогою Nest CLI.

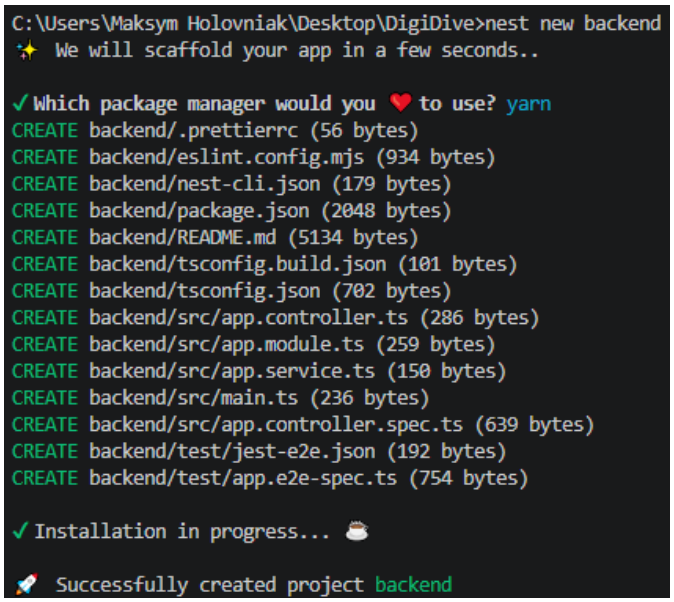

Рисунок 2.3 – Результат успішної ініціалізації серверної частини інтернет-магазину “DigiDive” за допомогою Nest CLI

Після формування базової структури здійснюється налаштування серверної частини згідно предметної області інтернет-магазину. Серверний застосунок організовано за модульним принципом, що означає поділ процесів на різні незалежні блоки. Кожен модуль відповідає за конкретну бізнес-сутність, зокрема користувачів, товари, категорії, бренди та замовлення.

На цьому етапі також визначається загальна структура каталогів та файлів проєкту, що забезпечує логічну організацію коду та полегшує обслуговування. Такий підхід дозволяє швидко орієнтуватися у структурі інтернет-магазину та зменшує складність інтеграції нових функціональних модулів у майбутньому. На рисунку 2.4 представлено файлову структуру серверної частини інтернет-магазину IoT-пристроїв “DigiDive”.

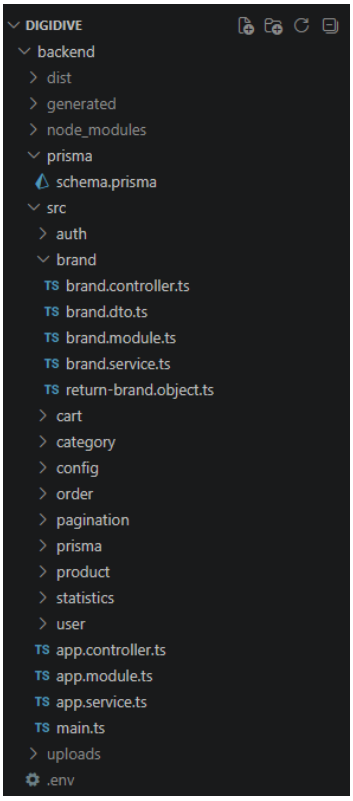

Рисунок 2.4 – Файлова структура серверної частини інтернет-магазину IoT-пристроїв “DigiDive”

HTTP-запити обробляються на рівні контролерів, які визначають маршрути REST API та приймають вхідні дані у форматі JSON. Початкова валідація даних реалізується за допомогою об’єктів DTO у поєднанні з бібліотекою class-validator та механізмом ValidationPipe, який перевіряє типи і структури вхідних даних.

Бізнес-логіка інтернет-магазину “DigiDive”, зокрема розрахунок вартості продуктів у замовлені, перевірка наявності товарів та керування складськими залишками, інкапсульована у сервісному шарі. Зв’язок з базою даних здійснюється через Prisma ORM, який забезпечує типобезпечний доступ до MySQL та спрощує виконання запитів і міграцій.

Захист маршрутів реалізовано за допомогою декораторів, які перехоплюють запити, перевіряють JWT-токени та контролюють доступ користувачів відповідно до їхніх ролей.

2.4.2 Реалізація та структура клієнтської частини інтернет-магазину “DigiDive”

Клієнтська частина інтернет-магазину IoT-пристроїв “DigiDive” реалізована на основі бібліотеки React з використанням мови програмування TypeScript. Вибір цього технологічного стеку ґрунтується на необхідності створення масштабованого, компонентного та типобезпечного інтерфейсу користувача, який відповідає вимогам сучасних вебзастосунків електронної комерції [15].

Ініціалізація клієнтського застосунку виконується за допомогою інструменту Vite, який забезпечує швидке створення проєкту, оптимізовану систему збірки та підтримку сучасних можливостей JavaScript і TypeScript [16]. Після виконання команди ініціалізації створюється базова структура проєкту, яка потім використовується як основа для розроблення інтерфейсу інтернет-магазину “DigiDive”. На рисунку 2.5 представлено результат успішної ініціалізації клієнтської частини інтернет-магазину “DigiDive” за допомогою Vite.

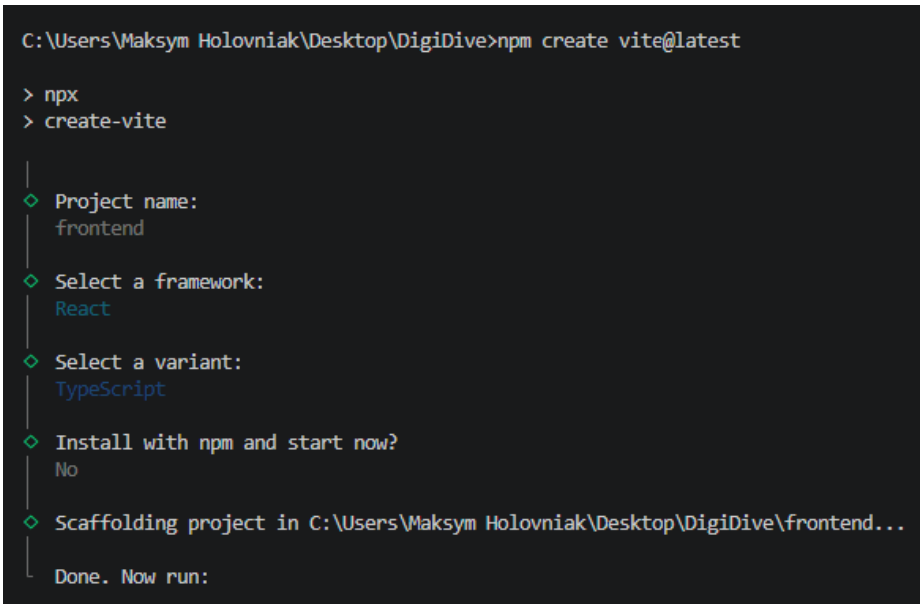

Рисунок 2.5 – Результат успішної ініціалізації клієнтської частини інтернет-магазину “DigiDive” за допомогою Vite

Після створення базової структури виконується організація клієнтського застосунку відповідно до принципів розділення відповідальності (Separation of Concerns). Вихідний код поділяється на логічні частини, такі як компоненти інтерфейсу, сторінки застосунку, модулі взаємодії з API та глобальне сховище стану.

Такий підхід дозволяє забезпечити повторне використання компонентів, спрощує супровід коду та підвищує його структурованість для подальшого масштабування інтернет-магазину “DigiDive”. На рисунку 2.6 представлено файлову структуру клієнтської частини інтернет-магазину IoT-пристроїв “DigiDive”.

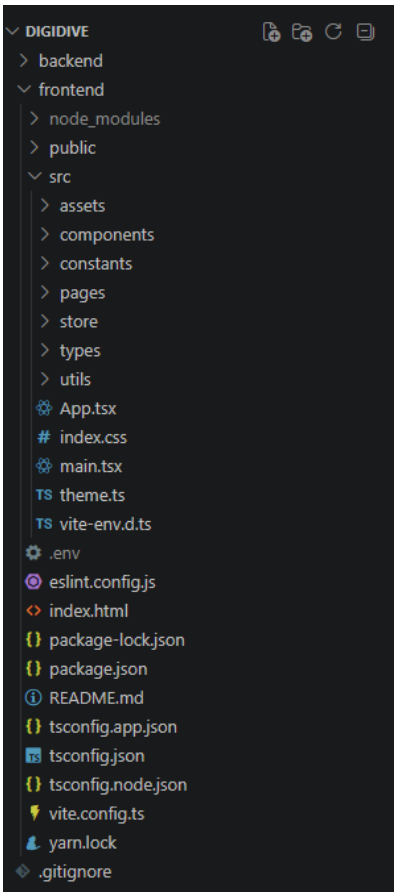

Рисунок 2.6 – Файлова структура клієнтської частини інтернет-магазину IoT-пристроїв “DigiDive”

Керування станом клієнтського застосунку виконується за допомогою Redux Toolkit, який забезпечує централізоване зберігання даних про товари,

кошик користувача, обрані товари та фільтри каталогу. Це дозволяє уникнути надмірної передачі даних між компонентами та забезпечує узгодженість стану інтерфейсу.

Взаємодія з серверною частиною здійснюється за допомогою RTK Query, який виконує асинхронні HTTP-запити до REST API. Крім того, механізм кешування дозволяє повторно використовувати отримані дані без надсилання нового запиту до сервера, що підвищує продуктивність та зменшує навантаження на серверну частину [17].

Загалом клієнтська частина реалізована як SPA, що дозволяє швидко переміщатись між сторінками без повного перезавантаження інтернет-магазину та покращує користувацький досвід.

2.5 Висновок до другого розділу

В другому розділі кваліфікаційної роботи було проведено повне проєктування та програмну реалізацію інтернет-магазину IoT-пристроїв “DigiDive”. На основі аналізу бізнес-процесів електронної комерції розроблено архітектурні та програмні рішення, які забезпечують виконання основних функціональних вимог вебзастосунку.

В рамках проєктування архітектури інтернет-магазину обґрунтовано доцільність використання трирівневої децентралізованої клієнт-серверної моделі. Були визначені межі відповідальності між рівнем представлення даних, рівнем бізнес-логіки та рівнем даних. Такий підхід логічно розділяє компоненти вебзастосунку, підвищує масштабованість та спрощує подальше обслуговування інтернет-магазину IoT-пристроїв “DigiDive”.

Під час формування інформаційної моделі вебдодатку визначено перелік основних сутностей, необхідних для роботи інтернет-магазину, та обґрунтовано використання реляційної системи керування базами даних MySQL для їх зберігання. На основі аналізу предметної області здійснено структурування сутностей із визначенням їхніх атрибутів та ключових полів.

Крім того, було побудовано ER-діаграму, яка відображає взаємозв’язки між сутностями інтернет-магазину. Це дозволило забезпечити цілісність даних, а також створити основу для ефективної організації реляційної структури.

На етапі програмної реалізації виконано побудову клієнтської та серверної частин інтернет-магазину “DigiDive”. Серверна частина реалізована на основі фреймворку NestJS та спроєктована за модульним принципом, що дозволяє ізолювати бізнес-логіку окремих сутностей. Для передачі та валідації даних використано об’єкти DTO, а захист маршрутів реалізовано за допомогою JWT-механізму та декораторів.

Клієнтська частина реалізована на основі бібліотеки React з використанням інструменту Vite, що дозволило створити SPA. Архітектура фронтенду побудована за принципом розділення відповідальності (Separation of Concerns). Для керування станом застосунку використано Redux Toolkit, а для взаємодії з серверною частиною RTK Query, що дозволяє кешувати запити, зменшує кількість повторних звернень до API та оптимізовує продуктивності інтерфейсу.

Таким чином, результати другого розділу підтверджують коректність обраних архітектурних рішень та забезпечують повну реалізацію функціональної моделі інтернет-магазину IoT-пристроїв “DigiDive” на клієнтському та серверному рівнях.

РОЗДІЛ 3. ТЕСТУВАННЯ ТА ЕКСПЛУАТАЦІЯ ІНТЕРНЕТ-МАГАЗИНУ “DIGIDIVE”

3.1 Інтерфейс користувача та особливості експлуатації інтернет-магазину “DigiDive”

Етап аналізу користувацького інтерфейсу та практичної експлуатації інтернет-магазину IoT-пристроїв “DigiDive” дозволяє оцінити якість перенесення бізнес-вимог у візуальне середовище. Інтерфейс користувача розроблено з використанням сучасних принципів адаптивного дизайну та компонентного підходу, що забезпечує коректне відображення елементів інтернет-магазину на пристроях з різною роздільною здатністю екрана.

Головна сторінка інтернет-магазину є основним елементом взаємодії користувача з платформою та виконує інформаційну й навігаційну функції. На ній розміщено банерний блок, секції з актуальними товарами та елементи швидкого переходу до основних розділів каталогу. На рисунку 3.1 представлено інтерфейс головної сторінки інтернет-магазину IoT-пристроїв “DigiDive”.

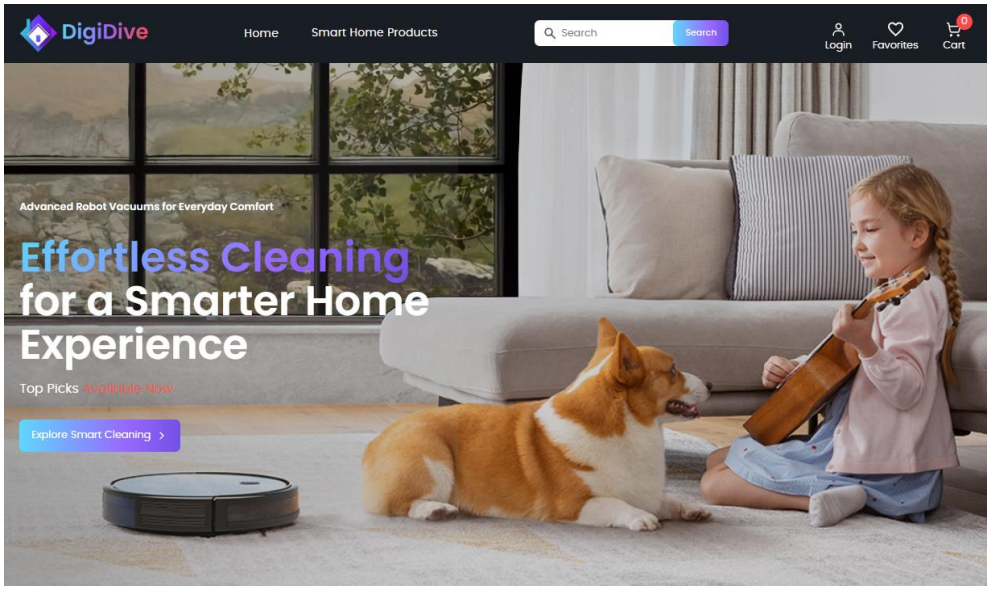

Рисунок 3.1 – Інтерфейс головної сторінки інтернет-магазину IoT-пристроїв “DigiDive”

Аналіз інтерфейсу головної сторінки інтернет-магазину IoT-пристроїв “DigiDive” показує чітке візуальне зонування контенту, що сприяє утриманню уваги користувача та спрощує початкове ознайомлення з асортиментом вебплатформи [18]. Така структура інтерфейсу забезпечує логічний розподіл елементів та підвищує зручність взаємодії з вебдодатком.

