БАКАЛАВРСЬКА РОБОТА

БР.ІП-89.00.00.000 ПЗ

Група ІП-22-4

Білоусов Роман

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Білоусов Роман Ігорович

(прізвище, ім'я, по батькові)

УДК 004.42
(індекс)

БАКАЛАВРСЬКА РОБОТА

Веб-застосунок по продажу техніки Apple

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121– Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня Білоусов Р. І.
(підпис, ініціали та прізвище здобувача)

Науковий керівник Корнута Володимир Андрійович, к.т.н., доцент
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

доц. Бандура В.В.
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

***ЗАТВЕРДЖУЮ*:**
Зав. кафедрою ***ІПЗ***
доц. В.В. Бандура
“____” _____ 2026 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Білоусову Роману Ігоровичу

(прізвище, ім’я, по-батькові)

1. ***Тема проекту (роботи)*** **“Веб-застосунок по продажу техніки Apple”**
керівник проекту (роботи) Корнута Володимир Андрійович, к.н.т., доцент

затверджені наказом закладу вищої освіти від “ 21 ” квітня 2026 р. № 179/7

2. ***Строк подання студентом проекту (роботи)*** 2026 р.

3. ***Вихідні дані до проекту (роботи)*** програмний код Full-Stack веб-застосунку Apple Shop, клієнтська частина на React, серверна частина на Node.js та Express, локальне Docker-середовище, дані каталогу нової та вживаної Apple-техніки, каталог аксесуарів, локальні зображення товарів, результати функціонального тестування, конфігурація розгортання та інтеграції з Google Identity, LiqPay і Nova Post API.

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

1. Аналіз предметної області та сучасних рішень електронної комерції.
2. Формування вимог до веб-застосунку Apple Shop.
3. Проєктування архітектури системи та основних бізнес-процесів.
4. Реалізація frontend, backend, каталогу, кошика, оплати, доставки та Trade In.
5. Тестування веб-застосунку та аналіз результатів впровадження.

5. ***Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)***

1. Загальна архітектура веб-застосунку Apple Shop (рис. 3.1, ст. 38)
2. UML-діаграма варіантів використання Apple Shop (рис. 3.2, ст. 41)
3. Структура репозиторію Apple Shop (рис. 4.1, ст. 52)
4. Схема рівнів тестування Apple Shop (рис. 5.1, ст. 67)
5. Підсумковий вигляд головної сторінки після впровадження (рис. 5.3, ст. 71)

1. ***Консультанти розділів проекту (роботи)***

Розділ	***Консультант***	***Підпис, дата***	
		Завдання видав	***Завдання прийняв***

2. ***Дата видачі завдання*** _____

Керівник _____ Корнута В. А.
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ Білоусов Р. І.
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	**Назва етапів дипломного проекту (роботи)**	**Строк виконання етапів проекту**	**Примітка**
1	Визначення та обґрунтування теми роботи	27.02.2026	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	05.03.2026	виконано
3	Побудова моделі або алгоритму власного рішення	23.04.2026	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	13.05.2026	виконано
5	Оформлення пояснювальної записки бакалаврської роботи завідувачем кафедри	09.06.2026	виконано

Студент – дипломник Білоусов Р. І.
(підпис) (розшифровка підпису)

Керівник роботи Корнута В. А.
(підпис) (розшифровка підпису)

АНОТАЦІЯ

Бакалаврська робота містить 74 сторінки, 14 рисунків, список використаних джерел із 50 найменуваннями.

Метою роботи є розробка веб-застосунку Apple Shop для продажу Apple-техніки, оплати, доставки та адміністрування.

Об'єкт дослідження: процеси розробки та функціонування веб-застосунків електронної комерції.

Предмет дослідження: методи, технології та інструменти реалізації інтернет-магазину з каталогом товарів, кошиком, системою замовлень, оплатою, доставкою та адмініструванням.

Результати дослідження: реалізовано прототип Apple Shop на основі React, Node.js та Express із підтримкою оплати, доставки, авторизації та Trade In.

У першому розділі проаналізовано сучасний стан електронної комерції, існуючі рішення, технології та нормативні аспекти.

У другому розділі сформульовано вимоги, визначено ролі користувачів і постановку задачі проєктування.

У третьому розділі наведено архітектуру системи, описано компоненти, бізнес-процеси, UML-діаграми та зовнішні інтеграції.

У четвертому розділі описано реалізацію модулів: каталогів товарів, глобального пошуку, кошика, checkout, Trade In, адміністративної панелі, LiqPay-оплати та доставки Новою поштою.

У п'ятому розділі представлено результати тестування, перевірку основних сценаріїв роботи системи та аналіз впровадження.

Висновок: прототип підтвердив можливість реалізації інтернет-магазину Apple-техніки з основними комерційними функціями.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУНОК, ІНТЕРНЕТ-МАГАЗИН, ЕЛЕКТРОННА КОМЕРЦІЯ, APPLE SHOP, REACT, NODE.JS, EXPRESS, REST API, LIQPAY, NOVA POST API, DOCKER COMPOSE.

ABSTRACT

The bachelor’s thesis contains 74 pages, 14 figures, and a list of 50 references.

The aim of the work is to develop the Apple Shop web application for selling Apple devices, processing payments, arranging delivery, and administration.

Object of research: the processes of developing and functioning of e-commerce web applications.

Subject of research: methods, technologies, and tools for implementing an online store with a product catalog, shopping cart, order system, payment, delivery, and administration.

Research results: an Apple Shop prototype was implemented using React, Node.js, and Express, with support for payment, delivery, authorization, and Trade In.

The first chapter analyzes the current state of e-commerce, existing solutions, technologies, and regulatory aspects.

The second chapter defines the requirements, user roles, and the design task.

The third chapter presents the system architecture, components, business processes, UML diagrams, and external integrations.

The fourth chapter describes the implementation of the main modules, integrations, and admin panel.

The fifth chapter presents testing results, verification of the main system scenarios, and implementation analysis.

Conclusion: the prototype confirmed the feasibility of an Apple device online store with core commercial features.

KEYWORDS: WEB APPLICATION, ONLINE STORE, E-COMMERCE, APPLE SHOP, REACT, NODE.JS, EXPRESS, REST API, LIQPAY, NOVA POST API, DOCKER COMPOSE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API - Application Programming Interface, інтерфейс прикладного програмування для взаємодії між частинами системи або зовнішніми сервісами.

Frontend - клієнтська частина веб-застосунку, що відповідає за інтерфейс і взаємодію з користувачем.

Backend - серверна частина веб-застосунку, що відповідає за бізнес-логіку, обробку запитів, дані та інтеграції.

REST - архітектурний стиль побудови веб API.

JSON - текстовий формат обміну структурованими даними між клієнтом і сервером.

React - JavaScript-бібліотека для побудови користувацьких інтерфейсів.

Node.js - середовище виконання JavaScript на сервері.

Express - веб-фреймворк для Node.js, використаний для побудови серверного API.

Docker Compose - інструмент для запуску багатоконтейнерного застосунку.

LiqPay - платіжний сервіс, використаний для онлайн-оплати замовлень.

Nova Post API - API Нової пошти для отримання відділень і поштоматів під час оформлення доставки.

Google Identity Services - сервіс Google для авторизації користувачів через Google-акаунт.

Trade In - процес обміну старого пристрою користувача на новий із можливою доплатою.

Checkout - процес оформлення замовлення, що включає введення даних, вибір доставки й оплати.

ВСТУП

Електронна комерція є одним із найважливіших напрямів розвитку сучасних інформаційних систем. Веб-застосунки для продажу товарів уже давно виконують не лише функцію електронного каталогу, а стали повноцінними сервісами для пошуку, вибору, оплати, доставки та адміністрування замовлень. Особливо актуальним це є для сфери продажу техніки, де користувач очікує швидкого доступу до характеристик, якісних зображень, актуальної ціни, зручного кошика, зрозумілого оформлення замовлення та можливості вибору способу доставки й оплати.

Актуальність теми бакалаврської роботи зумовлена потребою розроблення сучасного інтернет-магазину Apple-техніки, який поєднує зручний користувацький інтерфейс, серверну бізнес-логіку, адміністративне керування та інтеграцію із зовнішніми сервісами. На практиці спеціалізований магазин техніки має працювати не лише з новими пристроями, а й із вживаними товарами, аксесуарами, заявками Trade In, відгуками, замовленнями, онлайн-оплатою та доставкою. Тому створення такого веб-застосунку є актуальним як з погляду програмної інженерії, так і з погляду практичного використання в електронній комерції.

Об’єктом дослідження є процес розроблення веб-застосунку електронної комерції для продажу техніки та аксесуарів.

Предметом дослідження є методи, архітектурні рішення та програмні засоби створення інтернет-магазину Apple-техніки з підтримкою каталогу товарів, кошика, замовлень, авторизації, доставки, онлайн-оплати, Trade In і адміністративної панелі.

Метою бакалаврської роботи є розроблення веб-застосунку Apple Shop, який забезпечує продаж нової та вживаної Apple-техніки, аксесуарів, оформлення замовлень, вибір доставки, онлайн-оплату, авторизацію користувачів і керування магазином через адміністративну панель.

Для досягнення поставленої мети необхідно виконати такі завдання:

- проаналізувати сучасний стан електронної комерції у сфері продажу техніки та визначити особливості предметної області.
- розглянути існуючі технології, архітектурні підходи й інструменти розробки сервісних веб-застосунків.
- сформулювати функціональні та нефункціональні вимоги до системи Apple Shop.
- спроєктувати архітектуру веб-застосунку, основні бізнес-процеси, компоненти та структури даних.
- реалізувати клієнтську частину, серверну частину, каталог товарів, кошик, checkout, авторизацію, Trade In і адміністративну панель.
- інтегрувати зовнішні сервіси для онлайн-оплати, доставки та авторизації.
- провести тестування системи, оцінити результати та визначити можливості подальшого розвитку.

У роботі використано методи аналізу предметної області, порівняння існуючих рішень, моделювання програмної архітектури, компонентного проєктування, структурного аналізу вимог, ручного й автоматизованого тестування. Для реалізації системи використано React, Vite, Node.js, Express, Docker Compose, nginx, Google Identity Services, LiqPay та Nova Post API.

Практичне значення роботи полягає у створенні працездатного прототипу інтернет-магазину Apple-техніки з каталогом товарів, кошиком, оформленням замовлень, доставкою, авторизацією, Trade In та адміністративною панеллю.

У результаті роботи буде розроблено веб-застосунок Apple Shop, який дозволить користувачам переглядати нову й вживану техніку, аксесуари, оформлювати замовлення та подавати заявки на обмін старого пристрою, а адміністратору - керувати товарами, відгуками, замовленнями й заявками.

Структура та обсяг роботи. Робота містить 74 сторінки, 14 рисунків і 50 джерел. Складається з вступу, п’яти розділів, висновків та додатків, де описано аналіз, проєктування, реалізацію та тестування Apple Shop.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ТЕОРЕТИЧНИХ ОСНОВ ПРОЄКТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ APPLE-ТЕХНІКИ

1.1. Аналіз сучасного стану електронної комерції у сфері продажу техніки

Електронна комерція є одним із ключових напрямів розвитку сучасних інформаційних систем, оскільки поєднує технологічні, економічні, організаційні та правові аспекти взаємодії між продавцем і покупцем. В українському законодавстві електронна комерція визначається як сфера відносин, що виникають під час вчинення електронних правочинів із застосуванням інформаційно-комунікаційних систем [1]. Тому інтернет-магазин слід розглядати не лише як вебсторінку з переліком товарів, а як програмну систему, що забезпечує пошук, вибір товару, оформлення замовлення, оплату, доставку, збереження даних користувача та дотримання вимог щодо захисту прав споживача.

Розвиток електронної комерції пов’язаний із загальною цифровізацією суспільства. За даними DataReportal, кількість користувачів інтернету у світі продовжує зростати, а цифрові сервіси стали звичним середовищем для пошуку інформації, комунікації, оплати послуг і придбання товарів [4]. UNCTAD у звіті Digital Economy Report 2024 також підкреслює, що цифрова економіка активно розширюється, а електронна комерція є важливою частиною глобальних економічних процесів [5]. Це формує нові очікування користувачів: інтернет-магазин має бути швидким, зрозумілим, безпечним і зручним цифровим сервісом.

Особливо високі вимоги висуваються до інтернет-магазинів техніки. На відміну від простих товарів, техніка має значну кількість характеристик: модель, покоління, обсяг пам’яті, колір, стан, комплектацію, гарантію, ціну та наявність. У випадку продукції Apple додатково важливими є точність назви моделі, правильне відображення кольору, розділення нових і вживаних пристроїв,

наявність аксесуарів та можливість Trade In. Покупець очікує, що сайт дозволить швидко порівняти товари, переглянути зображення, зрозуміти різницю між моделями та оформити замовлення без зайвих дій.

Сучасні рішення у сфері електронної комерції можна умовно поділити на три групи: універсальні маркетплейси, спеціалізовані магазини техніки та SaaS/CMS-платформи для швидкого запуску інтернет-магазину. Маркетплейси мають масштаб, велику клієнтську базу й розвинену логістику, але менш гнучкі щодо специфіки окремої предметної області. Спеціалізовані магазини краще враховують особливості техніки, аксесуарів і сервісних послуг. SaaS- та CMS-рішення дозволяють швидко створити базовий магазин, однак часто обмежують індивідуальну логіку, складні інтеграції, сценарії Trade In або кастомну адміністративну панель.