Для забезпечення швидкої навігації у верхній частині інтерфейсу реалізовано навігаційну панель. Меню категорій, розміщене у шапці сайту, динамічно формує свою структуру на основі даних, отриманих з бази даних, та відображає ієрархічне дерево розділів. Це забезпечує користувачеві швидкий доступ до будь-якої категорії товарів без необхідності додаткових переходів. На рисунку 3.2 представлено навігаційне меню категорій у шапці інтернет-магазину IoT-пристроїв “DigiDive”.

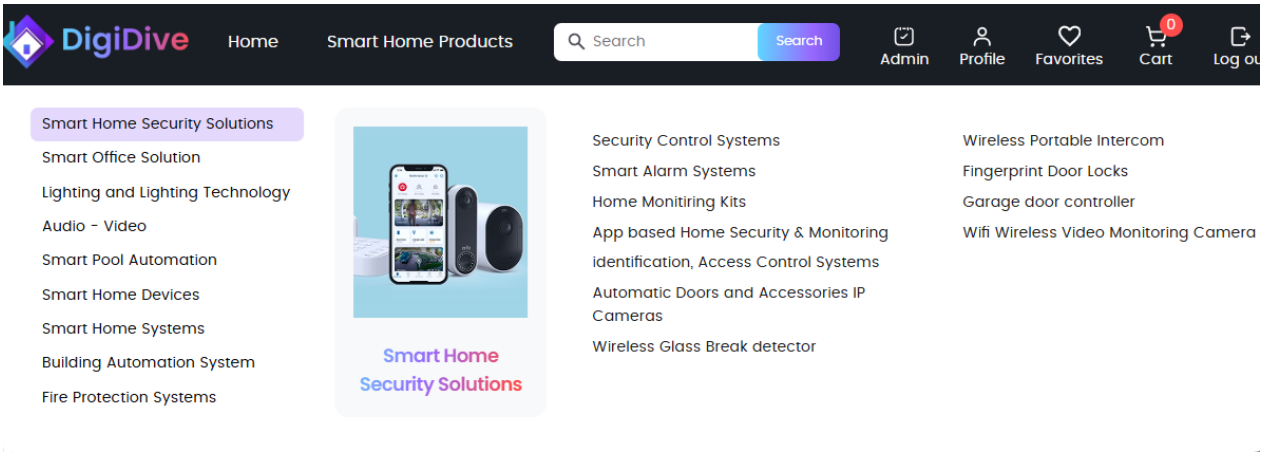

Рисунок 3.2 – Навігаційне меню категорій у шапці інтернет-магазину IoT-пристроїв “DigiDive”

Нижня частина кожної публічної сторінки інтернет-магазину завершується статичним підвалом (footer), який виконує інформаційну, юридичну та навігаційну функції. Підвал сайту містить основні категорії товарів, посилання на соціальні мережі, а також правову інформацію про захист персональних даних та інтелектуальної власності інтернет-магазину “DigiDive”. На рисунку 3.3 представлено інтерфейс підвалу інтернет-магазину IoT-пристроїв “DigiDive”.

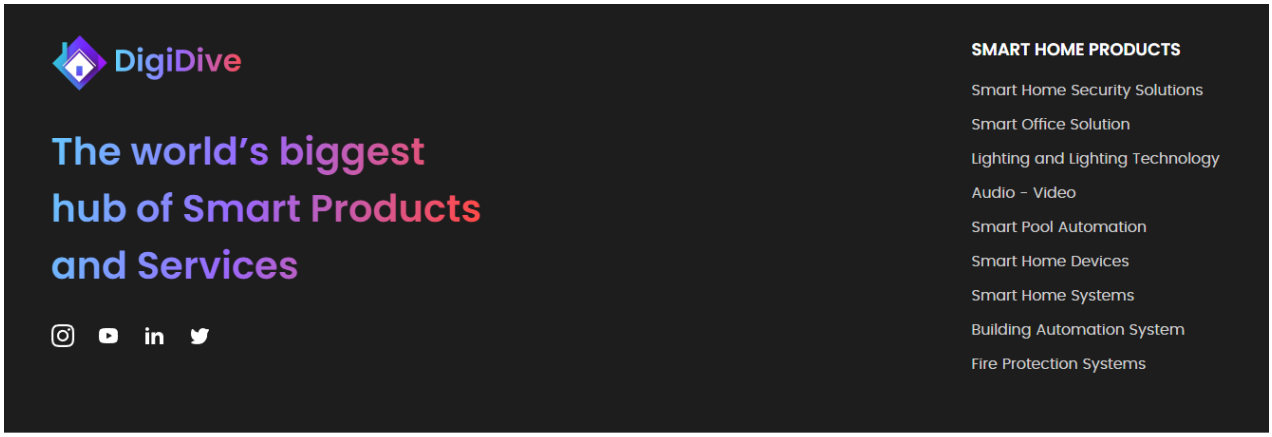

Рисунок 3.3 – Інтерфейс підвалу інтернет-магазину IoT-пристроїв “DigiDive”

Для забезпечення доступу до персоналізованого функціоналу, таких як оформлення замовлень та формування списку обраних товарів, в інтернет-магазині реалізовано механізм автентифікації користувачів. Форма авторизації забезпечує захищений інтерфейс для введення облікових даних та підтримує перевірку коректності введених даних перед їх передачею на сервер.

Після успішної перевірки введених даних генеруються JWT-токени доступу та оновлення (access token і refresh token), термін дії яких становить 15 хвилин та 3 дні відповідно. Залежно від вибору користувачем опції “Remember me” визначається тривалість збереження автентифікаційних даних та спосіб їх зберігання. Отримані токени використовуються для подальшої ідентифікації користувача та забезпечують доступ до захищених функцій інтернет-магазину IoT-пристроїв “DigiDive”.

Крім того, вебзастосунок підтримує можливість автентифікації через OAuth-провайдери, зокрема Google та GitHub, що дозволяє користувачам швидко входити без необхідності створення окремого облікового запису. Використання сторонніх сервісів автентифікації спрощує процес входу та підвищує зручність взаємодії користувачів із вебплатформою.

Інтерфейс форми авторизації розроблено відповідно до сучасних вимог щодо зручності використання та безпеки вебзастосунків. На рисунку 3.4 представлено форму авторизації інтернет-магазину IoT-пристроїв “DigiDive”.

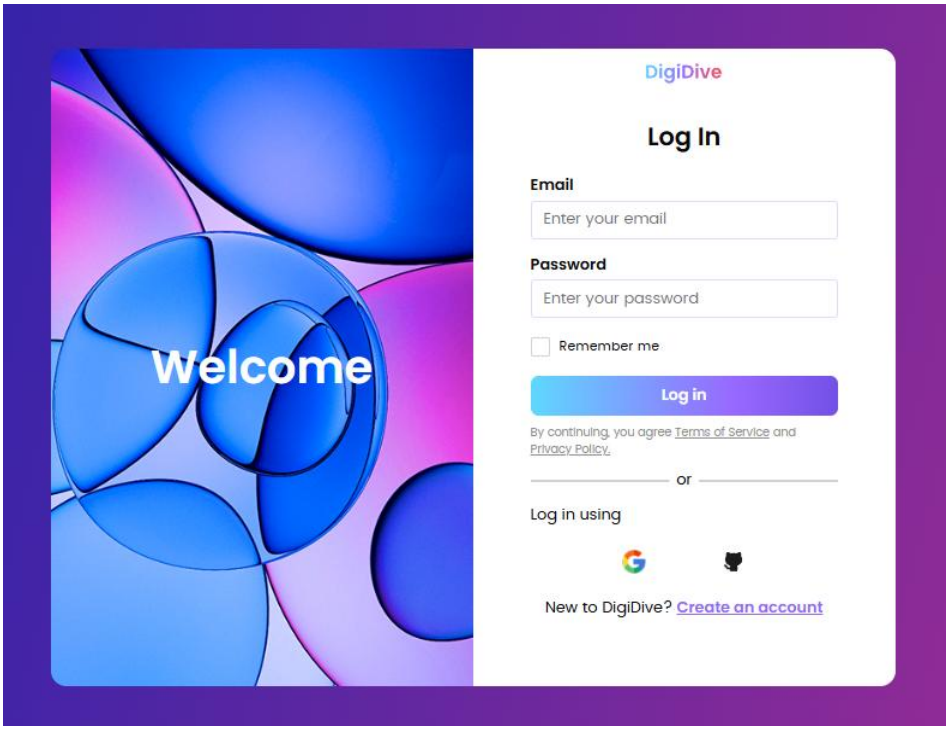

Рисунок 3.4 – Форма авторизації інтернет-магазину IoT-пристроїв “DigiDive”

Інтерфейс сторінки реєстрації користувача містить набір структурованих полів для створення нового облікового запису, що забезпечує зручний та інтуїтивно зрозумілий процес введення даних. Форма реєстрації забезпечує базову валідацію введених даних на рівні клієнтської частини, що дозволяє мінімізувати кількість помилок під час її заповнення.

На серверній стороні реалізовано перевірку унікальності адрес електронної пошти, що гарантує відсутність дублювання облікових записів в інтернет-магазині “DigiDive”. Якщо виявлено помилки, тоді користувачеві відображатимуться відповідні повідомлення, які спрощують процес створення нового облікового запису.

Додатково сторінка реєстрації також підтримує можливість швидкої автентифікації через зовнішні OAuth-провайдери, зокрема Google та GitHub, що дозволяє створювати обліковий запис без необхідності ручного заповнення всіх полів форми. Цей процес допомагає підвищити зручність використання вебплатформи та скорочує час реєстрації нових користувачів. На рисунку 3.5 представлено форму реєстрації інтернет-магазину IoT-пристроїв “DigiDive”.

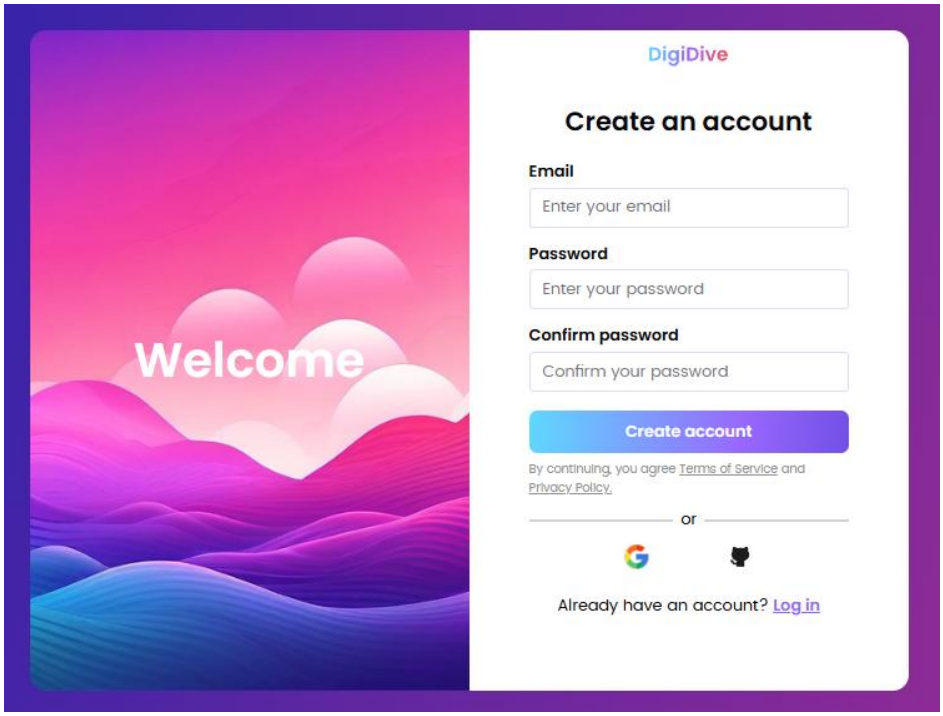

Рисунок 3.5 – Форма реєстрації інтернет-магазину IoT-пристроїв “DigiDive”

Після успішного завершення процесу авторизації або реєстрації користувач перенаправляється на головну сторінку інтернет-магазину, що забезпечує доступ до персоналізованого функціоналу.

Користувачі можуть обирати категорію продуктів у навігаційному меню вебплатформи. Після вибору відповідної категорії здійснюється перехід на сторінку каталогу товарів, яка містить перелік доступних IoT-пристроїв.

Сторінка каталогу продуктів представлена у вигляді адаптивної сітки карток товарів, доповненої інтерактивною бічною панеллю фільтрації, яка дозволяє легко знаходити та сортувати товари за різними параметрами. Крім того, користувач може переглядати основну інформацію про товар без необхідності переходу до сторінки його детального опису.

Ще однією реалізованою функцією інтерфейсу є алгоритм динамічного відображення брендів, який автоматично приховує бренди, що не мають активних товарів у межах поточної категорії, що підвищує релевантність відображуваного контенту. На рисунку 3.6 представлено інтерфейс сторінки каталогу товарів інтернет-магазину IoT-пристроїв ‘DigiDive”.

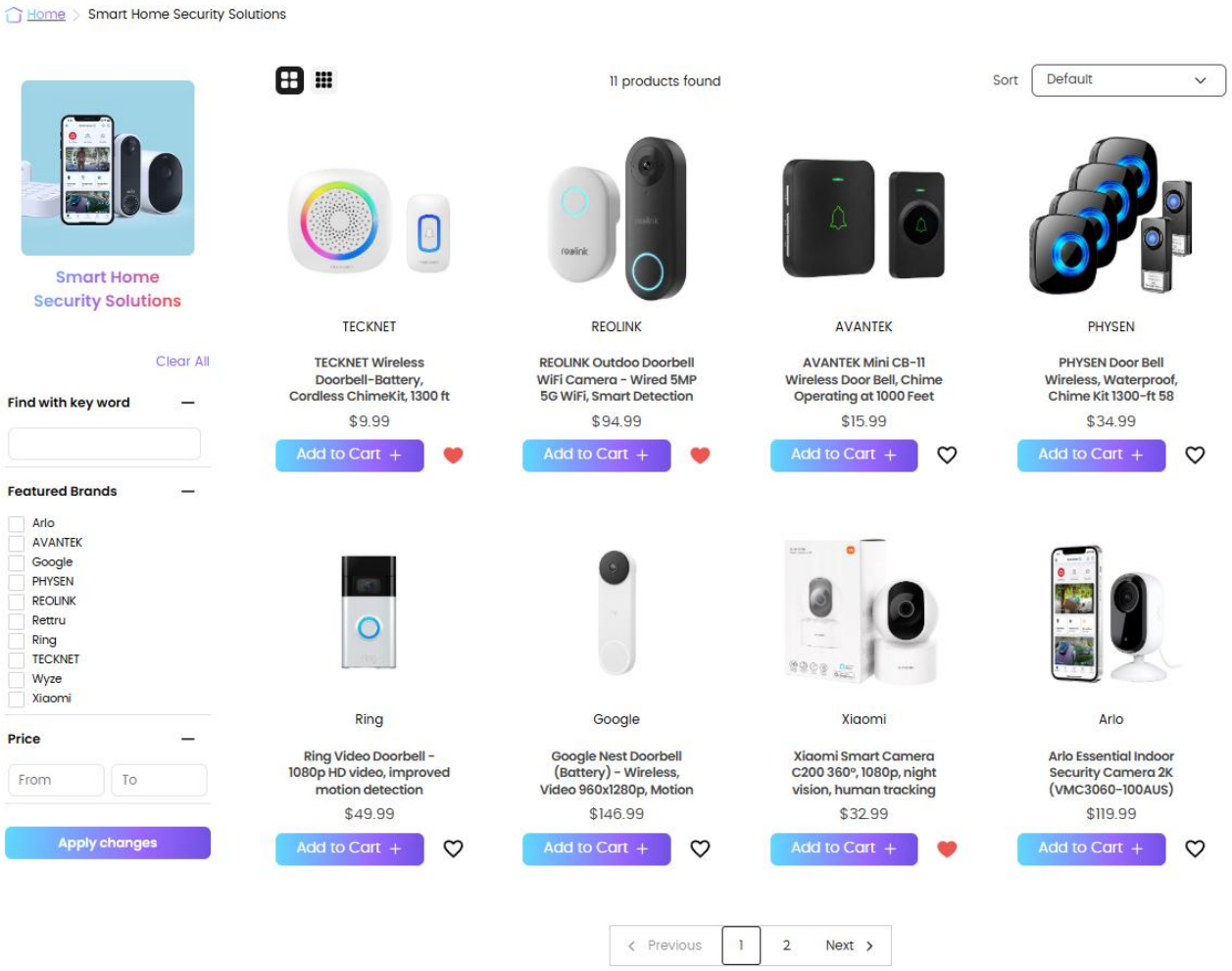

Рисунок 3.6 – Інтерфейс сторінки каталогу товарів інтернет-магазину IoT-пристроїв “DigiDive”

При виборі певного товару користувач потрапляє на сторінку з детальною інформацією про нього. Ця сторінка містить повну інформацію про вибраний пристрій, включаючи детальний текстовий опис, актуальну ціну, термін офіційної гарантії (у місяцях).