Для теми цієї роботи найбільш релевантними є спеціалізовані інтернет-магазини техніки, оскільки вони повинні підтримувати не лише продаж товарів, а й додаткові бізнес-процеси. До таких процесів належать продаж нових пристроїв, продаж вживаної техніки, продаж аксесуарів, приймання заявок на обмін старого пристрою на новий, робота з відгуками, опрацювання замовлень, підтвердження оплати та вибір доставки. Саме тому в межах дипломного проєкту доцільно створювати не абстрактний каталог товарів, а веб-застосунок Apple Shop, орієнтований на повний цикл взаємодії користувача з магазином.

Однією з поширених проблем інтернет-магазинів є фрагментарність користувацького досвіду. Каталог може працювати окремо від пошуку, пошук може не знаходити товари з усіх розділів, кошик може втрачати вибраний колір товару, а адміністративна панель може бути незручною для щоденної роботи. Дослідження Baymard Institute показують, що для електронної комерції суттєве значення мають повнота інформації у списках товарів, зрозуміла фільтрація, якісний пошук, коректна checkout-логіка та зменшення зайвих перешкод на шляху до покупки [6, 7].

Важливу роль у сучасних інтернет-магазинах відіграють зображення товарів. Для покупця техніки фото виконує не лише декоративну, а й

інформаційну функцію. Якщо товар має кілька кольорів, зміна кольору повинна змінювати зображення. Якщо товар є вживаним, фото має відповідати конкретному пристрою або його фактичному стану. Якщо аксесуар доступний у кількох кольорах, користувач повинен бачити, який саме варіант додає до кошика. У дослідженнях Baymard зазначається, що недостатність інформації у товарних списках ускладнює процес вибору та змушує користувачів виконувати зайві переходи [6].

Ще однією важливою особливістю електронної комерції є інтеграція із зовнішніми сервісами. Інтернет-магазин зазвичай не реалізує самостійно всі платіжні, логістичні та авторизаційні механізми, а інтегрується з платіжними системами, службами доставки, провайдерами авторизації та іншими API. Для українського ринку типовими прикладами є LiqPay для онлайн-оплати та API Нової пошти для вибору відділення або поштомата. Такий підхід дозволяє використовувати готові сервіси, які спеціалізуються на своїй предметній області [25, 26].

Поряд із функціональністю важливими залишаються правові та безпекові вимоги. Інтернет-магазин збирає персональні дані користувача: ім’я, номер телефону, електронну пошту, адресу доставки та історію замовлень. В Україні обробка персональних даних регулюється Законом України «Про захист персональних даних» [3]. Крім того, продаж товарів споживачам повинен враховувати вимоги Закону України «Про захист прав споживачів», який визначає права покупців і обов’язки продавців [2].

З погляду програмної інженерії інтернет-магазин є сервісним веб-застосунком, у якому поєднуються клієнтська частина, серверна логіка, API, сховище даних, модулі авторизації, оплати, доставки та адміністративна частина. Для таких систем важливими є підтримуваність, масштабованість, надійність, безпека та зручність використання. ISO/IEC 25010 визначає модель якості програмного продукту, яка включає функціональну придатність, продуктивність, сумісність, зручність використання, надійність, безпеку, супроводжуваність і переносимість [11]. Ці характеристики можна застосувати й до Apple Shop.

Аналіз існуючих рішень показує, що більшість сучасних інтернет-магазинів уже реалізують базові можливості: каталог, кошик, оформлення замовлення, оплату та доставку. Водночас не всі рішення однаково добре працюють зі специфічними сценаріями, такими як розділення нової та вживаної техніки, Trade In, локальне керування зображеннями, відображення конкретного кольору в кошику й замовленні, адміністративне підтвердження оплати або зручна адмін-панель без надмірної прокрутки.

Отже, сучасний стан електронної комерції свідчить про актуальність розробки спеціалізованих веб-застосунків для продажу техніки. Користувачі очікують від таких систем швидкої навігації, якісного пошуку, коректного відображення варіантів, безпечної авторизації, зручної доставки та прозорого оформлення замовлення. Наявні рішення частково задовольняють ці вимоги, однак для предметної області Apple-техніки доцільно розробити власну систему, яка враховує особливості нових пристроїв, БУ-товарів, аксесуарів і Trade In.

1.2. Основні поняття та особливості предметної області інтернет-магазину Apple-техніки

Для проєктування веб-застосунку Apple Shop необхідно визначити основні поняття предметної області та програмної інженерії. У межах цієї роботи інтернет-магазин розглядається як веб-застосунок, що забезпечує перегляд товарів, пошук, формування кошика, оформлення замовлення, оплату, доставку, авторизацію користувачів та адміністративне керування даними. Такий підхід відповідає розумінню електронної комерції як діяльності, що здійснюється із застосуванням інформаційно-комунікаційних систем [1].

Веб-застосунок відрізняється від статичного вебсайту тим, що має прикладну логіку та реагує на дії користувача. Він отримує дані із сервера, змінює стан інтерфейсу, виконує перевірки, зберігає інформацію та взаємодіє із зовнішніми сервісами. Для Apple Shop це означає, що система повинна не лише відображати сторінки з товарами, а й забезпечувати повний сценарій покупки:

від вибору товару до створення замовлення, вибору доставки та оплати.

Основою роботи Apple Shop є клієнт-серверна архітектура. Клієнтська частина працює у браузері користувача та відповідає за інтерфейс: сторінки, товарні картки, форми, кошик, пошук і адміністративну панель. Серверна частина обробляє запити, виконує бізнес-логіку, працює з даними, авторизацією, оплатою, доставкою та замовленнями. Такий поділ дозволяє розмежувати відповідальність між frontend і backend та спростити подальшу підтримку системи.

Для взаємодії між клієнтською і серверною частинами використовується API. API є інтерфейсом прикладного програмування, через який одна частина системи звертається до іншої. У веб-застосунках поширеним є REST-підхід, описаний Роєм Філдінгом як архітектурний стиль для мережевих програмних систем [12]. У Apple Shop REST API може використовуватися для отримання списку товарів, створення замовлення, авторизації користувача, надсилання Trade In-заявки та отримання даних доставки.

Важливим поняттям є SPA, тобто односторінковий застосунок. У SPA після початкового завантаження сторінки подальша навігація та оновлення інтерфейсу можуть відбуватися без повного перезавантаження. Дослідження Mesbah і van Deursen підкреслює можливість незалежного оновлення компонентів інтерфейсу в таких системах [13]. Для Apple Shop це зручно, оскільки користувач часто перемикає категорії, виконує пошук, обирає колір товару, додає товари до кошика та переходить між сторінками.

Поняття компонента є одним із ключових у frontend-розробці. Компонент - це самостійна частина інтерфейсу, яка має власне призначення та може використовуватися повторно. Наприклад, товарна картка, навігаційне меню, пошуковий рядок, форма входу, кошик і вкладки адміністративної панелі можуть бути окремими компонентами. Документація React визначає компонентний підхід як основу побудови інтерфейсів [17]. Для Apple Shop це важливо, оскільки одна й та сама картка товару може використовуватися в каталозі, на головній сторінці, у розділі аксесуарів і в результатах пошуку.

Життєвий цикл програмного забезпечення охоплює аналіз, проєктування, реалізацію, тестування, розгортання, експлуатацію та супровід. Стандарт ISO/IEC/IEEE 12207 визначає процеси життєвого циклу програмного забезпечення [9]. У межах Apple Shop це проявляється у переході від аналізу предметної області до реалізації функцій, тестування системи, запуску через Docker і подальшого вдосконалення. Вимоги до системи поділяються на функціональні та нефункціональні. Функціональні вимоги визначають, що саме повинна робити система, а нефункціональні описують її якісні характеристики: продуктивність, безпеку, зручність використання, надійність і переносимість [10].

У предметній області Apple Shop важливими поняттями є товар, варіант товару, кошик і замовлення. Товар має назву, категорію, опис, ціну, залишок, зображення та характеристики. Він може бути новим пристроєм, вживаним пристроєм або аксесуаром. Новий пристрій може мати кілька варіантів, наприклад різні кольори або обсяг пам’яті. Вживаний пристрій зазвичай є конкретною одиницею, тому пам’ять, колір і стан повинні відображатися як фіксовані характеристики, а не як варіанти для вибору.

Варіант товару - це конкретне представлення товару з певними характеристиками, наприклад чохол у кольорі Winter Blue або кабель у чорному кольорі. Для інтернет-магазину важливо, щоб варіант зберігав не лише назву кольору, а й відповідне зображення. Якщо користувач додає до кошика товар конкретного кольору, у кошику та замовленні має відображатися саме цей варіант. Це зменшує ризик помилки під час покупки та підвищує точність виконання замовлення.

Кошик є проміжним етапом між переглядом товарів і оформленням замовлення. Він зберігає вибрані позиції, кількість, ціну, варіант, колір, пам’ять і зображення. Для Apple Shop кошик має об’єднувати товари з різних розділів: нову техніку, БУ-пристрої, аксесуари та кабелі. Замовлення є результатом checkout-процесу і містить дані користувача, список товарів, суму, спосіб доставки, спосіб оплати та статус.

Окремим бізнес-процесом є Trade In. У межах Apple Shop Trade In розглядається як заявка користувача на обмін старого пристрою на новий. Така заявка не завершується миттєвою оплатою, а потребує оцінки моделі, стану, комплектації та можливої доплати. Саме тому Trade In-заявка повинна бути пов’язана з авторизованим користувачем.

Адміністративна панель призначена для керування даними магазину. Вона повинна дозволяти адміністратору додавати й редагувати товари, завантажувати локальні зображення, видаляти відгуки, опрацьовувати Trade In-заявки та керувати замовленнями. Зручність адміністративної панелі є важливою, оскільки саме адміністратор підтримує актуальність каталогу та контролює стан замовлень.

Важливим поняттям також є авторизація. Вона визначає права користувачів у системі. Гість може переглядати товари та додавати їх до кошика, авторизований користувач може створювати Trade In-заявки й переглядати власні замовлення, адміністратор має доступ до панелі керування. Безпека авторизації є критичною для веб-застосунків, оскільки OWASP Top 10 відносить порушення контролю доступу та помилки автентифікації до важливих ризиків [23].

Отже, предметна область Apple Shop включає не лише каталог техніки, а й пов’язані процеси: продаж, оплату, доставку, Trade In, авторизацію, адміністрування та роботу із зображеннями. Для якісного проєктування системи необхідно враховувати як загальні поняття програмної інженерії, так і особливості продажу Apple-техніки.

1.3. Сучасні технології та тенденції розробки сервісних веб-застосунків

Сучасні сервісні веб-застосунки розробляються з урахуванням кількох ключових тенденцій: компонентної побудови інтерфейсу, використання JavaScript-екосистеми, API-орієнтованої архітектури, контейнеризації, інтеграції

із зовнішніми сервісами, підвищених вимог до безпеки та зручності користування. Для Apple Shop ці напрями мають практичне значення, оскільки система повинна бути не лише навчальним прикладом, а й реалістичним прототипом інтернет-магазину Apple-техніки.

Однією з основних технологій для реалізації клієнтської частини є React. Його документація визначає компонентний підхід як основу побудови інтерфейсів, де окремі частини UI можуть мати власну логіку та повторно використовуватися [17]. Для інтернет-магазину це особливо зручно, оскільки товарні картки, форми, кнопки, фільтри, пошуковий рядок, кошик і вкладки адміністративної панелі мають повторювану структуру. Українські дослідження також звертають увагу на доцільність використання React для масштабованих веб-застосунків та інтернет-магазинів [15].

У випадку Apple Shop компонентний підхід дозволяє підтримувати єдину візуальну систему. Наприклад, товарна картка може використовуватися на головній сторінці, у каталозі нової техніки, у розділі БУ-товарів, у аксесуарах і в результатах пошуку. Якщо потрібно змінити стиль або логіку картки, це можна зробити централізовано. Такий підхід зменшує дублювання коду, спрощує супровід і дає змогу реалізувати динамічну поведінку: зміну кольору товару, оновлення зображення, показ повідомлень і перемикання вкладок без повного перезавантаження сторінки.

Для збирання frontend-частини доцільно використовувати Vite. Офіційна документація Vite описує його як інструмент, що поєднує швидкий dev server для розробки та build-команду для формування оптимізованих статичних файлів [18]. Для дипломного проєкту це важливо, оскільки швидкий запуск середовища скорочує час перевірки змін, а production build дозволяє підготувати клієнтську частину до розгортання через nginx або інший вебсервер.

Серверна частина сучасних веб-застосунків часто реалізується на Node.js. Node.js позиціонується як асинхронне подієво-орієнтоване JavaScript-середовище для створення масштабованих мережевих застосунків [19]. Для Apple Shop це є доцільним, оскільки backend виконує багато коротких операцій:

отримання товарів, створення замовлень, перевірку авторизації, формування LiqPay-платежу та звернення до API Нової пошти. Використання JavaScript на frontend і backend також зменшує кількість різних мов у проєкті.