Крім того, на сторінці є блок рекомендацій, який відображає список подібних товарів, що відповідають вибраному продукту за ключовими характеристиками або категорією, що сприяє розширенню вибору користувача та покращує навігацію по асортименту. На рисунку 3.7 представлено інтерфейс сторінки конкретного товару в інтернет-магазині IoT-пристроїв “DigiDive”.

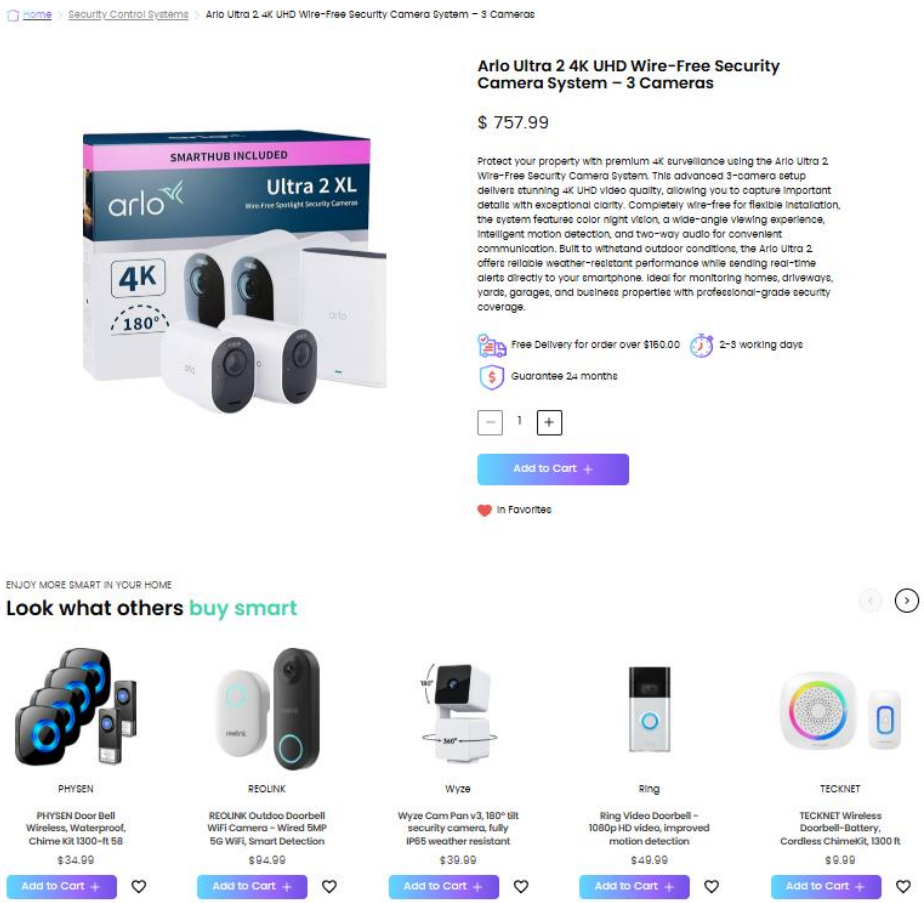

Рисунок 3.7 – Інтерфейс сторінки конкретного продукту в інтернет-магазині IoT-пристроїв “DigiDive”

Користувач може додати пристрій до кошика або зберегти його у списку обраних, що дозволяє створити персоналізований список товарів для подальшого перегляду або придбання.

На сторінці кошика відображається список товарів, вибраних для придбання, з можливістю динамічної зміни їх кількості. Сторінка автоматично перераховує загальну вартість замовлення та відображає вартість доставки. Це надає користувачеві актуальну та узгоджену інформацію про загальну ціну продажу. Такий підхід підвищує зручність взаємодії з інтерфейсом та дозволяє ефективно відстежувати наявність товарів та вартість замовлення. На рисунку 3.8 представлено інтерфейс сторінки кошика в інтернет-магазині IoT-пристроїв “DigiDive”.

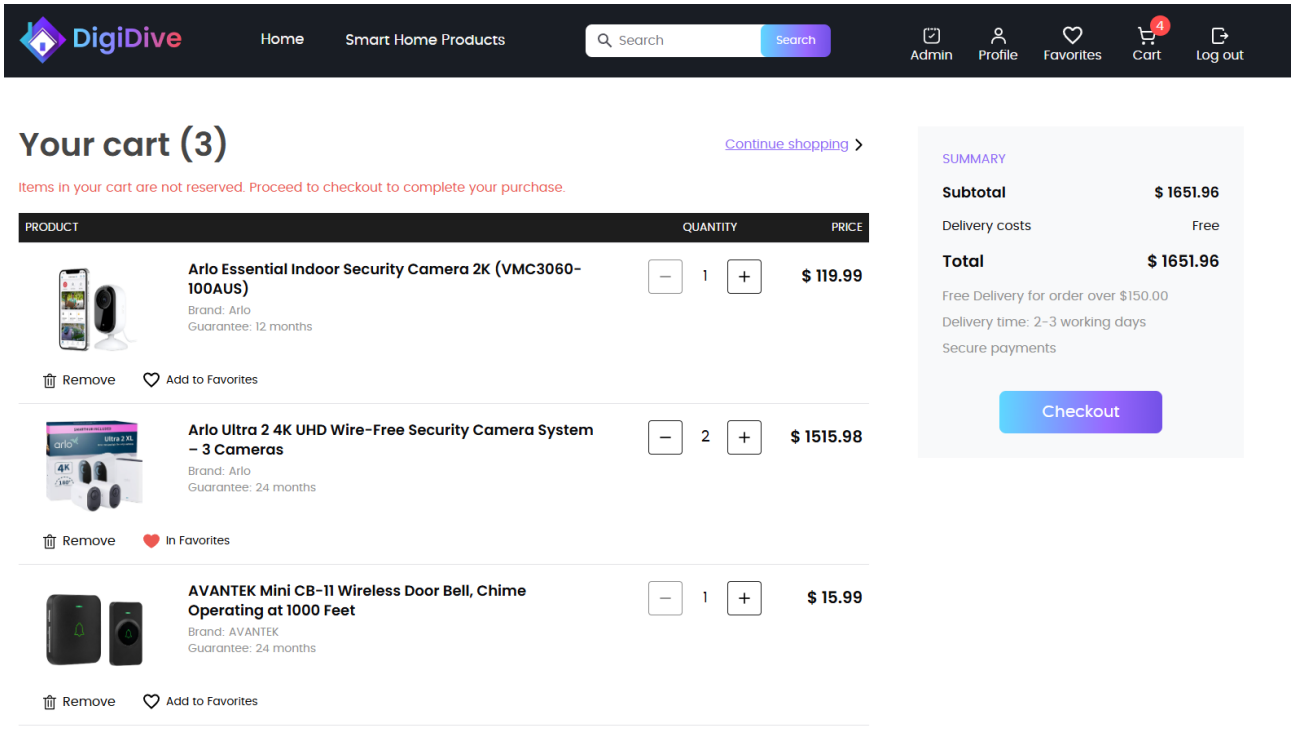

Рисунок 3.8 – Інтерфейс сторінки кошика в інтернет-магазині IoT-пристроїв “DigiDive”

Окрім можливості добавлення товару в кошик, користувач має доступ до персонального списку своїх улюблених товарів. Сторінка списку улюблених товарів дозволяє авторизованому користувачу зберігати свої улюблені пристрої в особистих закладках для швидкого доступу до них в наступних сесіях без необхідності повторного пошуку в каталозі.

Користувач може переглядати збережені товари, переходити на сторінку їх детального перегляду, а також за потреби переносити вибрані товари до кошика для подальшого оформлення замовлення. Це забезпечує більш ефективну взаємодію з каталогом продукції та спрощує процес прийняття рішення про покупку.

Такий підхід сприяє підвищенню зручності використання інтернет-магазину “DigiDive” та сприяє формуванню індивідуальних вподобань користувача. На рисунку 3.9 представлено інтерфейс сторінки списку улюблених товарів в інтернет-магазині IoT-пристроїв “DigiDive”.

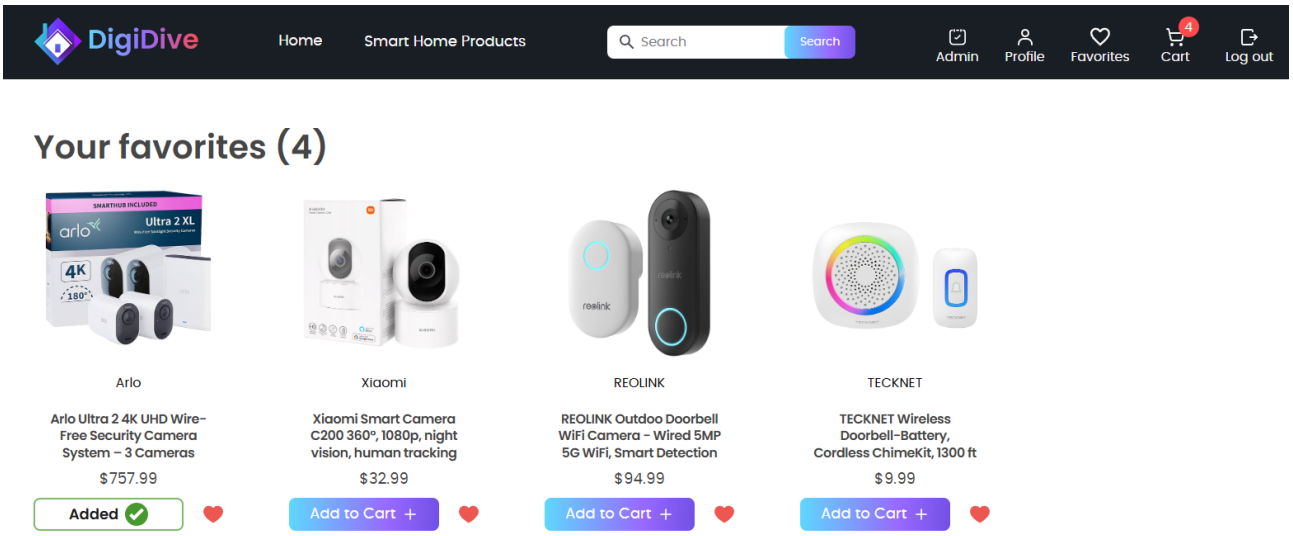

Рисунок 3.9 – Інтерфейс сторінки списку улюблених товарів в інтернет-магазині IoT-пристроїв “DigiDive”

Збережені товари на сторінці списку улюблених синхронізуються з базою даних за допомогою Prisma ORM, що забезпечує їх збереження та доступність незалежно від пристрою користувача.

Після натискання на кнопку “Checkout” (див. рисунок 3.8) відкривається сторінка оформлення замовлення. Ця сторінка містить форму для введення контактної інформації та адреси доставки, включаючи: ПІБ, номер телефону, назва компанії (за наявності), поштовий індекс, місто та повну адресу доставки. Перед підтвердженням замовлення здійснюється перевірка коректності введених даних, що дозволяє зменшити кількість помилок при обробці доставки. Користувачам також відображається зведена інформація про замовлення, включаючи список вибраних товарів та їх остаточну ціну.

Крім того, користувач може зберегти введені дані для подальшого використання, що полегшує повторне оформлення замовлень. Всі введені дані передаються на серверну частину для подальшої обробки та створення замовлення в базі даних. На рисунку 3.10 представлено інтерфейс сторінки оформлення замовлення в інтернет-магазині IoT-пристроїв “DigiDive”.

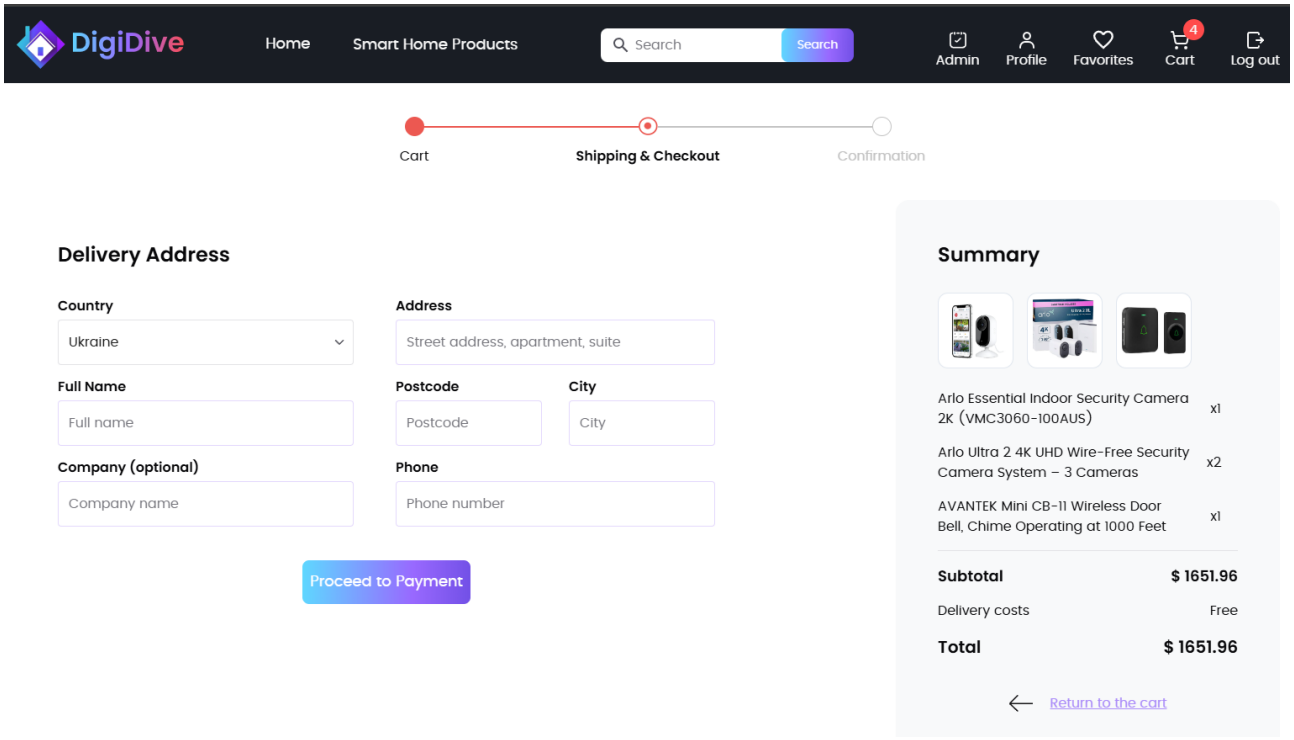

Рисунок 3.10 – Інтерфейс сторінки оформлення замовлення в інтернет-магазині IoT-пристроїв “DigiDive”

Після підтвердження форми на сторінці оформлення замовлення створюється замовлення зі статусом “PENDING”, після чого користувач перенаправляється до платіжного шлюзу Stripe для здійснення оплати. У разі успішної транзакції статус замовлення автоматично змінюється на “PAID”, що підтверджує завершення процесу оплати.

Далі керування подальшим обробленням замовлення переходить до адміністративної частини інтернет-магазину “DigiDive”. Головний екран адміністративної панелі виконує роль аналітичного центру, у якому у вигляді графіків та інформаційних карток відображаються ключові показники інтернет-магазину, такі як: статистика продажів, загальний дохід, кількість зареєстрованих користувачів та динаміка надходження нових замовлень. На рисунку 3.11 представлено інтерфейс сторінки статистики в адміністративній панелі інтернет-магазину IoT-пристроїв “DigiDive”.

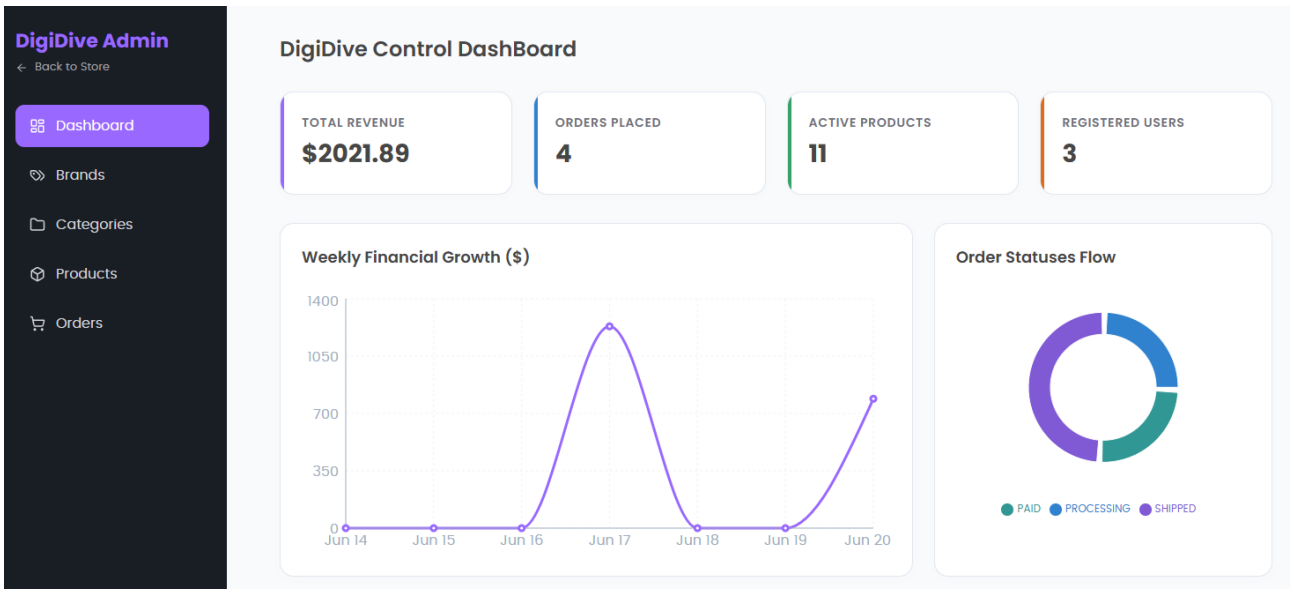

Рисунок 3.11 – Інтерфейс сторінки статистики в адміністративній панелі інтернет-магазину IoT-пристроїв “DigiDive”

Для керування брендами у адміністративній панелі передбачено окремий функціональний розділ. Сторінка керування брендами надає інтерфейс для виконання CRUD-операцій над виробниками продуктів, зокрема додавання нових брендів, редагування їх назв та видалення існуючих записів. Це дозволяє підтримувати актуальність даних про виробників та гнучко керувати IoT-пристроями, що надаються в інтернет-магазині “DigiDive”. На рисунку 3.12 представлено інтерфейс сторінки керування брендами в адміністративній панелі інтернет-магазину IoT-пристроїв “DigiDive”.

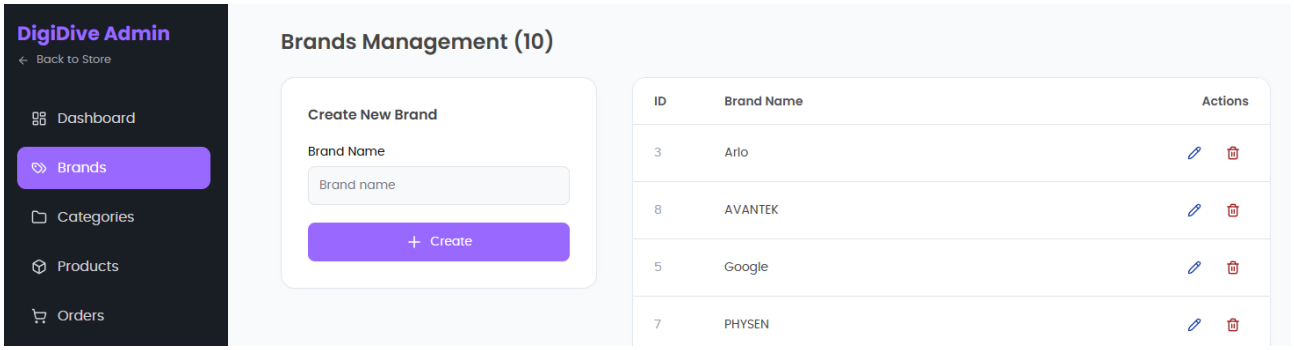

Рисунок 3.12 – Інтерфейс сторінки керування брендами в адміністративній панелі інтернет-магазину IoT-пристроїв “DigiDive”

Зміни, внесені через інтерфейс сторінки керування брендами миттєво відображаються в логіці клієнтських фільтрів, забезпечуючи актуальність даних на стороні користувача.

Аналогічний функціонал реалізовано для керування категоріями. Сторінка керування категоріями дозволяє адміністратору створювати нові розділи каталогу, завантажувати для них графічні зображення (медіа-обкладинки) та керувати рівнем вкладеності шляхом прив'язки підкатегорій до відповідних батьківських елементів деревоподібної структури. Це забезпечує структуровану організацію каталогу та зручність навігації по асортименту. На рисунку 3.13 представлено інтерфейс сторінки керування категоріями в адміністративній панелі інтернет-магазину IoT-пристроїв "DigiDive".

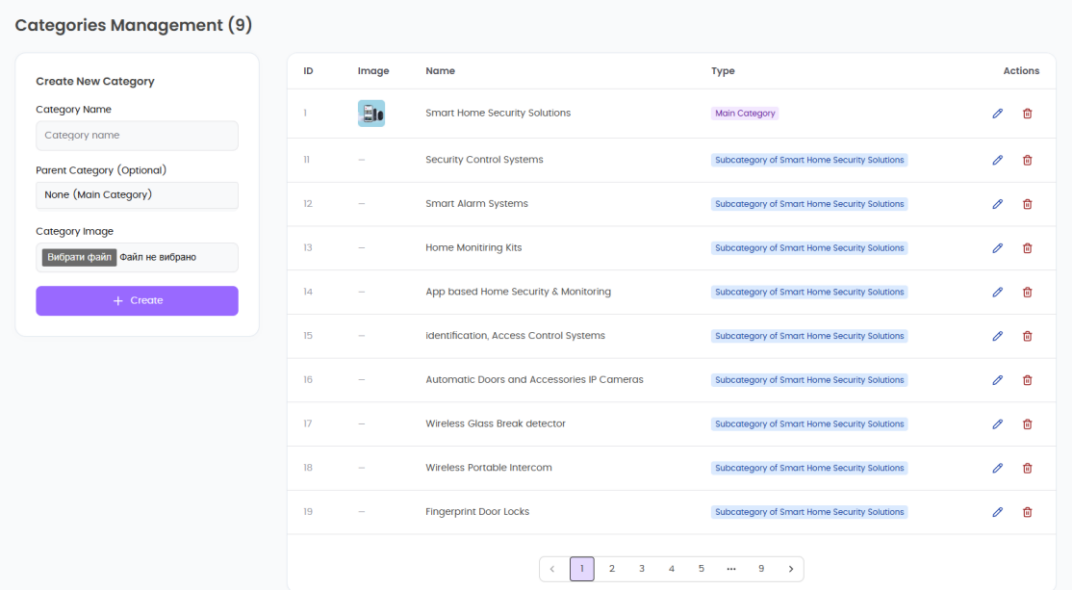

Рисунок 3.13 – Інтерфейс сторінки керування категоріями в адміністративній панелі інтернет-магазину IoT-пристроїв "DigiDive"

Основним елементом адміністрування інтернет-магазину "DigiDive" є керування продуктами. Сторінка керування продуктами містить інтерактивну таблицю з повним переліком IoT-товарів, що дозволяє адміністратору швидко змінювати ціни, коригувати залишки на складі та керувати параметром

“isActive” для архівування товарів. На рисунку 3.14 представлено інтерфейс сторінки керування продуктами в адміністративній панелі інтернет-магазину IoT-пристроїв “DigiDive”.

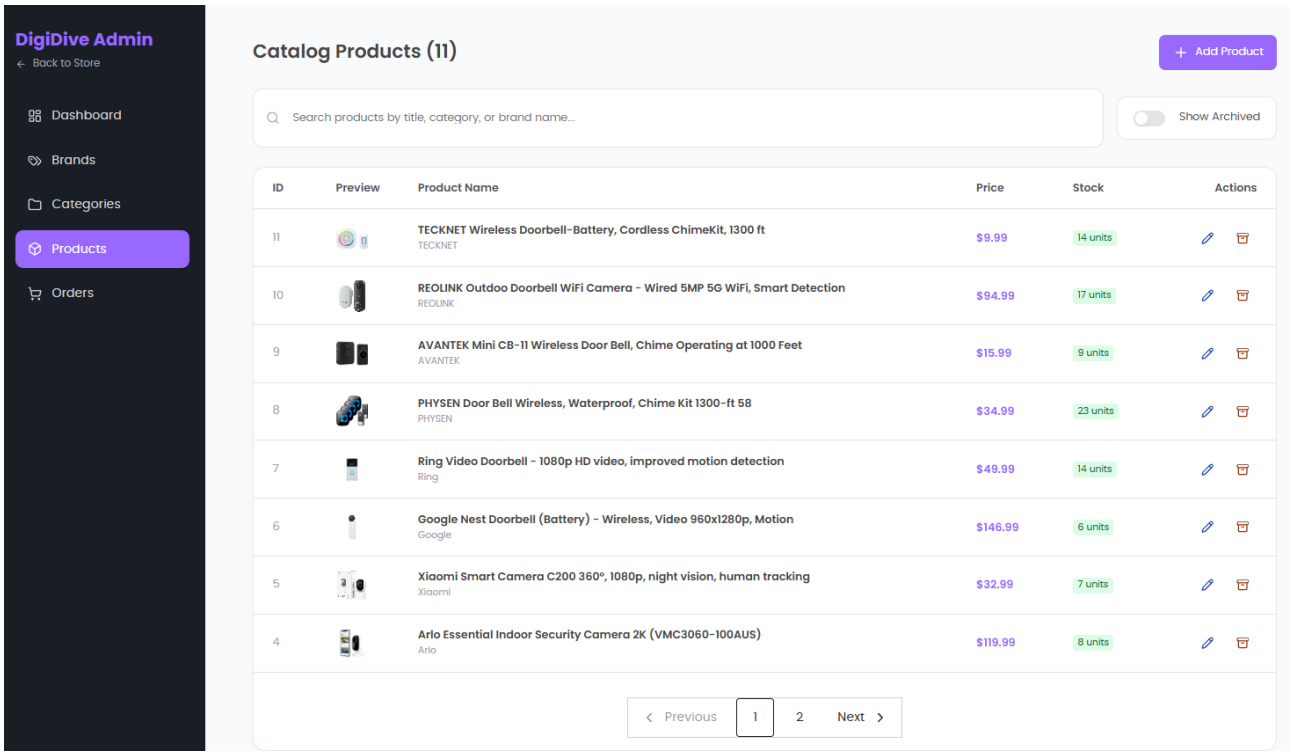

Рисунок 3.14 – Інтерфейс сторінки керування продуктами в адміністративній панелі інтернет-магазину IoT-пристроїв “DigiDive”

Інтерфейс сторінки керування продуктами виключає можливість випадкового видалення важливих даних з бази даних, що підвищує рівень безпеки та цілісності інформації.

Для супроводу та обробки замовлень реалізовано функціонал керування замовленнями. Сторінка керування замовленнями дозволяє адміністратору відстежувати покупки, перевіряти оплату та контактну інформацію клієнтів, переглядати детальну інформацію про конкретне замовлення та змінювати його статус зі статусу обробки на успішну доставку. На рисунку 3.15 представлено інтерфейс сторінки керування замовленнями в адміністративній панелі інтернет-магазину IoT-пристроїв “DigiDive”.