Для побудови HTTP API у Node.js часто використовується Express. Офіційна документація Express визначає його як мінімалістичний і гнучкий фреймворк для веб-застосунків Node.js, який забезпечує маршрутизацію та middleware-підхід [20]. У Apple Shop Express може застосовуватися для реалізації маршрутів товарів, замовлень, авторизації, Trade In, LiqPay, Нової пошти та адміністративних функцій. Middleware-підхід дозволяє додавати перевірку токена, обробку помилок, логування й валідацію даних.

Важливою тенденцією є API-орієнтоване проєктування. У такій моделі сервер не формує всі HTML-сторінки, а надає дані через API, тоді як клієнтська частина відповідає за їх відображення. Це дозволяє незалежно розвивати frontend і backend, а також створює основу для майбутнього розширення системи. Наприклад, у майбутньому до тих самих API можна підключити мобільний застосунок або окрему панель оператора.

Під час вибору архітектури важливо уникати надмірної складності. Мікросервісна архітектура є популярною, оскільки дозволяє розділяти систему на незалежні сервіси. Fowler і Lewis описують мікросервіси як підхід, за якого застосунок складається з набору малих сервісів, що працюють у власних процесах і взаємодіють через легкі механізми, часто HTTP API [14]. Проте для дипломного проєкту Apple Shop повноцінна мікросервісна архітектура була б надмірною, оскільки ускладнила б розгортання, тестування та керування даними. Тому доцільнішим є клієнт-серверний підхід із модульною серверною структурою.

Ще однією важливою тенденцією є контейнеризація. Docker Compose дозволяє описувати й запускати багатоконтейнерні застосунки за допомогою конфігураційного YAML-файлу [21]. Для Apple Shop це означає, що frontend, backend і reverse proxy можуть запускатися як окремі сервіси. Такий підхід спрощує демонстрацію та перевірку проєкту, оскільки середовище запуску стає

більш відтворюваним і не потребує ручного налаштування всіх компонентів на кожному комп'ютері.

Для розгортання frontend-частини та проксування API-запитів може використовуватися nginx. Офіційна документація nginx описує reverse proxy як механізм, за якого nginx приймає запит, передає його до проксованого сервера, отримує відповідь і повертає її клієнту [22]. У Apple Shop nginx може віддавати статичні файли React-застосунку та перенаправляти API-запити до Node.js backend. Це дозволяє мати єдину точку входу для користувача й робить структуру розгортання зрозумілішою. Для обґрунтування вибору стеку основні технології проєкту порівняно з альтернативами, що наведено в таблиці 1.1.

Таблиця 1.1

Порівняння обраних технологій

Компонент	**Обрана технологія**	**Альтернативи**	**Обґрунтування вибору**
Frontend	React	Vue.js, Angular	Компонентний підхід, зручне повторне використання елементів інтерфейсу
Збирання frontend	Vite	Webpack, Parcel	Швидкий запуск, проста конфігурація, швидке формування production build
Backend	Node.js	PHP, Python, Java	Можливість використовувати JavaScript на клієнті й сервері
Backend-фреймворк	Express	NestJS, Fastify	Простота реалізації REST API, маршрутизації та middleware
API	REST API	GraphQL, gRPC	Простий і зрозумілий формат взаємодії frontend і backend
Контейнеризація	Docker Compose	Ручне налаштування, Kubernetes	Зручний запуск frontend, backend і nginx як єдиної системи
Reverse proxy	nginx	Apache, Caddy	Віддавання frontend-файлів і проксування API-запитів до backend

Окрему роль відіграють зовнішні інтеграції. Google Identity Services дозволяє реалізувати авторизацію через Google-акаунт [24]. LiqPay Checkout API передбачає формування параметрів data і signature на сервері для переходу

користувача до сторінки оплати [25]. Nova Post API дає змогу отримувати дані про відділення та поштомати через JSON-запити з API-ключем [26]. Для Apple Shop такі інтеграції покращують користувацький досвід: користувач може швидко увійти, оплатити замовлення та вибрати зручне місце доставки без переходу на сторонні сайти для пошуку інформації.

Безпека залишається однією з найважливіших вимог до сервісних веб-застосунків. OWASP Top 10 визначає критичні ризики, серед яких порушення контролю доступу, криптографічні помилки, ін’єкції, неправильна конфігурація безпеки та проблеми автентифікації [23]. Для Apple Shop це означає необхідність перевіряти роль користувача перед доступом до адміністративної панелі, не дозволяти неавторизованим користувачам надсилати Trade In-заявки, не зберігати секретні ключі у frontend-коді та коректно обробляти помилки API.

Крім технічної частини, важливою тенденцією є уніфікація користувацького інтерфейсу. Інтернет-магазин повинен мати єдину систему відступів, кольорів, карток, кнопок, шрифтів і форматів цін. Якщо головна сторінка, каталог, БУ-товари, аксесуари й кошик мають різний стиль, користувач сприймає сайт як незавершений. Тому для Apple Shop важливо використовувати однакові принципи оформлення на всіх сторінках, а ціни відображати в єдиному форматі.

Сучасні веб-застосунки також повинні бути адаптивними. Користувач може відкривати сайт на ноутбуці, великому моніторі або смартфоні. Для магазину техніки це означає, що товарна сітка повинна коректно змінювати кількість колонок, зображення не повинні розтягувати сторінку, текст не має виходити за межі карток, а сайт не повинен мати горизонтальної прокрутки. Зручність використання є однією з характеристик якості програмного продукту у моделі ISO/IEC 25010 [11].

Для невеликого магазину важливим є локальне керування контентом. Адміністратор повинен мати можливість додавати товари, редагувати характеристики та завантажувати зображення без прямого редагування коду. У Apple Shop це реалізується через адміністративну панель і локальне

завантаження файлів. Такий підхід наближає систему до реального використання, оскільки каталог може оновлюватися без участі розробника.

У наукових і прикладних роботах, присвячених веб-застосункам електронної комерції, часто обґрунтовується використання клієнт-серверної архітектури, SPA-підходу, React, Node.js і RESTful API. Наприклад, у дослідженні щодо веб-застосунку книжкового магазину-маркетплейсу розглядається використання React.js, Node.js, RESTful API та клієнт-серверної архітектури як основи для електронної комерції [16]. Це підтверджує, що обраний напрям технологічної реалізації Apple Shop відповідає сучасній практиці розробки веб-застосунків.

Таким чином, сучасні технології розробки сервісних веб-застосунків дозволяють створити інтернет-магазин, який є зручним для користувача, підтримуваним для розробника і придатним до подальшого розширення. Для Apple Shop найбільш доцільним є поєднання React, Vite, Node.js, Express, Docker, nginx і зовнішніх API. Такий набір технологій охоплює основні потреби системи: інтерфейс, серверну логіку, розгортання, оплату, доставку та авторизацію.

1.4. Висновки по розділу

У першому розділі було розглянуто сучасний стан електронної комерції у сфері продажу техніки та визначено теоретичні основи проєктування інтернет-магазину Apple Shop. Встановлено, що сучасний інтернет-магазин є не просто вебсторінкою з переліком товарів, а повноцінною програмною системою, яка повинна забезпечувати каталог, пошук, кошик, оформлення замовлення, оплату, доставку, авторизацію, адміністрування, роботу з персональними даними та зручну взаємодію користувача з інтерфейсом.

Аналіз існуючих рішень показав, що універсальні маркетплейси, SaaS-платформи та спеціалізовані магазини мають власні переваги й обмеження. Маркетплейси забезпечують широку аудиторію та готову інфраструктуру продажів, SaaS-рішення дають змогу швидко запустити продажі, а спеціалізовані

магазини забезпечують більшу гнучкість і контроль над логікою системи. Для дипломного проєкту найбільш доцільним є створення власного спеціалізованого веб-застосунку, оскільки він дозволяє врахувати специфіку Apple-техніки: розділення нових і вживаних товарів, роботу з аксесуарами, локальні зображення, варіанти кольорів, Trade In-заявки, вибір доставки через Нову пошту та адміністративне керування замовленнями.

У розділі визначено основні поняття предметної області: інтернет-магазин, веб-застосунок, клієнт-серверна архітектура, API, SPA, компонент, товар, варіант товару, кошик, замовлення, Trade In, адміністративна панель та авторизація. Також розглянуто поняття життєвого циклу програмного забезпечення, функціональних і нефункціональних вимог, архітектурних патернів і характеристик якості програмного продукту.

Огляд сучасних технологій показав доцільність використання React для клієнтського інтерфейсу, Vite для збирання frontend-частини, Node.js та Express для серверного API, Docker Compose для контейнерного запуску, nginx для reverse proxy, а також зовнішніх сервісів Google Identity, LiqPay і Nova Post API. Обраний технологічний підхід відповідає сучасним тенденціям розробки сервісних веб-застосунків і дозволяє створити систему, придатну для демонстрації, тестування та подальшого розвитку.

Отже, перший розділ сформував теоретичну й аналітичну основу для подальшої роботи. Визначені особливості електронної комерції, вимоги до спеціалізованого інтернет-магазину та обґрунтований вибір технологій будуть використані під час аналізу вимог, проєктування архітектури та реалізації системи Apple Shop.

РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ПОСТАНОВКА ЗАДАЧІ

2.1. Аналіз вимог до веб-застосунку інтернет-магазину Apple Shop

Розроблення веб-застосунку Apple Shop потребує попереднього аналізу вимог, оскільки саме вони визначають функціональність майбутньої системи, її межі, ролі користувачів, якісні характеристики та критерії приймання. Відповідно до ISO/IEC/IEEE 29148, вимоги є основою для проєктування, реалізації, перевірки та супроводу програмної системи, тому їх необхідно формулювати чітко, повно та перевірювано [10]. Для інтернет-магазину це особливо важливо, оскільки система повинна забезпечувати повний цикл взаємодії користувача з магазином: перегляд товарів, пошук, вибір варіанта, додавання до кошика, оформлення замовлення, оплату, доставку та адміністрування.

Під час аналізу вимог до Apple Shop враховувалася специфіка предметної області. Магазин Apple-техніки відрізняється від універсального інтернет-магазину тим, що товари мають багато параметрів: модель, покоління, колір, пам’ять, стан, комплектацію, категорію, ціну, залишок і зображення. Крім нових товарів, система повинна підтримувати вживану техніку, де пам’ять і колір є не варіантами вибору, а характеристиками конкретної одиниці товару. Також система повинна містити аксесуари, серед яких окремі позиції мають кольорові варіанти із власними зображеннями.

Збір вимог виконувався на основі кількох джерел інформації. Першим джерелом був аналіз існуючих інтернет-магазинів техніки та типових сценаріїв електронної комерції. Це дозволило визначити функції, які користувачі очікують бачити у сучасному магазині: каталог, категорії, пошук, кошик, checkout, оплату, доставку, авторизацію і власні замовлення. Дослідження Baymard Institute показують, що для користувацького досвіду в електронній комерції особливо важливими є якісні товарні списки, зрозумілий checkout і відсутність зайвих перешкод під час оформлення покупки [6], [7].

Другим джерелом вимог був аналіз законодавчих і нормативних аспектів. Оскільки інтернет-магазин працює у сфері електронної комерції, він повинен враховувати вимоги Закону України «Про електронну комерцію» [1]. Також система обробляє персональні дані користувача, тому потрібно враховувати вимоги Закону України «Про захист персональних даних» [3]. Крім того, магазин продає товари кінцевим споживачам, тому важливими є положення Закону України «Про захист прав споживачів» [2]. Ці документи впливають на вимоги до прозорості оформлення замовлення, коректності інформації про товар і збереження персональних даних.

Третім джерелом вимог був аналіз ролей користувачів. У системі виділено чотири основні ролі: гість, авторизований користувач, адміністратор і зовнішній сервіс. Гість може переглядати товари, користуватися пошуком, додавати товари до кошика та переходити до оформлення замовлення. Авторизований користувач додатково може переглядати власні замовлення та подавати Trade In-заявку. Адміністратор має доступ до панелі керування товарами, відгуками, Trade In-заявками й замовленнями. Зовнішні сервіси LiqPay, Nova Post API та Google Identity забезпечують оплату, доставку й авторизацію [24], [25], [26].

Збір вимог також включав аналіз сценаріїв використання. Сценарій використання описує послідовність дій, яку виконує користувач для досягнення певної мети. Такий підхід дозволяє виявити не лише окремі функції, а й зв’язки між ними. Наприклад, вимога «додати товар до кошика» пов’язана з пошуком товару, вибором кольору або іншого варіанта, збереженням правильного зображення та передаванням цих даних у замовлення.

Одним із базових сценаріїв є перегляд каталогу нової техніки. Користувач відкриває сторінку каталогу, бачить категорії у верхній частині сторінки, фільтрує товари, переглядає картки та переходить на сторінку конкретного товару. Для цього сценарію важливими є швидкість завантаження, зрозуміла структура карток, єдиний формат цін і відсутність горизонтальної прокрутки. Google Core Web Vitals розглядає якість користувацького досвіду через завантаження, інтерактивність і візуальну стабільність сторінки [29].