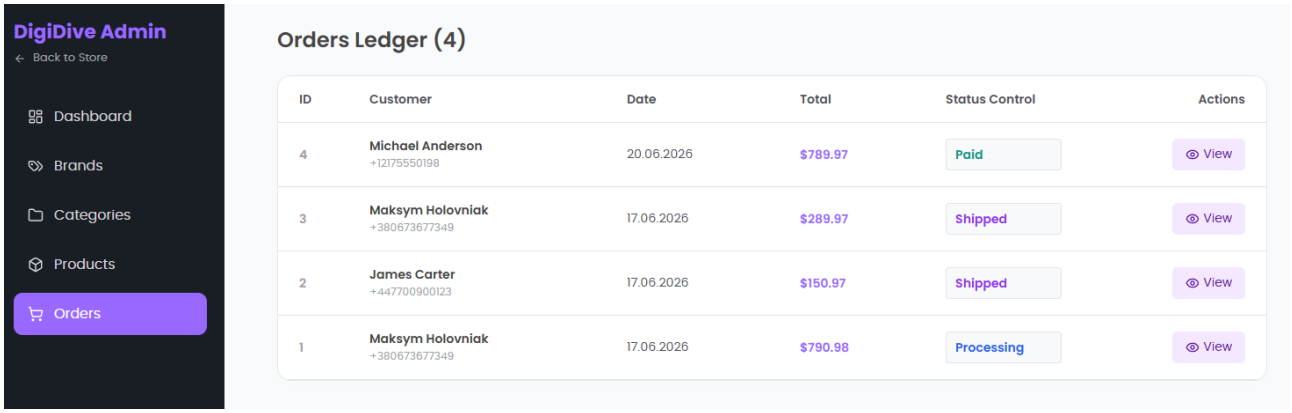

Рисунок 3.15 – Інтерфейс сторінки керування замовленнями в адміністративній панелі інтернет-магазину IoT-пристроїв “DigiDIve”

Аналіз усіх розроблених екранних форм та інтерфейсних рішень засвідчує відповідність інтернет-магазину IoT-пристроїв “DigiDive” архітектурним вимогам проєкту та сучасним інженерним підходам до розробки e-commerce платформ.

Впроваджені інструменти забезпечують високу швидкість взаємодії користувачів з каталогом товарів, а також надають адміністратору централізований та захищений механізм керування комерційними процесами інтернет-магазину IoT-пристроїв “DigiDive”.

3.2 Тестування і верифікація інтернет-магазину “DigiDive”

Важливим етапом життєвого циклу розроблення інформаційної системи є верифікація її працездатності, надійності та відповідність встановленим технічним вимогам. Метою тестування інтернет-магазину “DigiDive” є перевірка коректності реалізації бізнес-логіки, алгоритмів фільтрації, обробки транзакцій та механізмів обмеження прав доступу користувачів.

Для проведення тестування було використано підхід, що поєднує методологію “чорної скриньки” при перевірці функціональності через інтерфейс користувача та компонентне тестування REST API на рівні серверної частини.

Процес верифікації клієнтської частини зосереджувався на перевірці коректності роботи форм введення даних, зокрема форми реєстрації користувача та форми оформлення замовлення. Валідація на стороні клієнта, реалізована за допомогою TypeScript, запобігає надсиланню некоректних запитів на сервер. Під час тестування форми реєстрації перевірялася реакція інтернет-магазину на введення пароля довжиною менше 6 символів, а також некоректного формату електронної пошти. На рисунку 3.16 представлено інтерфейс форми реєстрації під час спрацювання механізмів клієнтської валідації.

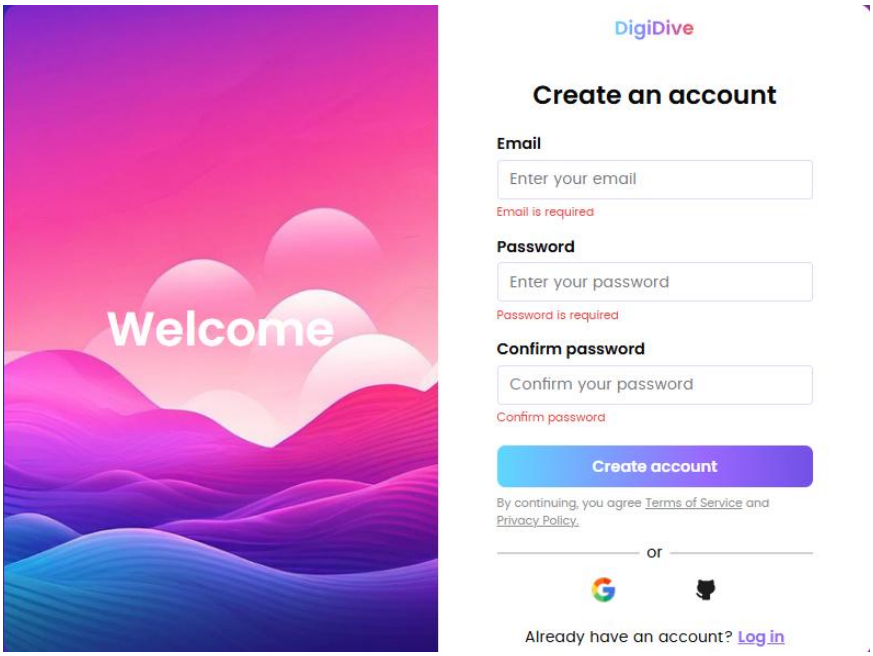

Рисунок 3.16 – Інтерфейс форми реєстрації під час спрацювання механізмів клієнтської валідації

Аналогічне тестування було проведено для форми оформлення замовлення, де важливим є коректне введення логістичних даних для доставки IoT-пристроїв. Результати тестування сторінки оформлення замовлення показали, що якщо робиться спроба надіслати форму з необов'язковим полем, таким як номер телефону або адреса доставки, інтерфейс блокує HTTP-запит і відображає відповідне повідомлення про помилку. На рисунку 3.17

представлено інтерфейс сторінки оформлення замовлення в момент спрацювання механізмів клієнтської валідації.

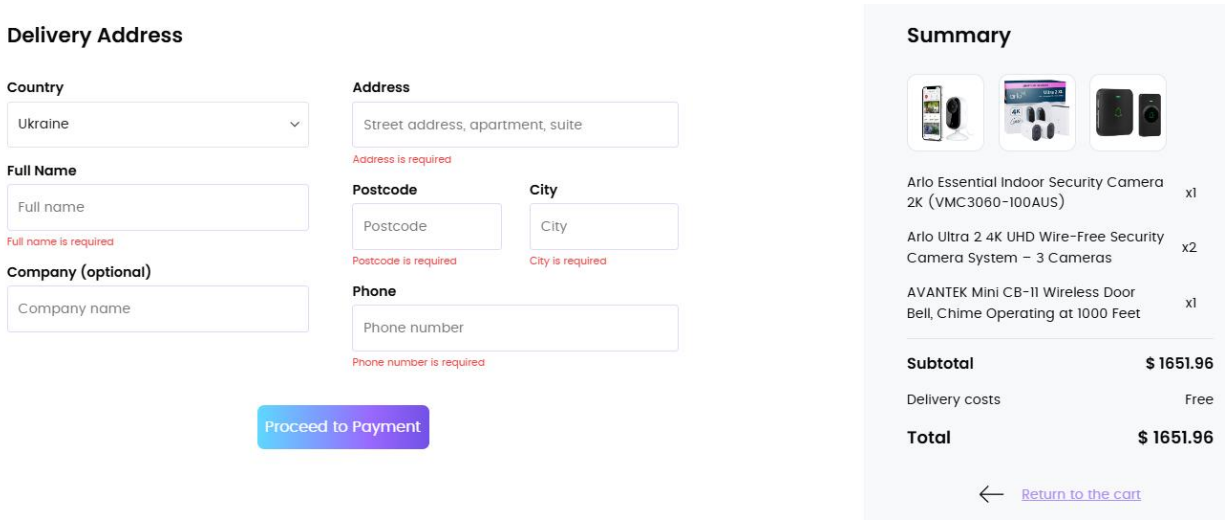

Рисунок 3.17 – Інтерфейс сторінки оформлення замовлення в момент спрацювання механізмів клієнтської валідації

Ще одним важливим аспектом верифікації фронтенд-частини є стабільність маршрутизації на клієнтській стороні, коли користувач намагається вручну ввести недійсні або неіснуючі URL-адреси в браузері. Інтернет-магазин “DigiDive” повинен належним чином обробляти недійсні запити та запобігати перебоям у роботі SPA-додатку.

Під час тестування моделювалися сценарії, в яких замість дійсного маршруту (наприклад, /checkout) вводилося недійсне значення або маршрут (наприклад, /checkoutt/). Реалізований механізм маршрутизації React Router успішно визначив, що відповідний маршрут не існує, та перенаправив користувача на спеціальну сторінку помилки (NotFoundPage). Крім того, було перевірено, чи коректно відображається інтерфейс сторінки помилки на різних роздільних здатностях екрана, що підтвердило адаптивність реалізованого рішення. На рисунку 3.18 представлено інтерфейс сторінки обробки неіснуючих маршрутів.

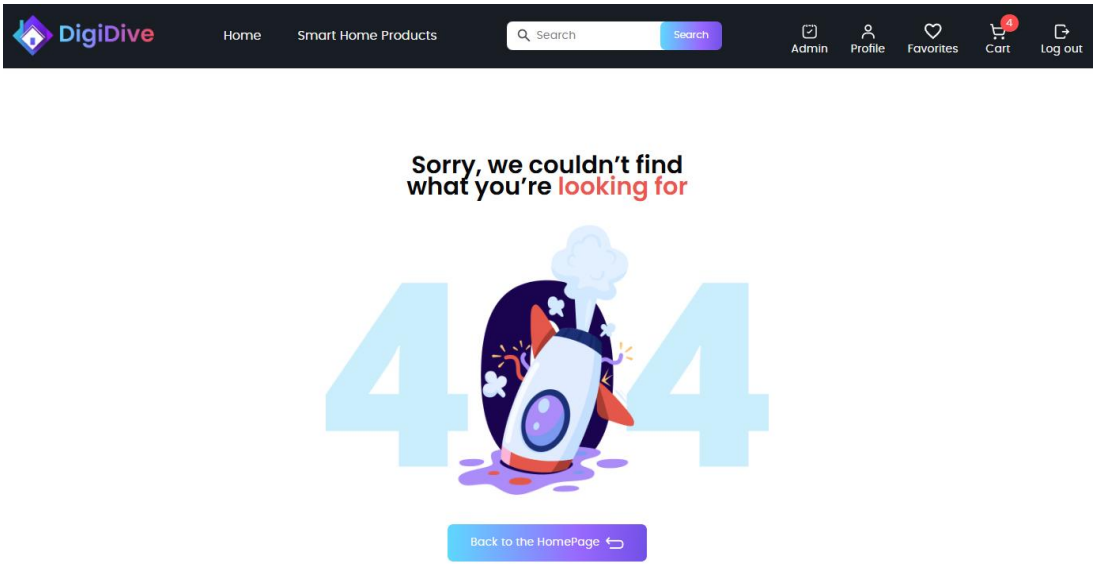

Рисунок 3.18 – Інтерфейс сторінки обробки неіснуючих маршрутів в інтернет-магазині IoT-пристроїв “DigiDive”

Як додаткову перевірку коректності процесу оплати було виконано тестування інтеграції з міжнародним платіжним сервісом Stripe. Після успішного заповнення всієї необхідної інформації на сторінці оформлення замовлення та ініціалізації платіжної сесії на стороні серверної частини NestJS інтернет-магазин автоматично перенаправляє користувача на захищений зовнішній домен платіжного сервісу Stripe. На рисунку 3.19 представлено інтерфейс платіжної сторінки Stripe під час завершення замовлення.

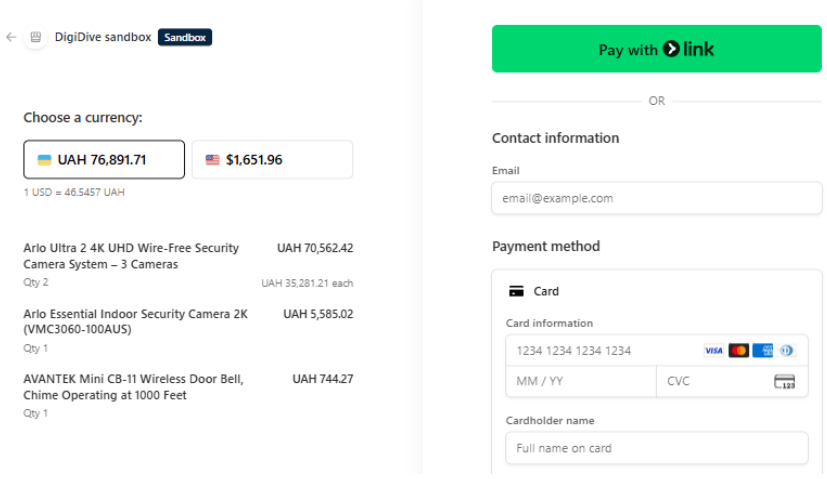

Рисунок 3.19 – Інтерфейс платіжної сторінки Stripe під час завершення замовлення

Завершальним етапом тестування успішного здійснення покупки є перевірка механізму зворотного перенаправлення (callback) після успішного підтвердження транзакції. Якщо від платіжного сервісу отримано позитивну відповідь, користувач автоматично перенаправляється на спеціальну сторінку інтернет-магазину, що свідчить про успішне завершення транзакції. На рисунку 3.20 представлено інтерфейс сторінки успішного підтвердження оплати в інтернет-магазині IoT-пристроїв “DigiDive”.

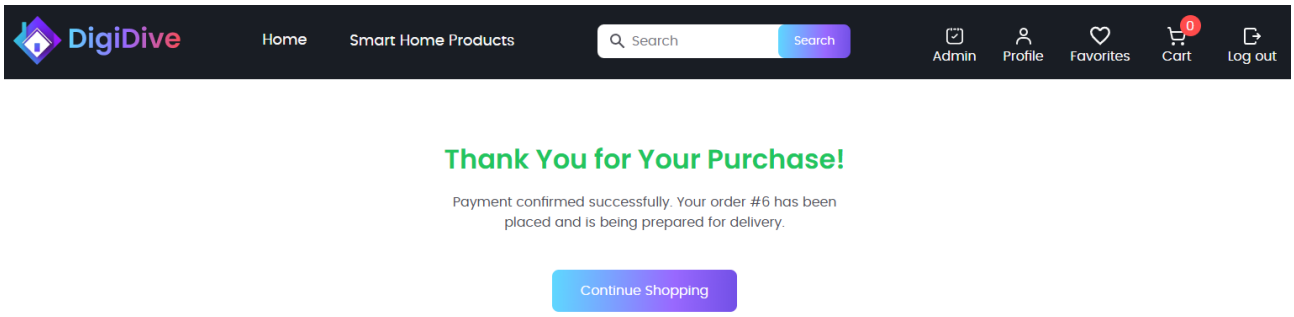

Рисунок 3.20 – Інтерфейс сторінки успішного підтвердження оплати в інтернет-магазині IoT-пристроїв “DigiDive”

Для повноцінної верифікації інтернет-магазину IoT-пристроїв “DigiDive”, окрім клієнтського інтерфейсу, було виконано ізольоване тестування серверної логіки, реалізованої на базі NestJS. Інструмент Postman використовувався для тестування кінцевих точок REST API, коректності обробки DTO, функціональності засобів безпеки та узгодженості швидкості відповідей HTTP. Це було використано для моделювання HTTP-запитів безпосередньо до бекенду, що дозволило протестувати бізнес-логіку незалежно від фронтенду.

Особливу увагу було приділено захищеним API-маршрутам. У випадку надсилання запиту на створення нової категорії або бренду без передачі дійсного JWT-токена в заголовку Authorization серверна частина NestJS коректно відхиляв операцію та повертав відповідний код помилки. Це підтверджує належну реалізацію механізмів авторизації та контролю доступу.

На рисунку 3.21 представлено результат тестування GET-запиту для отримання переліку товарів за допомогою інструменту Postman.

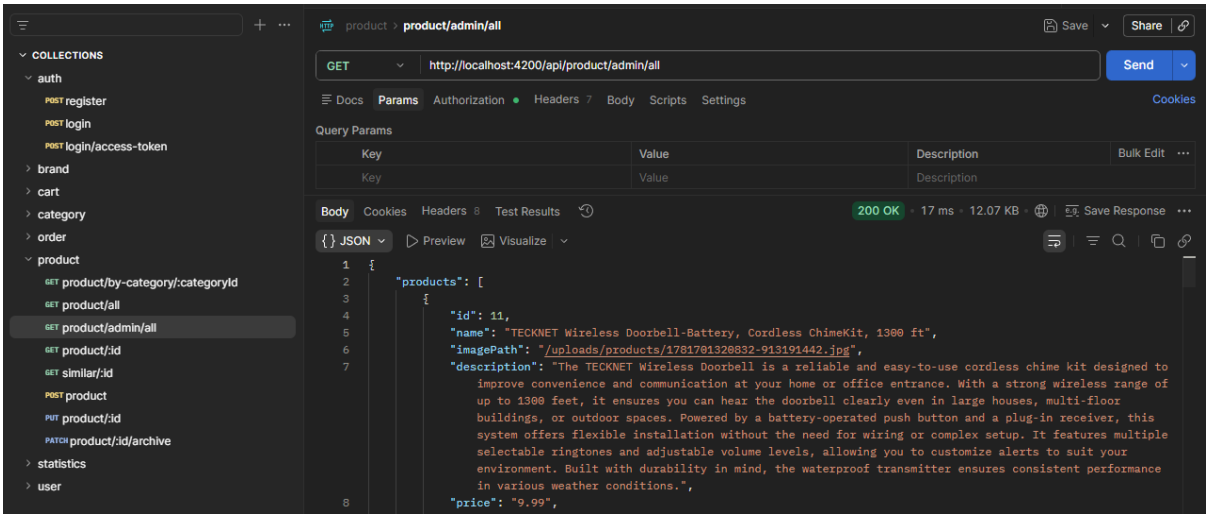

Рисунок 3.21 – Результат тестування GET-запиту для отримання переліку товарів за допомогою інструменту Postman

Аналіз результатів тестування показує, що інтернет-магазин “DigiDive” працює стабільно та коректно. Було визначено, що якщо в процесі автентифікації введено правильні облікові дані, користувач успішно отримує доступ до вебдодатку, генеруючи JWT-токен, тоді як використання неправильного пароля призводить до відмови у наданні HTTP-авторизації.

Отже, проведене функціональне тестування та верифікація підтвердили повну працездатність, логічну зв'язність та технічну готовність інтернет-магазину IoT-пристроїв “DigiDive” до подальшої експлуатації та масштабування.

3.3 План заходів для забезпечення захисту інтернет-магазину “DigiDive”

Забезпечення повного захисту інформаційних ресурсів інтернет-магазину “DigiDive” є пріоритетним завданням, спрямованим на запобігання фінансовим та репутаційним втратам. Для цього реалізовано багаторівневий підхід до

безпеки, який включає захист каналів зв’язку, автентифікацію користувачів, безпеку серверної логіки та цілісність даних на рівні СУБД.

Перший рівень безпеки забезпечується шляхом шифрування даних під час передачі між клієнтською частиною та REST API. Весь мережевий трафік передається через захищений протокол HTTPS з використанням сертифікатів TLS, що унеможливлює перехоплення даних сеансу та атаки типу “людина посередині” (Man-in-the-Middle) [19].

Другий рівень безпеки реалізовано за допомогою технології рольової моделі доступу (Role-Based Access Control). Після успішної автентифікації користувач отримує підписаний токен, що містить інформацію про його роль (user або admin), яка перевіряється механізмом безпеки NestJS. Паролі в базі даних MySQL зберігаються виключно у вигляді криптографічних хешів, згенерованих за допомогою алгоритму bcrypt.

Третій рівень безпеки призначений для зниження вразливостей програмного коду. Використання Prisma ORM усуває ризики SQL-ін’єкцій завдяки параметризованим запитам. Крім того, налаштовано політику CORS з обмеженням довірених доменів (origins), а механізми автентифікації на стороні сервера дозволяють фільтрувати потенційно недійсні або шкідливі вхідні дані, що підвищує безпеку типу XSS [20].

Кінцевим рівнем безпеки є забезпечення відмовостійкості даних. У СУБД MySQL виконуються регулярні автоматизовані резервні копії за певним розкладом, що дозволяє швидко відновити базу даних при виникненні збою. Також застосовано каскадні обмеження цілісності даних, що запобігає випадковому порушенню зв’язків між сутностями та втраті транзакційної інформації [21]. Набір реалізованих механізмів безпеки створює надійне та захищене середовище для забезпечення стабільного функціонування інтернет-магазину IoT-пристроїв “DigiDive”.

3.4 Висновок до третього розділу

В третьому розділі кваліфікаційної роботи виконано практичну верифікацію, аналіз експлуатації та безпеки інтернет-магазину IoT-пристроїв “DigiDive”. На цьому етапі було підтверджено, що розроблений вебдодаток відповідає технічним та архітектурним вимогам.

Аналіз інтерфейсу користувача підтвердив високу швидкість навігації та адаптивність розробленого SPA-додатка на базі React і Vite. Опис екранних форм публічної частини та адміністративної панелі підтвердив зручність взаємодії користувачів з каталогом, кошиком та формою оформлення замовлення, а також ефективність реалізованих інструментів керування контентом.

Під час тестування та верифікації інтернет-магазину IoT-пристроїв “DigiDive” з використанням підходу “чорної скриньки" було перевірено надійність механізмів валідації форм реєстрації та оформлення замовлення, а також коректність обробки некоректних URL шляхом перенаправлення на сторінку NotFoundPage. Окреме тестування REST API було проведено за допомогою інструменту Postman, який визначив коректність обробки GET-запиту, зокрема з параметрами пагінації.

Комплекс заходів, реалізованих для захисту інформаційних ресурсів забезпечив належний рівень безпеки інтернет-магазину “DigiDive”. Шифрування трафіку через HTTPS/TLS, виконання рольової автентифікації на основі JWT, хешування паролів за допомогою bcrypt та використання Prisma ORM мінімізують ризики кібератак. Крім того, регулярне резервне копіювання бази даних MySQL дозволяє швидко відновити вебзастосунок у разі збою. Отримані результати підтверджують повну технічну готовність інтернет-магазину IoT-пристроїв “DigiDive” до експлуатації.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Психологічні чинники небезпеки

У сучасних умовах розвитку інформаційних технологій професійна діяльність розробника програмного забезпечення характеризується значним інтелектуальним навантаженням, необхідністю тривалої концентрації уваги та високим рівнем відповідальності за результати роботи. Психологічний стан фахівця є важливим чинником, який впливає на продуктивність праці, якість програмного продукту та безпеку трудової діяльності. Процеси проєктування, розроблення, тестування та супроводу програмних систем вимагають постійної розумової активності, аналізу значних обсягів інформації та швидкого прийняття рішень, що створює додаткове психоемоційне навантаження на працівника.

Одним із найбільш поширених негативних психологічних чинників є інтелектуальний та емоційний стрес. Його виникнення пов’язане з необхідністю розв’язання складних технічних завдань, пошуку та усунення помилок у програмному коді, виконанням робіт в умовах обмеженого часу та дотриманням встановлених термінів реалізації проєкту. Додатковим джерелом психологічного навантаження є відповідальність за працездатність програмної системи, захист даних користувачів та стабільність функціонування окремих компонентів вебплатформи. Тривалий вплив стресових факторів може призводити до розумової втоми, погіршення концентрації уваги, зниження продуктивності праці та підвищення ймовірності виникнення помилок під час виконання професійних обов’язків [35].

Значний вплив на психоемоційний стан розробника також має необхідність тривалої роботи з текстовою інформацією. Аналіз програмного коду, документації, структур баз даних, результатів тестування та журналів виконання програм вимагає постійної концентрації та високої точності сприйняття інформації. У результаті тривалої напруженої роботи можуть

виникати симптоми розумової та зорової втоми, які проявляються зниженням швидкості мислення, погіршенням уважності та збільшенням кількості помилок під час розроблення програмного забезпечення.

Іншим важливим психологічним чинником є монотонність праці. Незважаючи на високий рівень інтелектуальної складності роботи, значна частина професійної діяльності розробника пов’язана з виконанням однотипних операцій. До них належать створення подібних компонентів інтерфейсу, налаштування форм введення даних, перевірка працездатності окремих модулів системи та повторення тестових сценаріїв. Тривале виконання таких завдань може спричиняти зниження рівня уваги, втрату зацікавленості в роботі та погіршення здатності до швидкого реагування на нестандартні ситуації.

Поєднання стресу, монотонності та розумової втоми за відсутності належного відпочинку може призводити до розвитку професійного вигорання. Даний стан характеризується емоційним виснаженням, зниженням мотивації до виконання професійних обов’язків, погіршенням психологічного самопочуття та зменшенням ефективності праці. Професійне вигорання негативно впливає не лише на стан здоров’я працівника, але й на якість створюваного програмного забезпечення, оскільки збільшує ризик виникнення помилок та знижує рівень відповідальності під час виконання робочих завдань [35].

Стійкість до впливу негативних психологічних чинників залежить від індивідуальних особливостей людини, її професійного досвіду, рівня підготовки та загального стану здоров’я. Водночас перевтома, недостатня тривалість сну, емоційне напруження або погіршення самопочуття можуть істотно знижувати працездатність навіть висококваліфікованих спеціалістів. За таких умов підвищується ймовірність прийняття помилкових рішень, недотримання вимог інформаційної безпеки та виникнення помилок під час роботи з програмним кодом або конфігурацією системи.

Для мінімізації негативного впливу психологічних чинників необхідно забезпечити раціональну організацію праці розробника програмного забезпечення. До основних профілактичних заходів належать дотримання

режиму праці та відпочинку, регулярне виконання перерв під час роботи за комп’ютером, оптимізація робочого навантаження та створення сприятливих умов праці [35]. Важливе значення мають ергономічне облаштування робочого місця, належний рівень освітлення, підтримання комфортного мікроклімату приміщення та використання сучасних засобів автоматизації процесу розроблення. Реалізація зазначених заходів сприяє збереженню працездатності працівника, підвищенню ефективності його професійної діяльності та зниженню ризику виникнення помилок під час розроблення програмного забезпечення.

4.2 Загальні вимоги безпеки з охорони праці для користувачів ПК

Забезпечення безпечних і нешкідливих умов праці під час роботи з персональними комп’ютерами є одним із важливих завдань охорони праці в галузі інформаційних технологій. Нормативно-правова база у цій сфері ґрунтується на державних санітарних нормах, правилах та стандартах, які регламентують вимоги до організації робочих місць, параметрів виробничого середовища та режимів експлуатації комп’ютерної техніки. Дотримання встановлених вимог сприяє мінімізації впливу шкідливих виробничих факторів, зокрема тривалого статичного навантаження, зорового перенапруження, несприятливих параметрів мікроклімату та недостатнього освітлення робочих приміщень.

До виконання робіт із використанням персонального комп’ютера допускаються особи, які пройшли відповідний інструктаж з охорони праці, ознайомлені з правилами пожежної безпеки та порядком надання домедичної допомоги. Працівник зобов’язаний дотримуватися вимог безпечної експлуатації обладнання, правил внутрішнього трудового розпорядку та інструкцій щодо використання технічних засобів [36]. У разі виявлення несправностей обладнання, пошкодження електропроводки або інших небезпечних ситуацій

необхідно негайно повідомити відповідальну особу та припинити роботу до усунення виявлених недоліків.

Організація робочого місця користувача персонального комп'ютера повинна відповідати ергономічним вимогам. Робочий стіл і крісло мають забезпечувати підтримання правильної робочої пози та можливість регулювання основних параметрів відповідно до індивідуальних особливостей працівника. Монітор рекомендується розташовувати на відстані 50–70 см від очей користувача, а його верхню межу – на рівні очей або дещо нижче. Таке розташування обладнання сприяє зменшенню навантаження на органи зору та опорно-руховий апарат [36]. Клавіатура і комп'ютерна миша повинні розміщуватися таким чином, щоб забезпечувати природне положення кистей рук і передпліч під час роботи.

Важливими умовами безпечної праці є належне освітлення та оптимальні параметри мікроклімату приміщення. Робочі місця слід розташовувати таким чином, щоб уникати потрапляння прямих сонячних променів та появи відблисків на екранах моніторів. Штучне освітлення має бути рівномірним і достатнім для виконання робіт з екранними пристроями. Температура повітря, відносна вологість та швидкість його руху повинні відповідати встановленим санітарним нормам, що забезпечує комфортні умови праці та сприяє підтриманню працездатності працівників протягом робочого дня.

Особливе значення під час експлуатації комп'ютерної техніки має дотримання вимог електробезпеки. Усе обладнання повинно підключатися до справної електромережі із захисним заземленням. Забороняється самостійно виконувати ремонт електрообладнання, використовувати несправні пристрої або працювати з пошкодженими кабелями живлення [36]. Також не допускається захаращення робочого місця сторонніми предметами, які можуть перешкоджати вентиляції обладнання та підвищувати ризик його перегрівання.

Важливим елементом профілактики професійної втоми є раціональний режим праці та відпочинку. Для зменшення негативного впливу тривалої роботи за комп'ютером рекомендується робити регламентовані перерви через

певні проміжки часу, змінювати характер діяльності та виконувати вправи для очей і м’язів опорно-рухового апарату. Дотримання зазначених вимог сприяє збереженню працездатності працівників, зниженню ризику виникнення професійних захворювань і створенню безпечних умов праці під час виконання робіт у сфері інформаційних технологій.

4.3 Висновок до четвертого розділу

В четвертому розділі кваліфікаційної роботи виконано комплексний аналіз питань охорони праці та безпеки життєдіяльності розробника програмного забезпечення в процесі створення сучасних вебзастосунків. Розглянуто основні фактори виробничого середовища, що впливають на працездатність фахівця, а також визначено організаційно-технічні заходи, спрямовані на формування безпечних і комфортних умов праці.

У межах дослідження психологічних чинників небезпеки проаналізовано вплив когнітивного навантаження на ефективність роботи розробника. Встановлено, що специфіка програмної інженерії, яка передбачає тривалу концентрацію уваги, виконання однотипних операцій та роботу в умовах обмежених термінів, може призводити до накопичення розумової та зорової втоми, що підвищує ризик професійного вигорання та виникнення інженерних помилок.

Окрему увагу приділено загальним вимогам охорони праці для користувачів персональних комп’ютерів. Сформульовано основні санітарно-гігієнічні та ергономічні вимоги до організації робочого місця, зокрема правильне розміщення монітора, клавіатури та миші, а також забезпечення належного рівня освітлення й оптимальних параметрів мікроклімату в приміщенні. Підкреслено важливість дотримання правил електробезпеки та впровадження регламентованого режиму праці й відпочинку з обов’язковими перервами.

ВИСНОВКИ

У кваліфікаційній роботі освітнього рівня «Бакалавр» здійснено теоретичне обґрунтування, проєктування, програмну реалізацію та верифікацію інтернет-магазину IoT-пристроїв “DigiDive”. Дослідницькі та інженерні рішення, виконані в рамках проекту, дозволили досягти поставлених цілей та забезпечили стабільну роботу вебплатформи.