Другим важливим сценарієм є глобальний пошук. Користувач повинен мати можливість знайти не лише новий iPhone, а й БУ-пристрій, чохол, кабель, зарядний блок, клавіатуру чи інший аксесуар. Отже, вимога до глобального пошуку передбачає пошук по всіх товарних групах: новій техніці, вживаних товарах і аксесуарах. Це підвищує зручність використання та зменшує кількість зайвих переходів.

Третім сценарієм є перегляд сторінки товару. На сторінці товару користувач повинен бачити назву, категорію, ціну, опис, характеристики, зображення та доступні варіанти. Якщо товар має кольори, вибір кольору повинен змінювати зображення без перезавантаження сторінки. Якщо товар належить до БУ-категорії, пам’ять і колір повинні відображатися як фіксовані характеристики, а не як кнопки вибору.

Четвертим сценарієм є оформлення замовлення. Користувач додає товари до кошика, переглядає список позицій, переходить до checkout, вводить контактні дані, обирає спосіб доставки, спосіб оплати та підтверджує замовлення. У разі вибору Нової пошти система повинна показувати реальні відділення та поштомати, використовуючи API служби доставки [26]. У разі вибору онлайн-оплати сервер повинен сформувати дані для LiqPay, оскільки приватний ключ платіжної системи не повинен передаватися у браузер [25].

П’ятим сценарієм є подання Trade In-заявки. Користувач хоче обміняти старий iPhone або інший пристрій на новий. Для цього він повинен бути авторизованим, оскільки заявка має бути пов’язана з конкретним користувачем. Якщо користувач не авторизований, система повинна заборонити надсилання заявки і запропонувати виконати вхід. Це необхідно для того, щоб адміністратор міг переглянути заявку та зв’язатися з користувачем.

Шостим сценарієм є робота адміністратора. Адміністратор повинен мати окрему панель із вкладками або кнопками для переходу між розділами: товари, відгуки, Trade In-заявки та замовлення. Такий підхід є зручнішим, ніж одна довга сторінка, яку потрібно постійно прокручувати. Nielsen Norman Group у своїх принципах зручності інтерфейсу підкреслює важливість видимості стану

системи, узгодженості, контролю користувача і зменшення зайвого когнітивного навантаження [32].

Під час збору вимог було враховано також вимоги до форм і повідомлень. Форми використовуються для входу, реєстрації, оформлення замовлення, Trade In-заявки, додавання товару та редагування даних. MDN зазначає, що перевірка форм може виконуватися на клієнтській стороні, але серверна перевірка також є необхідною, оскільки дані з браузера не можна вважати повністю надійними [31]. Повідомлення про виконані дії, наприклад додавання товару до кошика, повинні бути короткими та автоматично зникати через кілька секунд, щоб не перевантажувати інтерфейс [32].

У підсумку, аналіз вимог показав, що Apple Shop повинен бути не просто каталогом Apple-техніки, а комплексним веб-застосунком електронної комерції. Система повинна охоплювати публічну частину для покупця, персональні функції для авторизованого користувача, адміністративну панель для керування магазином і зовнішні інтеграції для оплати, доставки та авторизації. На основі цього аналізу у наступному підрозділі формулюються функціональні та нефункціональні вимоги.

2.2. Формулювання функціональних та нефункціональних вимог

Функціональні вимоги визначають, які дії повинна виконувати система. Для Apple Shop їх доцільно згрупувати за основними модулями: головна сторінка, каталог, пошук, сторінка товару, кошик, checkout, авторизація, Trade In, адміністративна панель, оплата, доставка та робота із зображеннями. Така класифікація дозволяє структурувати реалізацію та перевірити, чи всі бізнес-процеси мають програмну підтримку.

Головна сторінка повинна відображати промо-слайдер, блоки з хітами продажу, вкладки товарних добірок і сітку категорій. Вона виконує роль стартової точки, з якої користувач може перейти до каталогу, БУ-техніки, аксесуарів або Trade In. Каталог нової техніки повинен містити товари Apple,

згруповані за категоріями: iPhone, AirPods, Apple Watch, Mac, iPad та інші. На сторінці каталогу не повинно бути зайвого промо-слайдера, оскільки каталог має виконувати робочу функцію - показ товарів.

Окремий каталог БУ-техніки повинен містити вживані iPhone, iPad, MacBook та інші пристрої. Кожен такий товар є конкретною одиницею з фіксованим обсягом пам’яті, кольором і станом. Тому система не повинна показувати вибір пам’яті або кольору як варіанти, якщо для товару доступний лише один фактичний варіант. Сторінка аксесуарів повинна містити захисне скло, чохли, клавіатури, зарядні блоки, кабелі, мишки та інші товари. Якщо аксесуар має кольорові варіанти, вибір кольору повинен змінювати зображення товару.

Глобальний пошук повинен працювати по всьому сайту. Він має знаходити товари з каталогу нової техніки, БУ-техніки та аксесуарів. Результати пошуку повинні містити назву, категорію, ціну та зображення. Це відповідає практиці електронної комерції, де пошук є одним із головних інструментів навігації [6]. Сторінка товару повинна відображати назву, категорію, ціну, опис, характеристики, зображення, доступні варіанти, кнопку додавання до кошика та відгуки. Якщо товар має кілька кольорів, вибір кольору повинен змінювати головне зображення без повного оновлення сторінки.

Кошик повинен зберігати всі товари, які користувач додав до покупки. Для кожної позиції потрібно зберігати назву, ціну, кількість, вибраний колір, пам’ять, зображення варіанта і загальну суму. Checkout повинен дозволяти користувачу ввести контактні дані, вибрати спосіб доставки, спосіб оплати та підтвердити замовлення. Якщо обрано Нову пошту, система повинна показувати реальні відділення або поштомати через API служби доставки [26]. Якщо обрано онлайн-оплату, сервер повинен створити платіж LiqPay, оскільки приватний ключ платіжної системи не повинен передаватися у браузер [25].

Авторизація повинна підтримувати вхід через email і пароль, а також Google-авторизацію. Google Identity Services дозволяє реалізувати кнопку входу через Google на вебсайті [24]. Trade In повинен бути доступний тільки

авторизованим користувачам. Якщо неавторизований користувач намагається надіслати заявку, система повинна показати повідомлення і не створювати заявку. Форма Trade In повинна містити інформацію про старий пристрій, бажаний новий пристрій, стан, контактні дані й коментар.

Адміністративна панель повинна бути організована як набір окремих сторінок або вкладок. У ній повинні бути розділи товарів, відгуків, Trade In-заявок і замовлень. Адміністратор повинен мати можливість додавати й редагувати товари, завантажувати локальні зображення, видаляти недоречні коментарі, переглядати заявки, керувати замовленнями та підтверджувати оплату.

Нефункціональні вимоги визначають якість роботи системи. ISO/IEC 25010 виділяє такі характеристики якості програмного продукту, як функціональна придатність, продуктивність, сумісність, зручність використання, надійність, безпека, супроводжуваність і переносимість [11]. Для Apple Shop ці характеристики перетворюються на конкретні вимоги до швидкості, безпеки, дизайну, розгортання і підтримки.

Вимоги до продуктивності передбачають, що сторінки повинні завантажуватися швидко, а взаємодія з інтерфейсом не повинна супроводжуватися помітними затримками. Особливо важливо уникати перезавантаження сторінки під час зміни кольору товару або перемикання вкладок. Google Core Web Vitals визначає завантаження, інтерактивність і візуальну стабільність як важливі аспекти користувацького досвіду [29].

Вимоги до безпеки передбачають захист адміністративної панелі, контроль доступу, безпечне збереження секретних ключів і перевірку користувацьких даних. OWASP Top 10 визначає порушення контролю доступу, проблеми автентифікації, небезпечний дизайн і неправильну конфігурацію як критичні ризики веб-застосунків [23]. Для Apple Shop це означає, що адміністраторські дії повинні бути доступні лише адміністратору, Trade In-заявки не повинні створюватися без авторизації, а ключі LiqPay, Google і Нової пошти повинні зберігатися в .env.

Вимоги до зручності використання передбачають зрозумілу навігацію, однаковий формат карток товарів, логічне розташування елементів і відсутність зайвої прокрутки там, де її можна уникнути. Nielsen Norman Group підкреслює важливість узгодженості, видимості стану системи та запобігання помилкам [32]. Для Apple Shop це означає, що ціни повинні всюди відображатися у форматі з символом «₴», кнопки повинні мати однаковий стиль, а адміністративна панель повинна бути поділена на зручні розділи.

Вимоги до доступності, супроводжуваності та переносимості також є важливими. Інтерфейс повинен мати зрозумілі підписи полів, достатній контраст, коректні повідомлення про помилки та доступну структуру відповідно до рекомендацій WCAG 2.2 [30]. Код має бути розділений за відповідальністю: frontend - на компоненти сторінок, карток, форм і кошика, backend - на маршрути для товарів, замовлень, авторизації, Trade In, LiqPay і Нової пошти [9]. Переносимість забезпечується Docker Compose, який дозволяє описати кілька сервісів і запускати їх разом [21].

Критерії приймання описують умови, за яких вимога вважається виконаною. Wiegers і Beatty підкреслюють важливість конкретності вимог і погодженості щодо очікуваного результату [27]. Для Apple Shop критерії приймання повинні бути практичними: якщо користувач шукає аксесуар, він повинен знайтися; якщо користувач додає товар певного кольору, у кошику має бути саме цей колір; якщо адміністратор підтверджує оплату, статус замовлення повинен змінитися.

Таким чином, сформульовані функціональні та нефункціональні вимоги визначають очікування до Apple Shop. Функціональні вимоги описують, що саме повинна робити система, а нефункціональні - наскільки якісно вона повинна це робити.

2.3. Постановка задачі проектування системи

На основі проведеного аналізу вимог задача проєктування полягає у

створенні веб-застосунку Apple Shop, призначеного для продажу нової Apple-техніки, вживаних пристроїв, аксесуарів і підтримки Trade In-заявок. Система повинна забезпечувати користувацьку частину, адміністративну частину, серверне API, інтеграції з платіжним сервісом, службою доставки та Google-авторизацією. Кінцевим результатом має бути працездатний веб-застосунок, який можна запустити локально або в контейнерному середовищі та продемонструвати через браузер.

Метою проєктування є створення системи, яка поєднує зручний інтерфейс, коректну бізнес-логіку та придатність до подальшого розвитку. Система повинна бути достатньо простою для розуміння в межах бакалаврської роботи, але водночас достатньо повною, щоб демонструвати реальні процеси електронної комерції. Тому архітектура не повинна бути надмірно складною, наприклад повністю мікросервісною, але повинна мати чіткий поділ на frontend, backend, дані, інтеграції та deploy-конфігурацію.

Об’єктом проєктування є процес розроблення веб-застосунку електронної комерції для продажу техніки Apple. Предметом проєктування є методи, засоби й архітектурні рішення, які дозволяють реалізувати каталог товарів, глобальний пошук, кошик, замовлення, оплату, доставку, авторизацію, Trade In і адміністративне керування. Практична цінність системи полягає в тому, що вона моделює реальну роботу спеціалізованого магазину техніки.

До складу системи повинні входити такі основні підсистеми: публічний frontend, серверне API, модуль авторизації, модуль товарів, модуль кошика і замовлень, модуль оплати, модуль доставки, модуль Trade In, адміністративна панель і deploy-оточення. Кожна підсистема має власну відповідальність. Наприклад, frontend відповідає за інтерфейс користувача, backend - за бізнес-логіку і збереження даних, LiqPay-модуль - за формування платежу, Nova Post-модуль - за отримання відділень і поштоматів.

Межі системи визначають, які функції входять до дипломного проєкту, а які залишаються поза ним. До меж системи входить перегляд товарів, пошук, категорії, сторінка товару, кошик, checkout, Google-авторизація, Trade In,

адміністративна панель, LiqPay-оплата, Нова пошта, локальні зображення та Docker-запуск. До меж системи не входить повноцінна бухгалтерська система, складський облік у реальному часі, автоматичне створення товарно-транспортних накладних, промокоди, складна CRM-аналітика, програма лояльності та мобільний застосунок. Таке обмеження є обґрунтованим, оскільки дипломний проєкт повинен бути завершеним, але не надмірно розширеним.

Під час проєктування приймаються певні припущення. По-перше, система орієнтована на демонстраційне й навчальне використання, тому обсяг даних є обмеженим. По-друге, товари можуть зберігатися у файловому або простому локальному сховищі даних, якщо цього достатньо для дипломного прототипу. По-третє, зовнішні інтеграції залежать від правильності API-ключів і налаштувань акаунтів. Наприклад, LiqPay може вимагати коректного домену або налаштувань мерчанта, а Google-авторизація - правильного origin у консолі Google [24, 25].

Також приймається припущення, що адміністратор є довіреною особою магазину. Він має право додавати і редагувати товари, переглядати замовлення, підтверджувати оплату та видаляти відгуки. Водночас звичайний користувач не повинен мати доступу до цих функцій. Такий поділ відповідає вимогам безпеки і принципу мінімально необхідних прав. OWASP Top 10 визначає контроль доступу як один із ключових ризиків веб-застосунків, тому розмежування ролей є обов’язковою умовою [23].

Задача проєктування включає також визначення початкових даних. Для демонстрації системи необхідно наповнити каталог товарами. До нього мають входити нові iPhone, AirPods, Apple Watch, MacBook, iPad, аксесуари, а також БУ-пристрої, наприклад iPhone 15, iPhone 14, iPhone 13, старі MacBook та iPad. Для аксесуарів потрібно передбачити товари з кольорами: чохли, Smart Folio, Magic Mouse, кабелі. Це дозволить перевірити роботу варіантів, зображень, кошика та замовлень.

Окремим завданням є проєктування структури зображень. Система повинна підтримувати локальні файли, зокрема PNG, WebP та інші формати, які

коректно відображаються у браузері. Для товарів із прозорим фоном потрібно забезпечити однакове візуальне відображення на картках. Це важливо, оскільки різні товари можуть відображатися на головній сторінці, у каталозі, у пошуку й у кошику. Якщо фон і розмір картинок не узгоджені, сайт виглядатиме незавершеним.

Задача проєктування адміністративної панелі полягає в тому, щоб зробити її придатною для щоденної роботи. Панель не повинна бути однією довгою сторінкою. Вона повинна мати окремі вкладки або кнопки: товари, відгуки, Trade In, замовлення. У розділі товарів адміністратор повинен бачити список позицій і мати можливість створювати або редагувати товар. У розділі замовлень він повинен бачити статуси і мати можливість підтвердити оплату. Такий підхід зменшує час пошуку потрібної дії та покращує ергономіку.

Задача проєктування checkout-процесу включає збереження правильного стану замовлення. Якщо користувач вибрав конкретний колір товару, замовлення повинно містити саме цей колір. Якщо користувач вибрав доставку Новою поштою, замовлення повинно містити вибране відділення або поштомат. Якщо користувач обрав LiqPay, замовлення може мати статус очікування оплати до моменту підтвердження. Це дозволяє системі не втрачати замовлення навіть у випадку помилки зовнішнього платіжного сервісу.

Для перевірки виконання задачі проєктування визначено загальні критерії готовності системи. Система вважається реалізованою, якщо користувач може пройти шлях від головної сторінки до створення замовлення; якщо пошук працює по всіх товарах; якщо Trade In захищений авторизацією; якщо адміністратор може керувати товарами, відгуками, заявками і замовленнями; якщо LiqPay-дані формуються на сервері; якщо Нова пошта показує реальні точки доставки; якщо сайт запускається через Docker.

У результаті постановки задачі визначено, що Apple Shop повинен бути повноцінним прототипом інтернет-магазину Apple-техніки з практичною бізнес-логікою. Система повинна не лише демонструвати сторінки, а й зберігати дані, обробляти користувацькі дії, взаємодіяти із зовнішніми сервісами та надавати

адміністратору інструменти керування. Сформульована задача є основою для подальшого проєктування архітектури, структури даних і компонентів системи.

2.4. Висновки по розділу

У другому розділі виконано аналіз вимог до веб-застосунку Apple Shop і сформульовано постановку задачі проєктування. Було визначено основні джерела вимог: аналіз предметної області, вивчення існуючих інтернет-магазинів, законодавчі документи, сценарії використання, ролі користувачів і технічні можливості зовнішніх сервісів. Такий підхід дозволив сформувати не абстрактний перелік функцій, а пов’язану систему вимог до реального інтернет-магазину, який має працювати з товарами, користувачами, замовленнями, оплатою, доставкою та адміністративним керуванням.

У процесі аналізу виділено основні ролі системи: гість, авторизований користувач, адміністратор і зовнішні сервіси. Для кожної ролі визначено очікувані дії та результати. Гість повинен мати можливість переглядати товари, користуватися пошуком і ознайомлюватися з інформацією про магазин. Авторизований користувач отримує доступ до повного сценарію покупки, оформлення замовлення, вибору доставки та подання Trade In-заявки. Адміністратор відповідає за керування товарами, зображеннями, відгуками, замовленнями та заявками користувачів. Зовнішні сервіси забезпечують допоміжні функції, зокрема авторизацію, оплату та вибір відділень доставки.

Особливу увагу приділено сценаріям перегляду каталогу, глобального пошуку, вибору товару, додавання до кошика, оформлення замовлення, подання Trade In-заявки та роботи адміністратора. Аналіз цих сценаріїв дозволив визначити, які модулі повинні бути реалізовані у системі та як вони мають взаємодіяти між собою. Зокрема, було встановлено необхідність реалізації окремих сторінок для нової техніки, БУ-товарів, аксесуарів, Trade In, кошика, замовлень і адміністративної панелі.

Сформульовано функціональні вимоги до головної сторінки, каталогу

нової техніки, БУ-товарів, аксесуарів, сторінки товару, кошика, checkout, авторизації, Trade In, LiqPay, Нової пошти та адміністративної панелі. Окремо визначено вимоги до коректної роботи варіантів товару, зокрема кольору, пам’яті та відповідного зображення товару. Це є важливим для інтернет-магазину техніки, оскільки користувач повинен бачити саме той варіант товару, який додає до кошика або оформлює в замовленні.

Також визначено нефункціональні вимоги до продуктивності, безпеки, зручності використання, доступності, супроводжуваності та переносимості. Для системи електронної комерції ці вимоги мають не менше значення, ніж функціональні, оскільки навіть наявність усіх потрібних функцій не гарантує якісної роботи магазину без швидкого завантаження сторінок, зрозумілого інтерфейсу, захисту персональних даних і стабільної роботи основних модулів. Для ключових вимог наведено критерії приймання, які можуть використовуватися під час тестування готової системи.

На основі зібраних вимог сформульовано задачу проєктування: створити веб-застосунок Apple Shop, який забезпечує продаж нової та вживаної Apple-техніки, продаж аксесуарів, оформлення замовлень, онлайн-оплату, доставку Новою поштою, Google-авторизацію, Trade In-заявки та адміністративне керування магазином. Визначено межі системи, припущення, обмеження та критерії готовності. Це дозволяє чітко відокремити функції, які входять до меж дипломного проєкту, від можливостей, що можуть бути реалізовані в майбутньому.

Таким чином, у другому розділі було сформовано основу для практичного проєктування системи. Отримані результати дають змогу перейти від загального аналізу предметної області до конкретних архітектурних рішень. Вони є підґрунтям для третього розділу, у якому буде розглянуто архітектуру програмного забезпечення, бізнес-процеси, системні компоненти та обґрунтування вибору технологій для реалізації Apple Shop.

РОЗДІЛ 3. ПРОЕКТУВАННЯ СИСТЕМИ

3.1. Розробка архітектури програмного забезпечення

Проєктування архітектури програмного забезпечення є одним із ключових етапів розроблення Apple Shop, оскільки саме архітектура визначає структуру системи, спосіб взаємодії її компонентів, розподіл відповідальності між модулями та можливість подальшого розвитку. У праці Bass, Clements і Kazman архітектура програмного забезпечення розглядається як сукупність структур системи, що містять програмні елементи, їхні властивості та відношення між ними [33]. Для інтернет-магазину це означає, що ще до реалізації потрібно визначити основні частини системи, їхню взаємодію та потоки даних.

Під час вибору архітектурної моделі для Apple Shop розглядалися монолітна, мікросервісна та клієнт-серверна архітектура. Монолітна архітектура є простою на початковому етапі, але з часом може ускладнювати масштабування та незалежну зміну окремих частин системи. Мікросервісна архітектура передбачає поділ застосунку на незалежні сервіси, які взаємодіють через легкі механізми, часто HTTP API [14]. Для великих платформ це може бути доцільно, однак для дипломного проєкту такий підхід є надмірним, оскільки потребує складнішого розгортання, моніторингу й підтримки кількох окремих процесів.

Найбільш доцільною для Apple Shop є клієнт-серверна архітектура з модульною серверною частиною. Вона передбачає поділ системи на frontend, який працює у браузері користувача, та backend, який обробляє запити, працює з даними, авторизацією та зовнішніми сервісами. REST-підхід до побудови API є поширеним способом організації взаємодії між клієнтом і сервером у веб-застосунках [12]. У такій моделі frontend може розвиватися як інтерактивний React-застосунок, а backend - надавати API для товарів, замовлень, Trade In, оплати та доставки.

Обрана архітектура Apple Shop складається з п’яти основних рівнів: клієнтського, серверного, рівня даних, рівня зовнішніх інтеграцій та рівня

розгортання. Клієнтський рівень представлений React SPA і відповідає за сторінки магазину, товарні картки, каталог, БУ-товари, аксесуари, кошик, checkout, авторизацію, Trade In-сторінку та адміністративну панель. Серверний рівень представлений Node.js API, який приймає HTTP-запити, виконує бізнес-логіку, перевіряє дані, обробляє замовлення, авторизацію, заявки Trade In і адміністративні операції.

Рівень даних відповідає за збереження товарів, користувачів, замовлень, Trade In-заявок, відгуків, параметрів товарів і шляхів до локальних зображень. Рівень зовнішніх інтеграцій охоплює Google Identity для авторизації, LiqPay для онлайн-оплати та Nova Post API для роботи з відділеннями й поштоматами. Рівень розгортання містить Docker Compose і nginx: Docker Compose використовується для запуску компонентів системи в узгодженому середовищі, а nginx виконує роль reverse proxy. Загальну архітектуру системи наведено на рисунку 3.1.

Рисунок 3.1 - Загальна архітектура веб-застосунку Apple Shop

Архітектурно систему також можна описати через C4-підхід, який

використовується для візуалізації програмної архітектури на кількох рівнях: контекст, контейнери, компоненти та код [35]. Для Apple Shop на рівні контексту система взаємодіє з покупцем, адміністратором і зовнішніми сервісами. На рівні контейнерів можна виділити frontend-застосунок, backend API, сховище даних, файлове сховище зображень і reverse proxy. На рівні компонентів frontend поділяється на сторінки, UI-компоненти, модулі кошика, пошуку та адміністративної панелі, а backend - на маршрути товарів, замовлень, авторизації, Trade In, LiqPay і Нової пошти.

Важливою особливістю архітектури є розмежування публічної та адміністративної частини. Публічна частина призначена для всіх користувачів і містить головну сторінку, каталог, БУ-товари, аксесуари, сторінку товару, кошик, checkout, сторінку входу та Trade In. Адміністративна частина доступна лише користувачу з роллю адміністратора та містить керування товарами, відгуками, Trade In-заявками і замовленнями. Таке розмежування важливе з погляду безпеки, оскільки OWASP Top 10 визначає порушення контролю доступу як один із критичних ризиків веб-застосунків [23].

Клієнтська частина реалізується як SPA, що дозволяє уникати повного перезавантаження сторінки при переході між внутрішніми маршрутами або зміні стану інтерфейсу. Наприклад, під час вибору кольору товару має оновлюватися лише зображення, а не вся сторінка. Серверна частина виконує роль центрального вузла бізнес-логіки: перевіряє права доступу, читає або змінює дані, формує відповіді, створює платіжні дані для LiqPay, отримує інформацію від API Нової пошти та обробляє авторизацію. Секретні ключі повинні зберігатися у змінних середовища, що відповідає принципам Twelve-Factor App [37].

Для розгортання системи використовується контейнерний підхід. Docker Compose дозволяє описати кілька сервісів в одному конфігураційному файлі та запускати їх разом [21]. У випадку Apple Shop це дозволяє запускати frontend, backend і nginx як пов’язану систему. nginx у цій архітектурі може виконувати роль статичного вебсервера для frontend і reverse proxy для API-запитів [22]. У

межах дипломного проєкту можна використовувати спрощене локальне або файлове сховище даних, однак у майбутньому систему доцільно перенести на PostgreSQL, яка підтримує складні запити, транзакції та роботу зі структурованими даними [38].

Отже, архітектура Apple Shop проєктується як клієнт-серверна система з React frontend, Node.js backend, REST API, зовнішніми інтеграціями та контейнерним розгортанням. Такий підхід забезпечує достатню гнучкість, зрозумілу структуру, можливість тестування і подальшого розвитку без надмірної складності мікросервісної архітектури.

3.2. Моделювання бізнес-процесів та системних компонентів

Моделювання бізнес-процесів і системних компонентів дозволяє формалізувати логіку роботи Apple Shop до початку або під час реалізації. Для опису взаємодії користувачів із системою доцільно використовувати UML-діаграми, оскільки Unified Modeling Language є стандартною мовою для візуалізації, специфікації та документування програмних систем [34]. У межах цієї роботи найбільш корисними є діаграми варіантів використання, діаграми діяльності, діаграми послідовності та компонентні діаграми.

Першим кроком моделювання є визначення акторів системи. В Apple Shop виділяються такі актори: гість, авторизований користувач, адміністратор, Google Identity, LiqPay і Nova Post API. Гість взаємодіє з публічною частиною сайту: переглядає головну сторінку, каталог, аксесуари, БУ-товари, виконує пошук, переглядає сторінку товару та додає товари до кошика. Авторизований користувач має додаткові можливості: перегляд власних замовлень і подання Trade In-заявки. Адміністратор керує товарами, відгуками, замовленнями та Trade In. Зовнішні сервіси беруть участь у сценаріях авторизації, оплати та доставки. UML-діаграму варіантів використання Apple Shop наведено на рисунку 3.2.