У першому розділі подано аналітичний огляд предметної області Інтернету речей, виявлено специфічні виклики електронної комерції, пов’язані з високою технологічною складністю та ієрархічною структурою продукту. Розглянуто архітектуру взаємодії акторів системи, включаючи гостя, авторизованого користувача та адміністратора, для яких побудовано UML-діаграму варіантів використання. Також визначено повний перелік функціональних, нефункціональних та технічних вимог до інтернет-магазину. На завершення обґрунтовано вибір сучасного технологічного стеку, який включає TypeScript, React, Chakra UI, Redux Toolkit та RTK Query на стороні клієнта, а також NestJS, Prisma ORM та MySQL на серверній частині.

У другому розділі досліджено та змодельовано трирівневу архітектуру вебзастосунку, що забезпечує чітке розділення клієнтського інтерфейсу та серверної бізнес-логіки. Обґрунтовано перелік інформаційних сутностей, визначено способи їх реляційного зберігання та спроєктовано концептуальну модель даних у вигляді ER-діаграми. На основі цього розроблено логічну структуру серверної частини на базі фреймворку NestJS з використанням модульного підходу, DTO-валідації та механізмів безпеки, а також реалізовано компонентну структуру клієнтського SPA на базі React.

У третьому розділі розроблено та впроваджено користувацький інтерфейс публічних сторінок та адміністративної панелі з використанням Chakra UI. Інтернет-магазин IoT-пристроїв “DigiDive” був протестований за допомогою підходу “чорної скриньки” через графічний інтерфейс, що підтвердило коректність обробки форм, маршрутизації та обробки недійсних URL-адрес. За

допомогою Postman верифіковано роботу REST API NestJS, зокрема GET-запитів із параметрами пагінації. Крім того, запропоновано набір заходів захисту інформаційних ресурсів платформи, що включає HTTPS, JWT-автентифікацію, хешування паролів за допомогою Bcrypt, інтеграцію платіжного сервісу Stripe та резервне копіювання бази даних MySQL.

У розділі «Безпека життєдіяльності, основи охорони праці» розглянуто психологічні чинники професійної діяльності розробника програмного забезпечення, включаючи аналіз впливу інтелектуального навантаження, монотонності праці та розумової втоми на ризик виникнення помилок. Також визначено загальні вимоги охорони праці для користувачів персональних комп’ютерів, що базуються на державних санітарних нормах, правилах ергономіки робочого місця, оптимальних параметрах мікроклімату, освітлення, електробезпеки та раціонального режиму праці й відпочинку, що в сукупності забезпечує збереження здоров’я фахівця та стабільність його діяльності.

ПЕРЕЛІК ДЖЕРЕЛ

1 Аналіз сучасного ринку IoT-пристроїв та основних тенденцій його розвитку [Електронний ресурс]: Режим доступу: https://www.wonderfulpcb.com/uk/blog/internet-of-things-iot-impact-trends-uses-security-guide/.

2 Використання Інтернету речей (IoT) у “розумних” будівлях [Електронний ресурс]: Режим доступу: https://micronet.com.ua/internet-rechej-iot-u-rozumnyh-budivlyah-majbutnye-vzhe-tut/.

3 Застосування та переваги Інтернету речей (IoT) у бізнесі [Електронний ресурс]: Режим доступу: https://seo-evolution.com.ua/blog/poleznye-sovety/iot-v-biznesi-zastosuvannya-ta-perevagi.

4 Основні етапи розроблення інтернет-магазину [Електронний ресурс]: Режим доступу: https://wezom.com.ua/ua/blog/etapy-razrabotki-internet-magazina.

5 Проєктування та організація інтернет-магазину [Електронний ресурс]: Режим доступу: https://wezom.com.ua/ua/blog/struktura-internet-magazina-klyuchevye-momenty-sozdaniya.

6 Критерії вибору технологічного стеку для розроблення сучасних вебзастосунків [Електронний ресурс]: Режим доступу: https://wezom.com.ua/ua/blog/tehnologicheskij-stek-proekta.

7 RTK Query як інструмент оптимізації роботи з даними в React-застосунках [Електронний ресурс]: Режим доступу: https://blog.ithillel.ua/articles/rtk-query.

8 Система керування базами даних MySQL: огляд та переваги використання [Електронний ресурс]: Режим доступу: https://lemon.school/blog/systema-upravlinnya-bazamy-danyh-mysql.

9 Розробка веб-застосунку: від ідеї до запуску [Електронний ресурс]: Режим доступу: https://avada-media.ua/blog/how-a-web-interface-is-created-from-idea-to-launch/#veb-interfeys-yak-vzayemodiya-z-brendom.

10 Офіційна документація Prisma ORM [Електронний ресурс]: Режим доступу: https://www.prisma.io/docs.

11 Підходи до оптимізації вебзастосунків та інструменти підвищення продуктивності [Електронний ресурс]: Режим доступу: https://dou.ua/forums/topic/43011/.

12 ER-діаграми та їх застосування в проєктуванні баз даних [Електронний ресурс]: Режим доступу: https://happymonday.ua/er-diahrama.

13 Переваги та особливості використання TypeScript у фронтенд-розробці [Електронний ресурс]: Режим доступу: https://itproger.com/ua/news/typescript-pochemu-etot-yazik-stanovitsya-standartom.

14 Офіційна документація фреймворку NestJS [Електронний ресурс]: Режим доступу: https://docs.nestjs.com/.

15 Побудова та масштабування цифрових бізнес-платформ [Електронний ресурс]: Режим доступу: https://wezom.com.ua/ua/blog/platforma-dlya-biznesu.

16 Офіційна документація інструменту збірки фронтенд-застосунків Vite [Електронний ресурс]: Режим доступу: https://vite.dev/.

17 Механізми кешування та їх оптимізація у вебсистемах [Електронний ресурс]: Режим доступу: https://www.hostragons.com/uk/що-таке-кеш-кешування-веб-сайту/.

18 Принципи UX-дизайну в сучасній розробці інтерфейсів [Електронний ресурс]: Режим доступу: https://wezom.com.ua/ua/blog/principy-ux-dizajna.

19 REST Security Cheat Sheet: рекомендації з безпеки REST API [Електронний ресурс]: Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html.

20 OWASP Top 10: основні вразливості вебзастосунків [Електронний ресурс]: Режим доступу: https://research.kr-labs.com.ua/owasp-top-10-vulberabilities/.

21 Забезпечення кібербезпеки в система електронної комерції та методи захисту даних [Електронний ресурс]: Режим доступу:

https://wezom.com.ua/ua/blog/kiberbezpeka-v-proektah-ecommerce-kompleksniy-gayd.

22 Офіційна документація бібліотеки React [Електронний ресурс]: Режим доступу: https://react.dev/reference/react.

23 Порівняльний аналіз систем керування базами даних [Електронний ресурс]: Режим доступу: https://data-b-i.com/uk/article/porivnyannya-subd-mysql-postgresql-mssqlserver.html.

24 Офіційна документація бібліотеки Redux Toolkit [Електронний ресурс]: Режим доступу: https://redux-toolkit.js.org/introduction/getting-started.

25 Аналіз та огляд платіжної системи Stripe для інтернет-магазинів [Електронний ресурс]: Режим доступу: https://wezom.com.ua/ua/blog/stripe-priyom-onlayn-platezhiv-na-sayti-cherez-stripe.

26 Принципи побудови архітектури REST API та їх застосування у вебзастосунках [Електронний ресурс]: Режим доступу: https://foxminded.ua/shcho-take-rest-api/.

27 Застосування JWT для автентифікації у вебзастосунках [Електронний ресурс]: Режим доступу: https://devzone.org.ua/post/iak-vykorystovuvaty-json-web-tokens-jwt-dlia-avtentyfikatsiyi.

28 Огляд трирівневої архітектури вебзастосунків та принципи її побудови [Електронний ресурс]: Режим доступу: https://www.vpnunlimited.com/ua/help/cybersecurity/3-tier-architecture.

29 Методологія тестування “чорної скриньки” у програмній інженерії [Електронний ресурс]: Режим доступу: https://www.zaptest.com/uk/тестування-чорної-скриньки-що-це-так.

30 Офіційна документація Stripe API [Електронний ресурс]: Режим доступу: https://docs.stripe.com/api.

31 Архітектурний патерн MVC у програмній інженерії [Електронний ресурс]: Режим доступу: https://developer.mozilla.org/en-US/docs/Glossary/MVC.

32 Оптимізація роботи з базами даних за допомогою ORM [Електронний ресурс]: Режим доступу: https://it-rating.ua/optimizatsiya-roboti-z-bazami-danih-za-dopomogoyu-orm-object-relational-mapping.

33 Використання Postman для тестування API [Електронний ресурс]: Режим доступу: https://training.qatestlab.com/blog/technical-articles/use-postman-in-testing/.

34 Методи підвищення продуктивності запитів у MySQL [Електронний ресурс]: Режим доступу: https://alexhost.com/uk/faq/how-to-use-mysql-query-optimization-techniques/.

35 Желібо Є.П. Безпека життєдіяльності : підручник / В. В. Зацарний. Київ : Каравела, 2023. 344 с.

36 Жидецький В.Ц. Охорона праці користувачів комп’ютерів : підручник. Львів : Афіша, 2020. 176 с.

37Kharchenko O., Bodnarchuk I., Galay I. Trade-off for quality attributes of software architecture on the base of multicriterion choice models // The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM) : Proceedings of the XII International Conference. — Lviv-Polyana : IEEE, 2013. — P. 145–147.

38 Bodnarchuk I., Kharchenko O., Galay I. Multicriteria architecture choice of software system under design and reengineering // The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM) : Proceedings of the XIII International Conference. — Lviv-Polyana : IEEE, 2015. — P. 328–330.

39 Боднарчук І. О., Галай І. О. Метод багатокритеріальної оптимізації програмної архітектури на основі аналізу компромісів // Інженерія програмного забезпечення. — Київ : НАУ, 2012. — № 3–4 (11–12). — С. 28–37.

ДОДАТКИ

Додаток А

Вміст файлу schema.prisma

```
generator client {
      provider = "prisma-client"
      output   = "../generated/prisma"
      moduleFormat = "cjs"
    }
datasource db {
      provider = "mysql"
    }
enum Role {
      user
      admin
    }
enum OrderStatus {
      PENDING
      PAID
      PROCESSING
      SHIPPED
      DELIVERED
      CANCELLED
    }
model User {
      id        Int      @id @default(autoincrement())
      createdAt DateTime @default(now()) @map("created_at")
      updatedAt DateTime @updatedAt @map("updated_at")
      email     String   @unique
      password  String
      role    Role  @default(user)
      favorites Favorite[]
      cart Cart[]
      orders Order[]
    }
model Product {
      id Int @id @default(autoincrement())
      name String @unique
      imagePath String @map("image_path")
      description String @db.Text
      price Decimal @db.Decimal(10,2)
      stock Int @default(0)
      warrantyMonths Int @default(0) @map("warranty_months")
      createdAt DateTime @default(now()) @map("created_at")
      updatedAt DateTime @updatedAt @map("updated_at")
      isActive Boolean  @default(true)
      categoryId Int @map("category_id")
      category Category @relation(fields:
      [categoryId],  references: [id])
      brandId Int @map("brand_id")
      brand Brand @relation(fields: [brandId], references: [id])
      favorites Favorite[]
```

```
    cart Cart[]
    orderItems OrderItem[]
    @@index([categoryId])
  }
model Brand {
    id Int @id @default(autoincrement())
    name String @unique
    products Product[]
  }
model Category {
    id Int @id @default(autoincrement())
    name String
    imagePath String? @map("image_path")
    parentId Int? @map("parent_id")
    parent    Category?    @relation("CategoryTree",    fields:
  [parentId], references: [id])
    children Category[] @relation("CategoryTree")
    products Product[]
    @@index([parentId])
    @@unique([name, parentId])
  }
model Favorite {
    id Int @id @default(autoincrement())
    userId Int @map("user_id")
    productId Int @map("product_id")
    user  User  @relation(fields:  [userId],  references:  [id],
  onDelete: Cascade)
    product  Product  @relation(fields:  [productId],  references:
  [id], onDelete: Cascade)
    @@unique([userId, productId])
    @@index([userId])
    @@index([productId])
  }
model Cart {
    id Int @id @default(autoincrement())
    userId Int @map("user_id")
    productId Int @map("product_id")
    quantity Int @default(1)
    addedAt DateTime @default(now())
    user  User  @relation(fields:  [userId],  references:  [id],
  onDelete: Cascade)
    product  Product  @relation(fields:  [productId],  references:
  [id], onDelete: Cascade)
    @@unique([userId, productId])
    @@index([userId])
    @@index([productId])
  }
model Order {
    id Int @id @default(autoincrement())
    userId Int @map("user_id")
    createdAt DateTime @default(now()) @map("created_at")
    updatedAt DateTime @updatedAt @map("updated_at")
    totalPrice Decimal @db.Decimal(10,2) @map("total_price")
```

```
  deliveryFee     Decimal     @db.Decimal(10,2)     @default(0)
@map("delivery_fee")
  user User @relation(fields: [userId], references: [id])
  status OrderStatus @default(PENDING)
  country String
  fullName String @map("full_name")
  company String?
  address String
  postCode String @map("post_code")
  city String
  phone String
  items OrderItem[]
  @@index([userId])
  @@index([status])
}
model OrderItem {
  id Int @id @default(autoincrement())
  orderId Int @map("order_id")
  productId Int @map("product_id")
  quantity Int
  price Decimal @db.Decimal(10,2)
  order  Order  @relation(fields:  [orderId],  references:  [id],
onDelete: Cascade)
  product  Product  @relation(fields:  [productId],  references:
[id], onDelete: Restrict)
  @@index([orderId])
  @@index([productId])
}
```