Рисунок 3.2 - UML-діаграма варіантів використання Apple Shop

Основним бізнес-процесом системи є покупка товару. Він починається з того, що користувач відкриває головну сторінку або каталог. Далі користувач переглядає категорії, знаходить потрібний товар через каталог або пошук, відкриває сторінку товару, вибирає варіант, додає товар до кошика і переходить до оформлення замовлення. На етапі checkout користувач вводить контактні дані, обирає доставку та оплату. Після підтвердження система створює замовлення і, за потреби, формує LiqPay-платіж. UML-діаграму діяльності процесу оформлення замовлення наведено на рисунку 3.3.

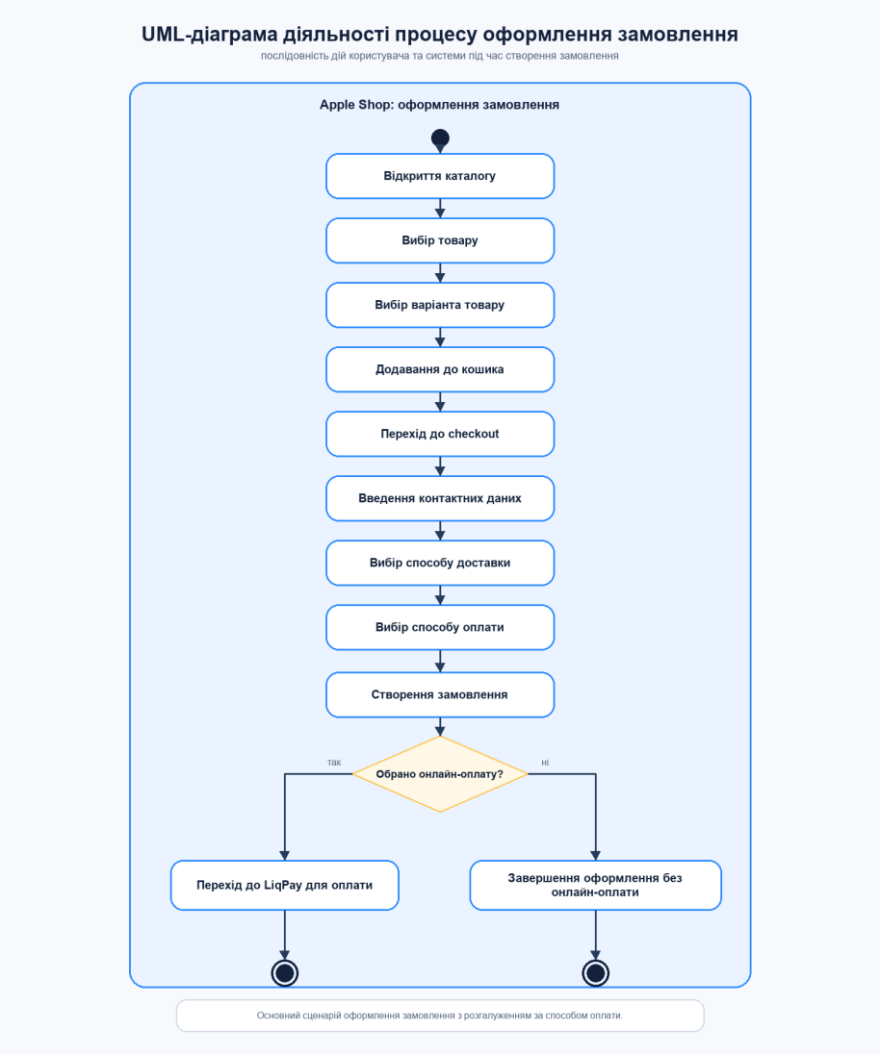

Рисунок 3.3 – UML-діаграма діяльності процесу оформлення замовлення

Особливу увагу потрібно приділити процесу додавання товару до кошика. Для звичайного товару достатньо зберегти ідентифікатор, назву, ціну і кількість. Проте для Apple Shop цього недостатньо. Якщо товар має кольорові варіанти, необхідно зберегти вибраний колір і відповідне зображення. Якщо товар є БУ-пристроєм, потрібно зберегти його фіксовані характеристики: пам’ять, колір, стан і стан акумулятора. Таким чином, кошик є не просто списком товарів, а структурою даних, що зберігає конкретний контекст вибору користувача.

Другим важливим бізнес-процесом є Trade In. Він починається з перевірки авторизації. Якщо користувач не авторизований, система не повинна дозволяти надсилання заявки. Якщо користувач авторизований, він заповнює форму, вказує модель старого пристрою, стан, бажаний новий пристрій і контактні дані. Після

надсилання заявка зберігається і стає доступною адміністратору. Цей процес відрізняється від замовлення тим, що не має миттєвої оплати і потребує подальшого ручного опрацювання. UML-діаграму діяльності процесу Trade In наведено на рисунку 3.4.

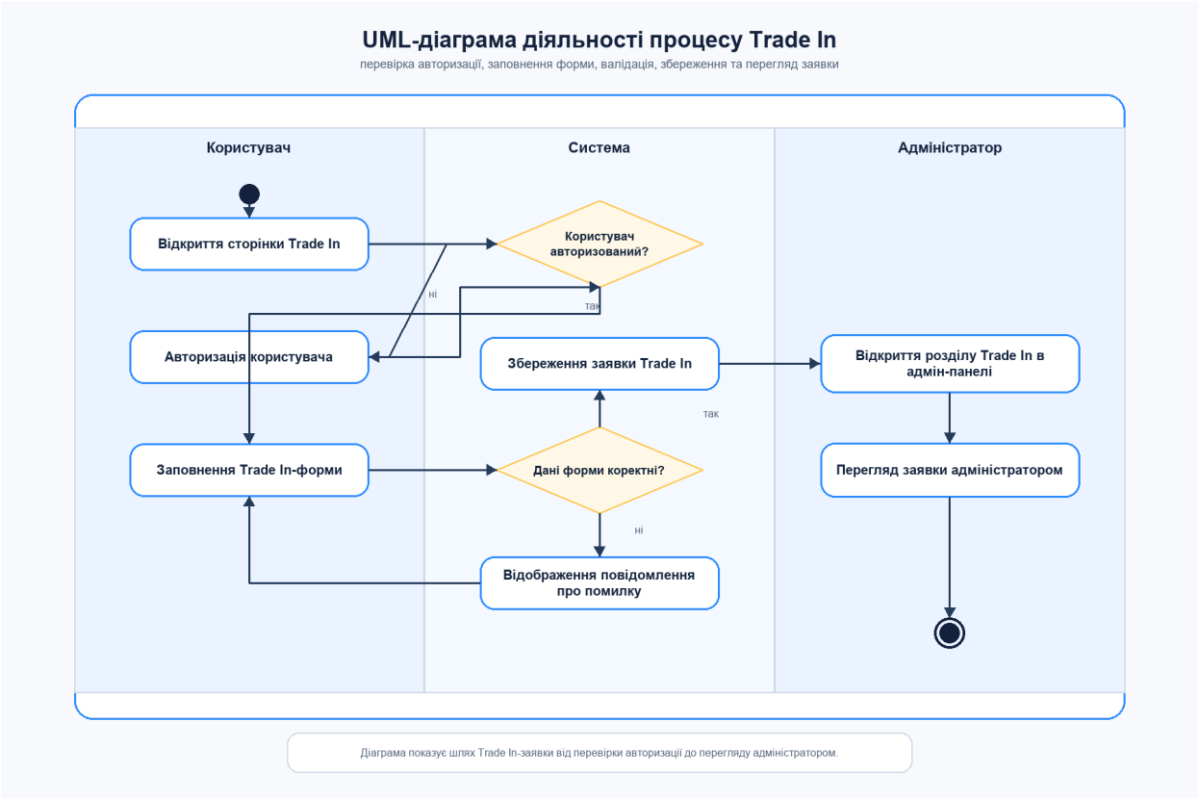

Рисунок 3.4 - UML-діаграма діяльності процесу Trade In.

Третім бізнес-процесом є адміністративне керування магазином. Адміністратор входить у систему, відкриває адміністративну панель і переходить між окремими розділами: товари, відгуки, Trade In, замовлення. У розділі товарів адміністратор додає або редагує товар, вказує назву, категорію, ціну, залишок, характеристики, варіанти та зображення. У розділі відгуків адміністратор видаляє недоречні коментарі. У розділі Trade In він переглядає заявки користувачів. У розділі замовлень адміністратор може підтвердити оплату, якщо замовлення перебуває у статусі очікування.

Компонентна модель frontend-частини формується відповідно до сторінок

і повторно використовуваних елементів. До основних сторінок належать HomePage, CatalogPage, UsedProductsPage, AccessoriesPage, ProductPage, SearchResultsPage, CartPage, CheckoutPage, OrdersPage, TradeInPage, LoginPage, SignupPage і AdminPage. До спільних компонентів належать Header, Footer, ProductCard, CategoryTabs, HeroSlider, ProductGrid, ToastMessage, ColorSelector, ImagePreview, AdminTabs і FormControls. Компонентний підхід дозволяє уникнути дублювання коду й забезпечити однакову поведінку товарних карток на різних сторінках [17].

Backend-частина проєктується як набір логічних модулів. Модуль товарів відповідає за отримання списку товарів, пошук, перегляд конкретного товару, додавання, редагування та роботу з варіантами. Модуль замовлень відповідає за створення замовлення, перегляд замовлень користувача, перегляд усіх замовлень адміністратором і зміну статусу оплати. Модуль авторизації відповідає за вхід через email і пароль, Google-авторизацію та перевірку ролей. Модуль Trade In відповідає за створення і перегляд заявок. Модуль LiqPay відповідає за формування платіжних даних, а модуль Nova Post - за отримання відділень і поштоматів.

Важливою частиною проєктування є визначення основних сутностей даних. До них належать User, Product, ProductVariant, CartItem, Order, OrderItem, Review і TradeInRequest. Сутність Product зберігає загальні дані товару: назву, категорію, опис, ціну, залишок і основне зображення. ProductVariant описує конкретний варіант товару: колір, пам’ять, зображення, додаткову ціну або інші параметри. Order містить інформацію про покупця, список товарів, суму, доставку, оплату і статус. TradeInRequest містить дані про старий пристрій і контакт із користувачем. UML-діаграму класів основних сутностей Apple Shop наведено на рисунку 3.5.

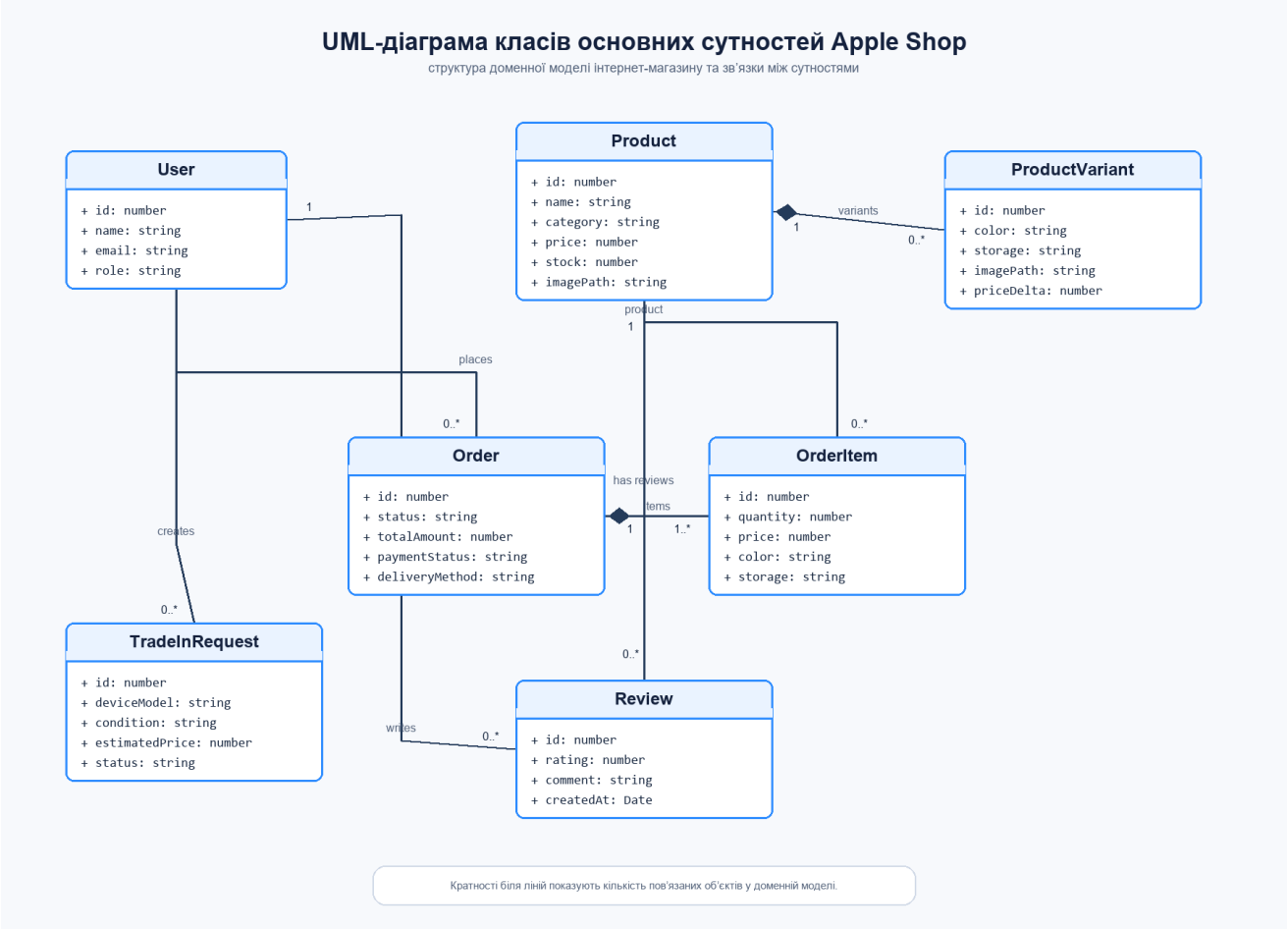

Рисунок 3.5 – UML-діаграма класів основних сутностей Apple Shop

Послідовність оформлення замовлення можна описати за допомогою UML-діаграми послідовності. Користувач взаємодіє з CheckoutPage, яка надсилає запит до Orders API. Backend перевіряє дані, створює замовлення, зберігає його у сховищі та повертає результат. Якщо обрано LiqPay, backend формує data і signature, після чого frontend перенаправляє користувача на сторінку оплати. LiqPay після завершення платежу може повернути результат або надіслати callback [25]. UML-діаграму послідовності оформлення замовлення з LiqPay наведено на рисунку 3.6.

Окремо моделюється процес вибору доставки. CheckoutPage надсилає запит до backend, backend звертається до Nova Post API, отримує список відділень або поштоматів і повертає його frontend. Користувач обирає потрібну точку доставки, після чого її ідентифікатор, назва або адреса зберігаються у замовленні. Такий підхід важливий, оскільки frontend не повинен напряму використовувати

секретний API-ключ служби доставки [26].

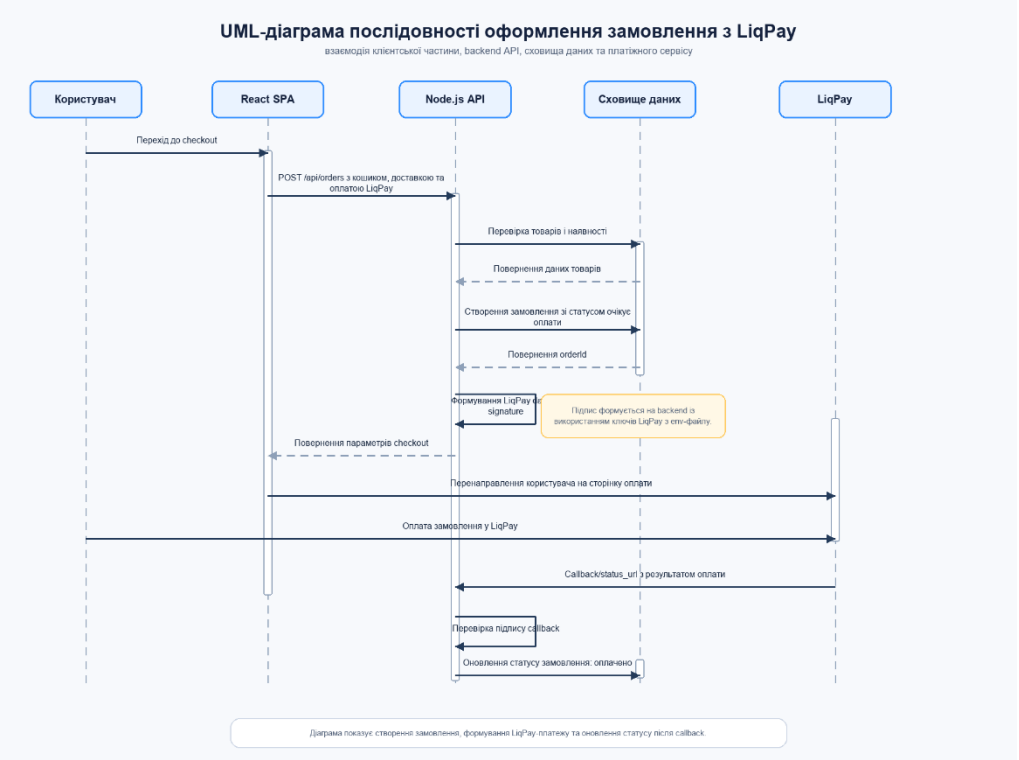

Рисунок 3.6 – UML-діаграма послідовності оформлення замовлення з LiqPay

Таким чином, моделювання бізнес-процесів і компонентів дозволяє показати, як система працює не лише на рівні сторінок, а й на рівні даних, ролей, зовнішніх сервісів і послідовності дій. Це створює основу для реалізації, тестування й подальшого розширення Apple Shop.

3.3. Вибір та обгрунтування технологій та інструментів розробки

Вибір технологій для Apple Shop здійснювався з урахуванням вимог, сформульованих у другому розділі. Система повинна мати інтерактивний інтерфейс, каталог товарів, глобальний пошук, кошик, checkout, авторизацію, Trade In, адміністративну панель, локальні зображення та зовнішні інтеграції. Крім того, вона має підтримувати контейнерний запуск і бути зрозумілою для

подальшого супроводу.

Для frontend-частини обрано React. Основною причиною вибору є компонентний підхід, який дозволяє будувати інтерфейс із повторно використовуваних частин [17]. Для Apple Shop це особливо важливо, оскільки однакові компоненти можуть застосовуватися на різних сторінках: товарна картка - у каталозі, на головній сторінці та в результатах пошуку, а вкладки - в адміністративній панелі. React також добре підходить для керування станом інтерфейсу: вибраним кольором, кошиком, результатами пошуку, повідомленнями та активною вкладкою.

Для збирання frontend-частини обрано Vite. Його перевагою є швидкий dev server і простий production build [18]. У процесі розробки це дозволяє швидко перевіряти зміни інтерфейсу, а на етапі розгортання - сформувати статичні файли, які може віддавати nginx. Vite добре поєднується з React, тому є доцільним вибором для сучасного SPA.

Для backend-частини обрано Node.js. Node.js є асинхронним JavaScript-середовищем для створення масштабованих мережевих застосунків [19]. Для Apple Shop це зручно, оскільки backend повинен обробляти багато коротких HTTP-запитів: отримання товарів, пошук, створення замовлень, авторизацію, завантаження зображень, роботу з LiqPay і Новою поштою. Використання JavaScript і на frontend, і на backend також зменшує складність технологічного стеку.

Для побудови API використовується Express. Express є мінімалістичним вебфреймворком для Node.js, який підтримує маршрутизацію та middleware [20]. Це дозволяє організувати backend як набір маршрутів: товари, замовлення, авторизація, Trade In, відгуки, завантаження файлів, LiqPay, Нова пошта та адміністративні операції. Middleware може застосовуватися для перевірки авторизації, ролі адміністратора, обробки помилок і валідації запитів.

Для розгортання обрано Docker Compose. Контейнеризація дозволяє описати середовище запуску незалежно від локального комп’ютера розробника [21]. У межах Apple Shop frontend, backend і nginx можуть запускатися як окремі

сервіси. Такий підхід зменшує кількість проблем під час демонстрації проєкту, оскільки не потрібно вручну налаштовувати всі залежності.

Nginx використовується для віддавання статичних файлів frontend і проксування API-запитів до backend. Reverse proxy дозволяє користувачу працювати з одним доменом або портом, тоді як запити до API перенаправляються до Node.js-сервера [22]. Це спрощує структуру розгортання і наближає її до реальної production-схеми.

Для авторизації використовується класичний email/password-вхід і Google Identity. Google Identity Services дозволяє додати на сайт кнопку входу через Google [24]. Це покращує зручність для користувача, оскільки він може швидко авторизуватися без створення окремого пароля. Водночас сервер повинен перевіряти отримані дані й визначати роль користувача.

Для онлайн-оплати використовується LiqPay. Параметри платежу повинні формуватися на сервері, а не у frontend. LiqPay Checkout API використовує дані платежу та підпис, які мають створюватися із застосуванням приватного ключа [25]. Тому приватний ключ зберігається у змінних середовища, а frontend отримує лише готові дані для переходу до оплати. Це відповідає принципам безпечного зберігання конфігурації [37].

Для доставки використовується Nova Post API. Система повинна отримувати актуальні відділення і поштомати, щоб користувач міг обрати точку доставки без переходу на сторонні сайти [26]. API-ключ Нової пошти також повинен зберігатися на backend, щоб не розкриватися у браузері.

Для майбутньої автоматизації перевірки якості можна використовувати CI/CD. GitHub Actions дозволяє автоматизувати робочі процеси, зокрема запуск тестів, перевірку збірки та деплой [36]. У межах дипломного проєкту повноцінний CI/CD не є обов’язковим, однак архітектура і Docker-конфігурація повинні дозволяти додати його в майбутньому.

З погляду продуктивності обрані технології відповідають вимогам проєкту. React і Vite забезпечують швидку розробку та оптимізовану збірку, Node.js ефективно обробляє короткі запити, а nginx швидко віддає статичні

файли. З погляду масштабованості система може бути розширена шляхом перенесення даних у PostgreSQL, додавання кешування, розширення API або винесення окремих модулів у самостійні сервіси.

З погляду підтримуваності важливим є чітке розділення модулів. Frontend повинен містити окремі компоненти сторінок, карток, форм і адміністративних вкладок. Backend повинен мати окремі маршрути та сервіси для товарів, замовлень, авторизації, доставки й оплати. З погляду безпеки важливими є контроль доступу, серверна обробка платежів, приховування секретних ключів і валідація даних. OWASP Top 10 підкреслює важливість правильного контролю доступу, безпечної конфігурації та захисту від типових вебризиків [23].

Таким чином, обраний технологічний стек є збалансованим для дипломного проєкту. Він не є надмірно складним, але дозволяє реалізувати всі основні вимоги: інтерактивний інтерфейс, API, авторизацію, оплату, доставку, адмін-панель, локальні зображення та контейнерне розгортання.

3.4. Висновки по розділу

У третьому розділі було розглянуто архітектурні рішення та моделі майбутньої системи Apple Shop. На основі аналізу монолітної, мікросервісної та клієнт-серверної архітектури обрано клієнт-серверну модель із React frontend, Node.js backend, REST API, зовнішніми інтеграціями та Docker-розгортанням. Такий підхід забезпечує чіткий поділ відповідальності між інтерфейсом, серверною логікою, даними, зовнішніми сервісами та інфраструктурою запуску.

Було визначено основні рівні системи: клієнтський, серверний, рівень даних, рівень зовнішніх інтеграцій і рівень розгортання. Описано взаємодію React SPA з Node.js API, роль nginx як reverse proxy, використання Docker Compose для запуску системи та можливість майбутнього переходу до PostgreSQL як промислової СУБД. Це робить систему зрозумілою для підтримки, тестування та подальшого розширення.

У розділі змодельовано основні бізнес-процеси Apple Shop: перегляд

каталогу, глобальний пошук, додавання товару до кошика, оформлення замовлення, онлайн-оплату, вибір доставки, Trade In і роботу адміністратора. Для їх візуального представлення запропоновано UML-діаграми варіантів використання, діяльності, послідовності та класів. Також визначено основні frontend-компоненти, backend-модулі та сутності даних: User, Product, ProductVariant, Order, OrderItem, Review і TradeInRequest.

Обґрунтовано вибір технологій та інструментів розробки: React, Vite, Node.js, Express, Docker Compose, nginx, Google Identity, LiqPay, Nova Post API та PostgreSQL як можливу промислову СУБД. Обраний стек відповідає вимогам продуктивності, підтримуваності, безпеки, масштабованості та зручності розгортання.

Отже, третій розділ формує архітектурну й технологічну основу для практичної реалізації системи. Запропонована структура дозволяє перейти від аналізу вимог до розробки програмного коду, організації модулів, реалізації алгоритмів і перевірки працездатності Apple Shop у четвертому розділі.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ПРОЕКТУ

4.1. Опис розробки програмного коду та структури проєкту

Практична реалізація веб-застосунку Apple Shop виконувалася відповідно до архітектурних рішень, описаних у третьому розділі. Основною метою реалізації було створення працездатної клієнт-серверної системи, яка забезпечує перегляд товарів, роботу з каталогами, глобальний пошук, кошик, оформлення замовлень, авторизацію, Trade In, адміністративну панель, інтеграцію з LiqPay, інтеграцію з Новою поштою та контейнерне розгортання. Оскільки система має frontend і backend, структура коду була організована так, щоб кожна частина відповідала за власну зону відповідальності.

Організація репозиторію має важливе значення для підтримуваності програмного забезпечення. У стандарті ISO/IEC 25010 підтримуваність розглядається як одна з характеристик якості програмного продукту [11]. Це означає, що код повинен бути зрозумілим для подальшого редагування, розширення і тестування. Для Apple Shop це особливо важливо, оскільки проєкт містить багато різних модулів: каталог, кошик, замовлення, адміністративну панель, авторизацію, оплату, доставку та завантаження зображень.