Додаток Б

Програмний код компонента App.tsx

```
import { BrowserRouter, Route, Routes } from "react-router-dom";
import HomePage from "./pages/HomePage";
import { ChakraProvider } from "@chakra-ui/react";
import theme from "./theme.ts";
import MainLayout from "./components/layouts/MainLayout";
import { lazy } from "react";
import ProtectedRoute from "./components/auth/ProtectedRoute.tsx";
import { Toaster } from "./components/ui/toaster.tsx";
import AdminRoute from "./components/auth/AdminRoute.tsx";
import AdminLayout from "./components/layouts/AdminLayout.tsx";
const ProductsPage = lazy(() => import("./pages/ProductsPage"));
const ProductPage = lazy(() => import("./pages/ProductPage"));
const CartPage = lazy(() => import("./pages/CartPage"));
const FavoritesPage = lazy(() => import("./pages/FavoritesPage"));
const CheckoutPage = lazy(() => import("./pages/CheckoutPage"));
const ProfilePage = lazy(() => import("./pages/ProfilePage"));
const PaymentSuccessPage = lazy(() =>
import("./pages/PaymentSuccessPage.tsx"));
const AdminDashboardPage = lazy(() =>
import("./pages/AdminDashboardPage.tsx"));
const AdminBrandsPage = lazy(() =>
import("./pages/AdminBrandsPage.tsx"));
const AdminCategoriesPage = lazy(() =>
import("./pages/AdminCategoriesPage.tsx"));
const AdminProductsPage = lazy(() =>
import("./pages/AdminProductsPage.tsx"));
const AdminOrdersPage = lazy(() =>
import("./pages/AdminOrdersPage.tsx"));
const SignInPage = lazy(() => import("./pages/SignInPage"));
const SignUpPage = lazy(() => import("./pages/SignUpPage"));
const NotFoundPage = lazy(() => import("./pages/NotFoundPage"));
const App = () => {
      return (
        <BrowserRouter>
          <ChakraProvider value={theme}>
            <Routes>
              <Route element={<MainLayout />}>
                <Route path="/" element={<HomePage />} />
                <Route path="/products" element={<ProductsPage />} />
                <Route path="/products/:categoryId"
               element={<ProductsPage />} />
                <Route path="/product/:id" element={<ProductPage />}/>
                <Route element={<ProtectedRoute />}>
                  <Route path="/cart" element={<CartPage />} />
                  <Route path="/favorites"
                    element={<FavoritesPage />} />
                  <Route path="/checkout"
                    element={<CheckoutPage />} />
```

```
                    <Route path="/profile" element={<ProfilePage />} />
                    <Route
                      path="/payment/success"
                      element={<PaymentSuccessPage />} />
                  </Route>
                </Route>
                <Route element={<AdminRoute />}>
                  <Route element={<AdminLayout />}>
                    <Route path="/admin"
                      element={<AdminDashboardPage />} />
                    <Route
                      path="/admin/brands"
                      element={<AdminBrandsPage />} />
                    <Route
                      path="/admin/categories"
                      element={<AdminCategoriesPage />} />
                    <Route
                      path="/admin/products"
                      element={<AdminProductsPage />} />
                    <Route
                      path="/admin/orders"
                      element={<AdminOrdersPage />} />
                  </Route>
                </Route>
                <Route path="/sign-in" element={<SignInPage />} />
                <Route path="/sign-up" element={<SignUpPage />} />
                <Route path="*" element={<NotFoundPage />} />
              </Routes>
              <Toaster />
            </ChakraProvider>
          </BrowserRouter>
        );
      };
export default App;
```

Додаток В

Програмний код захищеного API-модуля (protected.api.ts)

```
import {
      createApi,
      fetchBaseQuery,
      type BaseQueryFn,
      type FetchArgs,
      type FetchBaseQueryError,
    } from "@reduxjs/toolkit/query/react";
import { API_URL } from "@/constants/api.constants";
import Cookies from "js-cookie";
import { saveAuthData, removeAuthData }
from "@/utils/auth.helper";
import type { UserResponse } from "@/types/auth.types";

const baseQuery = fetchBaseQuery({
      baseUrl: API_URL,
      prepareHeaders: (headers) => {
        const accessToken = Cookies.get("accessToken");

         if (accessToken) {
          headers.set("authorization", `Bearer ${accessToken}`);
        }

         return headers;
      },
    });

const   baseQueryWithReauth:   BaseQueryFn<string   |   FetchArgs,
    unknown, FetchBaseQueryError> = async (
      args,
      api,
      extraOptions,
    ) => {

    let result = await baseQuery(args, api, extraOptions);

    if (result.error && result.error.status === 401) {

    const refreshToken = Cookies.get("refreshToken");

    if (refreshToken) {
          const refreshResult = await baseQuery(
            {
              url: "/auth/login/access-token",
              method: "POST",
              body: { refreshToken },
            },
            api,
            extraOptions,
```

```
            );
            if (refreshResult.data) {
              const data = refreshResult.data as UserResponse;
              saveAuthData(data);
              result = await baseQuery(args, api, extraOptions);

            } else {
              removeAuthData();
              window.location.href = "/sign-in";
            }
          } else {
            removeAuthData();
            window.location.href = "/sign-in";
          }
        }
        return result;
      };
      export const protectedApi = createApi({
        reducerPath: "protectedApi",
        baseQuery: baseQueryWithReauth,
        tagTypes:   ["UserProfile",   "Cart",   "Order",   "Brands",
      "Categories", "Products"],
        endpoints: () => ({}),
});
```

Додаток Г

Програмний код API-товарів (product.api.ts)

```
import {
      type AdminProduct,
      type CurrentProduct,
      type GetAdminProductsArgs,
      type GetAdminProductsResponse,
      type GetProductsArgs,
      type GetProductsResponse,
      type GetSimilarProductsArgs,
    } from "@/types/product.types";
import { protectedApi } from "./protected.api";
import { publicApi } from "./public.api";
import { orderApi } from "./order.api";
import { cartApi } from "./cart.api";
import { userApi } from "./user.api";
export const publicProductApi = publicApi.injectEndpoints({
      endpoints: (builder) => ({
        getProducts:             builder.query<GetProductsResponse,
    GetProductsArgs>({
          query: ({ categoryId, page = 1, perPage = 9, ...rest })
    => ({
            url: categoryId ? `/product/by-category/${categoryId}`
    : "/product/all",
            params: {
              page,
              perPage,
              ...rest,
            },
          }),
          providesTags: ["Products"],
        }),
        getProductById: builder.query<CurrentProduct, number>({
          query: (id) => `/product/${id}`,
          providesTags: (_, __, id) => [{ type: "Products", id }],
        }),
        getSimilarProducts:      builder.query<GetProductsResponse,
    GetSimilarProductsArgs>({
          query: ({ id, page = 1, perPage = 4 }) => ({
            url: `/product/similar/${id}`,
            params: { page, perPage },
          }),
          providesTags: ["Products"],
        }),
      }),
    });
    export const adminProductApi = protectedApi.injectEndpoints({
      endpoints: (builder) => ({
        getAdminProducts:   builder.query<GetAdminProductsResponse,
    GetAdminProductsArgs>({
```

```
      query: ({ page = 1, perPage = 10, searchTerm,
showArchived }) => ({
        url: "/product/admin/all",
        params: { page, perPage, searchTerm, showArchived },
      }),
      providesTags: ["Products"],
    }),
    createProduct:            builder.mutation<CurrentProduct,
FormData>({
      query: (formData) => ({
        url: "/product",
        method: "POST",
        body: formData,
      }),
      invalidatesTags: ["Products"],
      async onQueryStarted(_, { dispatch, queryFulfilled }) {
        try {
          await queryFulfilled;
 dispatch(publicProductApi.util.invalidateTags(["Products"]));
        } catch {}
      },
    }),
    updateProduct:  builder.mutation<CurrentProduct,  {  id:
number; formData: FormData }>({
      query: ({ id, formData }) => ({
        url: `/product/${id}`,
        method: "PATCH",
        body: formData,
      }),
      invalidatesTags: ["Products"],
      async onQueryStarted({ id }, { dispatch, queryFulfilled
}) {
        try {
          await queryFulfilled;

dispatch(publicProductApi.util.invalidateTags(["Products",   {
type: "Products", id }]));
          dispatch(cartApi.util.invalidateTags(["Cart"]));

dispatch(userApi.util.invalidateTags(["UserProfile"]));
        } catch {}
      },
    }),
    toggleProductArchive:       builder.mutation<AdminProduct,
number>({
      query: (id) => ({
        url: `/product/${id}/archive`,
        method: "PATCH",
      }),
      invalidatesTags: ["Products"],
      async onQueryStarted(_, { dispatch, queryFulfilled }) {
        try {
          await queryFulfilled;
```

```
    dispatch(publicProductApi.util.invalidateTags(["Products"]));
              dispatch(orderApi.util.invalidateTags(["Order"]));
              dispatch(cartApi.util.invalidateTags(["Cart"]));
            } catch {}
          },
        }),
      }),
    });

    export const { useGetProductsQuery, useGetProductByIdQuery,
    useGetSimilarProductsQuery } = publicProductApi;

    export const {
      useGetAdminProductsQuery,
      useCreateProductMutation,
      useUpdateProductMutation,
      useToggleProductArchiveMutation,
} = adminProductApi;
```

Додаток Д

Програмний код бізнес-логіки товарів (product.service.ts)

```
import { Injectable, Logger, NotFoundException }
from '@nestjs/common'
import { PrismaService } from '../prisma/prisma.service'
import { PaginationService }
from '../pagination/pagination.service'
import {
    EnumProductSort,
    GetAdminProductsDto,
    GetAllProductDto
} from './dto/get-all.product.dto'
import { Prisma } from '../../generated/prisma/client'
import {
    productAdminReturnObject,
    productGetReturnObject,
    productReturnObject
} from './return-product.object'
import { PaginationDto } from '../pagination/pagination.dto'
import { join } from 'path'
import { unlink } from 'fs/promises'
import    {    CreateProductDto,    UpdateProductDto    }    from
    './dto/product.dto'
@Injectable()
export class ProductService {
private readonly logger = new Logger(ProductService.name)
constructor(
         private prisma: PrismaService,
         private pagination: PaginationService
    ) {}
private async deleteLocalFile(imagePath: string) {
const fullPath = join(process.cwd(), imagePath.replace(/^\//, ''))
try {
    await unlink(fullPath)
} catch (err) {
this.logger.warn(`Failed    to    delete    file:    ${fullPath}`,
String(err))
}}
async getAdminAll(dto: GetAdminProductsDto = {}) {
const { sort, searchTerm, showArchived } = dto
const prismaSort: Prisma.ProductOrderByWithRelationInput[] = []
switch (sort) {
    case EnumProductSort.Alphabetical:
         prismaSort.push({ name: 'asc' })
         break
    case EnumProductSort.EXPENSIVE:
         prismaSort.push({ price: 'desc' })
         break
    case EnumProductSort.CHEAPER:
         prismaSort.push({ price: 'asc' })
```

```
    break
    default:
    prismaSort.push({ id: 'desc' })
    break
    }
prismaSort.push({ stock: 'desc' })
const    prismaSearchTermFilter:    Prisma.ProductWhereInput    =
searchTerm? {
OR: [
    { name: { contains: searchTerm } },
    { brand: { name: { contains: searchTerm } } },
    { category: { name: { contains: searchTerm } } }
    ]}: {}
const prismaArchiveFilter: Prisma.ProductWhereInput =
showArchived === 'true' ? { isActive: false } : { isActive: true }
const whereFilters: Prisma.ProductWhereInput = {
AND: [prismaSearchTermFilter, prismaArchiveFilter]}
const { perPage, skip } = this.pagination.getPagination(dto)
const products = await this.prisma.product.findMany({
    where: whereFilters,
    select: productAdminReturnObject,
    orderBy: prismaSort,
    skip,
    take: perPage
})
    return {
          products,
          length: await this.prisma.product.count({
               where: whereFilters
          })
         }
    }
async getAll(categoryId: number | undefined, dto: GetAllProductDto
= {}) {
const { sort, searchTerm, brand, minPrice, maxPrice } = dto
const prismaSort: Prisma.ProductOrderByWithRelationInput[] = []
    switch (sort) {
         case EnumProductSort.Alphabetical:
               prismaSort.push({ name: 'asc' })
               break
         case EnumProductSort.EXPENSIVE:
               prismaSort.push({ price: 'desc' })
               break
         case EnumProductSort.CHEAPER:
               prismaSort.push({ price: 'asc' })
               break
         default:
               prismaSort.push({ id: 'desc' })
               break
         }
prismaSort.push({ stock: 'desc' })
let prismaCategoryFilter: Prisma.ProductWhereInput = {}
```

```
          if (categoryId) {
           const category = await this.prisma.category.findUnique({
                where: { id: categoryId },
                include: { children: true }
           })
           if (category) {
                const categoryIds = [
                     category.id,
                     ...category.children.map(child => child.id)
                ]
                prismaCategoryFilter = {
                     categoryId: {
                          in: categoryIds
                     }
                }
           }
          }
const prismaSearchTermFilter:
Prisma.ProductWhereInput = searchTerm? {
OR: [
    {
          name: {
          contains: searchTerm
    }},
    {
          brand: {
          name: {
          contains: searchTerm
    }}
    },
    {
          category: {
          name: {
          contains: searchTerm
    }}}]}: {}

const prismaSearchBrandFilter: Prisma.ProductWhereInput = brand? {
          brand: {
          name: {
          contains: brand
          }}}: {}

const prismaPriceFilter: Prisma.ProductWhereInput = {}
          if (minPrice !== undefined || maxPrice !== undefined) {
           prismaPriceFilter.price = {
           ...(minPrice !== undefined ? { gte: minPrice } : {}),
           ...(maxPrice !== undefined ? { lte: maxPrice } : {})
           }}
const { perPage, skip } = this.pagination.getPagination(dto)

const whereFilters: Prisma.ProductWhereInput = {
    AND: [
          { isActive: true },
```

```
            prismaSearchTermFilter,
            prismaSearchBrandFilter,
            prismaCategoryFilter,
                  prismaPriceFilter
             ]
            }

const products = await this.prisma.product.findMany({
    where: whereFilters,
    select: productGetReturnObject,
    orderBy: prismaSort,
    skip,
    take: perPage
})
            return {
             products,
             length: await this.prisma.product.count({
                  where: whereFilters
             })
            }
    }
async byId(id: number) {
    const product = await this.prisma.product.findUnique({
          where: { id },
          select: productReturnObject
         })
    if (!product) throw new NotFoundException('Product not found')
          return product
    }

async getSimilar(id: number, dto: PaginationDto) {
    const { perPage, skip } = this.pagination.getPagination(dto)
    const currentProduct = await this.byId(id)
    if (!currentProduct)
    throw new NotFoundException('Current Product not found')

    const products = await this.prisma.product.findMany({

          where: {
                categoryId: currentProduct.categoryId,
                isActive: true,
                NOT: {
                     id: currentProduct.id
                }
           },
           orderBy: [{ stock: 'desc' }, { price: 'desc' }],
           select: productGetReturnObject,
           skip,
           take: perPage
          })
          return {
           products,
           length: await this.prisma.product.count({
```

```
                where: {
                     categoryId: currentProduct.categoryId,
                     isActive: true,
                     NOT: {
                          id: currentProduct.id
                     }
                }
           })
          }
    }

async createProduct(dto: CreateProductDto,
imagePath?: string | null) {

    const product = await this.prisma.product.create({
           data: {
                name: dto.name,
                description: dto.description,
                imagePath: imagePath || '',
                price: dto.price,
                stock: dto.stock,
                warrantyMonths:
                dto.warrantyMonths ? +dto.warrantyMonths : 0,
                brandId: dto.brandId,
                categoryId: dto.categoryId
           },
           select: productReturnObject
          })

          return product
    }

async updateProduct(
          id: number,
          dto: UpdateProductDto,
          imagePath?: string | null
    ) {

          const product = await this.byId(id)

    if (imagePath && product.imagePath) {

          await this.deleteLocalFile(product.imagePath)
    }

          return this.prisma.product.update({
           where: { id },
           data: {
                name: dto.name,
                description: dto.description,
                imagePath: imagePath !== undefined ? imagePath || ''
    : undefined,
                price: dto.price,
```

```
                stock: dto.stock,
                warrantyMonths:
dto.warrantyMonths ? +dto.warrantyMonths : undefined,
                brandId: dto.brandId,
                categoryId: dto.categoryId
            }
        })
    }

async toggleProductArchive(id: number) {
        const product = await this.prisma.product.findUnique({
         where: { id }
        })

        if (!product)
        throw new NotFoundException('Product not found')

        return this.prisma.product.update({
         where: { id },
         data: {
              isActive: !product.isActive
         },
         select: productAdminReturnObject
        })
    }
}
```