Загальна структура проєкту поділена на кілька основних частин: клієнтську частину, серверну частину, deploy-конфігурацію, статичні ресурси, дані для початкового наповнення та службові файли. Такий поділ відповідає клієнт-серверній архітектурі, де frontend відповідає за інтерфейс користувача, а backend - за бізнес-логіку, API та взаємодію із зовнішніми сервісами. Node.js використовується як серверне середовище для обробки HTTP-запитів, що відповідає його призначенню як подієво-орієнтованого JavaScript runtime для мережевих застосунків [19]. Структуру репозиторію Apple Shop наведено на рисунку 4.1.

Рисунок 4.1 – Структура репозиторію Apple Shop

Клієнтська частина реалізована на React. React дозволяє будувати інтерфейс із компонентів, які можуть повторно використовуватися на різних сторінках [17]. У межах Apple Shop це використовується для товарних карток, верхнього меню, сітки товарів, форм, вибору кольору, повідомлень, вкладок адміністративної панелі та інших елементів. Компонентний підхід зменшує дублювання коду, оскільки одна й та сама картка товару може використовуватися на головній сторінці, у каталозі, у розділі аксесуарів, у БУ-товарах і в результатах пошуку.

Frontend-частина має сторінкову і компонентну організацію. Сторінки відповідають маршрутам застосунку, а компоненти є меншими повторно використовуваними частинами інтерфейсу. До сторінок належать HomePage, CatalogPage, UsedProductsPage, AccessoriesPage, ProductPage, SearchResultsPage, CartPage, CheckoutPage, OrdersPage, TradeInPage, LoginPage, SignupPage та AdminPage. До компонентів належать Header, Footer, ProductCard, ProductGrid, CategoryTabs, ColorSelector, ToastMessage, HeroSlider, AdminTabs, FormInput та інші.

Серверна частина реалізована на Node.js з використанням Express. Express є мінімалістичним і гнучким вебфреймворком для Node.js, який підтримує маршрутизацію та middleware [20]. У backend-частині маршрути поділено за функціональними напрямами: товари, замовлення, авторизація, відгуки, Trade In, завантаження файлів, LiqPay, Нова пошта та адміністративні дії. Така організація дозволяє швидко знайти потрібний фрагмент коду й не змішувати різні бізнес-процеси в одному файлі.

Під час реалізації важливо було дотримуватися єдиних стандартів найменувань. Для React-компонентів доцільно використовувати PascalCase, наприклад ProductCard, CartPage, AdminTabs. Для функцій і змінних використовується camelCase, наприклад formatPrice, getProductImage, selectedColor, cartItems. Для маршрутів API використовується kebab-case або зрозумілі REST-подібні шляхи, наприклад /api/products, /api/orders, /api/trade-in. Для змінних середовища використовується UPPER_SNAKE_CASE, наприклад LIQPAY_PUBLIC_KEY, LIQPAY_PRIVATE_KEY, NOVA_POSHTA_API_KEY, GOOGLE_CLIENT_ID.

Окрему увагу приділено конфігурації. Секретні ключі не повинні зберігатися у frontend-коді або відкрито публікуватися в репозиторії. The Twelve-Factor App рекомендує відокремлювати конфігурацію від коду та зберігати її у змінних середовища [37]. Саме тому ключі LiqPay, Нової пошти та Google-авторизації повинні зчитуватися backend-частиною з .env. У публічний репозиторій доцільно додавати лише приклад конфігурації без реальних секретів.

Також було оновлено .gitignore, щоб у репозиторій не потрапляли зайві файли. До таких файлів належать node_modules, build-артефакти, кеші, локальні .env, тимчасові файли IDE, системні файли операційної системи та застарілі файли попередньої Ruby-реалізації. Це важливо не лише для чистоти репозиторію, а й для безпеки, оскільки локальні конфігурації можуть містити приватні ключі. OWASP Top 10 підкреслює ризики, пов’язані з неправильною конфігурацією безпеки та розкриттям чутливих даних [23].

Клієнтський код організовано так, щоб кожна сторінка відповідала за

сценарій користувача, а спільна логіка виносилася в окремі функції або сервіси. Наприклад, логіка отримання товарів із backend може бути винесена в API-service, логіка форматування ціни - у utility-функцію, логіка кошика - у контекст або окремий модуль стану. Такий підхід зменшує зв’язність між компонентами й полегшує тестування окремих частин.

Для кошика може використовуватися локальне збереження стану у браузері. Web Storage API дозволяє зберігати пари ключ-значення для певного origin, а localStorage зберігає дані між сесіями браузера [44]. У межах Apple Shop це корисно для збереження товарів у кошику, якщо користувач оновив сторінку або тимчасово закрив вкладку. Водночас у localStorage не можна зберігати секретні ключі або критичні персональні дані, оскільки це клієнтське сховище.

Серверний код організовано за модульним принципом. Для кожної бізнес-області створюється окремий набір маршрутів і функцій обробки. Наприклад, модуль товарів відповідає за отримання списку товарів, пошук, перегляд конкретного товару, додавання та редагування. Модуль замовлень відповідає за створення замовлення, перегляд замовлень користувача, перегляд замовлень адміністратором і зміну статусу. Модуль Trade In відповідає за створення заявок і перегляд їх в адміністративній панелі.

Під час реалізації інтерфейсу було враховано вимоги до єдиного стилю. Усі ціни повинні відображатися з символом гривні «₴», а не в різних форматах. Картки товарів на головній сторінці, у каталозі, БУ-розділі та аксесуарах повинні мати однакову структуру. Це зменшує візуальну різнорідність і робить сайт більш професійним. Зручність використання є однією з характеристик якості програмного продукту за ISO/IEC 25010 [11].

Особливістю реалізації стало опрацювання зображень товарів. Для частини товарів використовуються статичні зображення з каталогу проєкту, а для товарів, які додає адміністратор, передбачено локальне завантаження файлів. При цьому шлях до зображення повинен бути збережений у даних товару. Якщо товар має варіанти кольору, кожен варіант може мати власне зображення. Це дозволяє при виборі кольору на сторінці товару змінювати фото без

перезавантаження сторінки. Сторінка товару з вибором кольору наведено на рисунку 4.2.

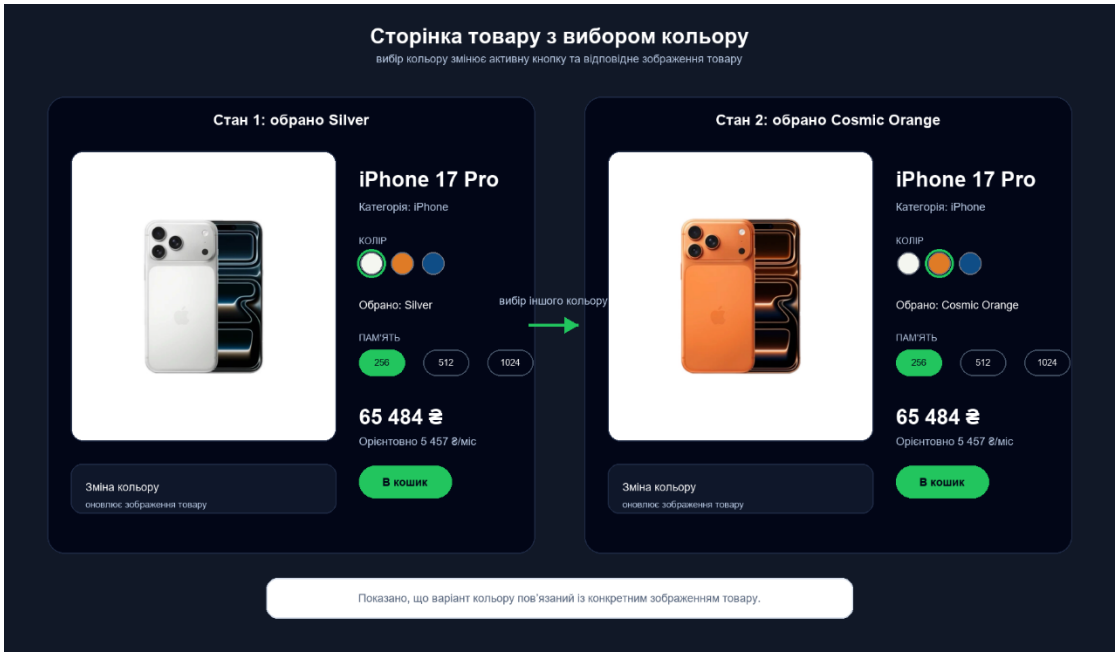

Рисунок 4.2 – Сторінка товару з вибором кольору

Окремо реалізовано адміністративну панель. У попередньому варіанті панель могла бути довгою сторінкою з великою прокруткою, що ускладнювало роботу адміністратора. У фінальній логіці вона поділяється на окремі вкладки або кнопки: товари, відгуки, Trade In і замовлення. Це відповідає евристикам зручності Nielsen Norman Group, зокрема принципам узгодженості, видимості стану системи та зменшення зайвого навантаження на користувача [32]. Адміністративну панель Apple Shop із вкладками для керування товарами, замовленнями, відгуками та Trade In-заявками наведено на рисунку 4.3.

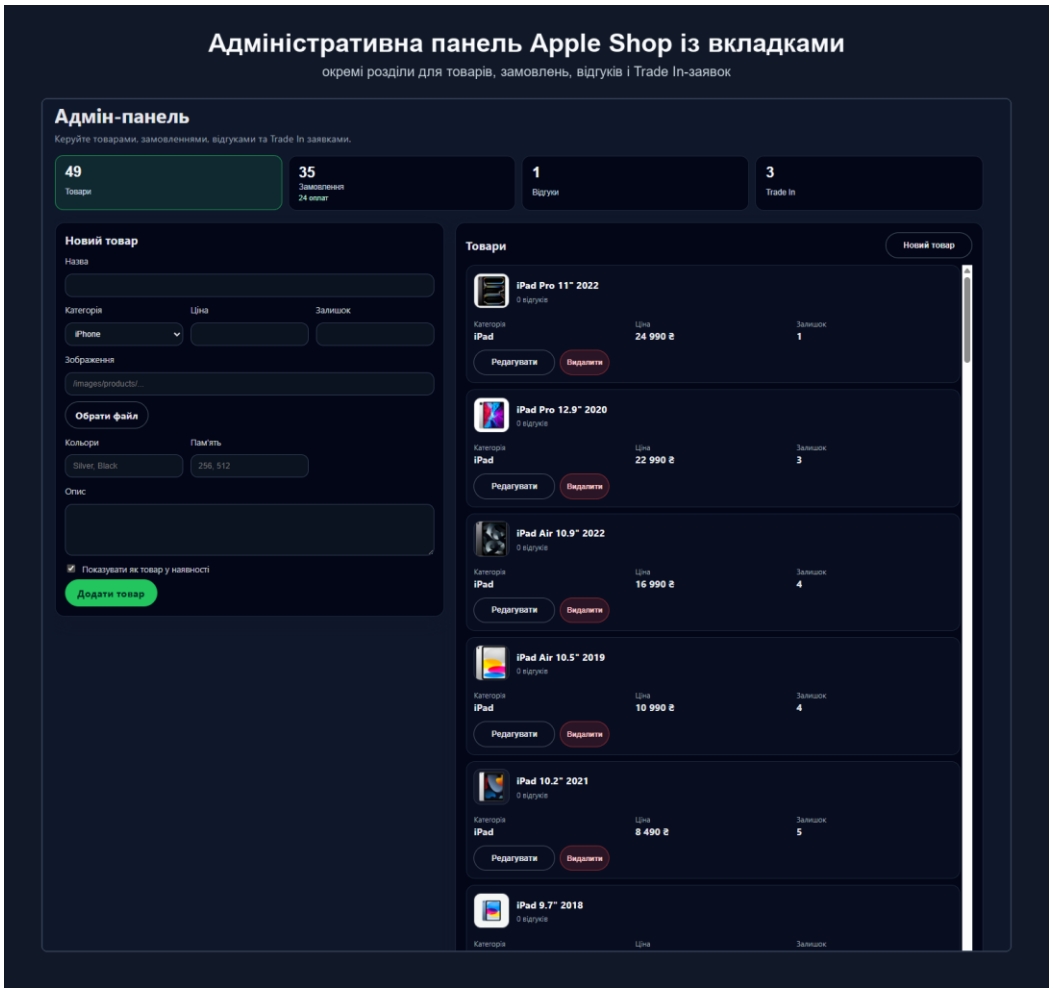

Рисунок 4.3 - Адміністративна панель Apple Shop із вкладками.

У підсумку, практична організація коду Apple Shop базується на модульності, розділенні відповідальності та єдиному стилі реалізації. Такий підхід дає змогу розвивати систему поступово: додавати нові категорії товарів, змінювати дизайн карток, розширювати адміністративну панель або підключати нові інтеграції без повного переписування проєкту.

4.2. Використані алгоритми, структури даних та логіка обробки товарів

У реалізації Apple Shop використано набір алгоритмів і структур даних, які забезпечують роботу каталогу, пошуку, варіантів товарів, кошика, замовлень, Trade In, адміністративної панелі та зовнішніх інтеграцій. Більшість алгоритмів

не є складними математично, однак вони важливі для коректної бізнес-логіки: система повинна зберігати вибір користувача, правильно передавати дані між сторінками та забезпечувати передбачувану поведінку інтерфейсу.

Базовою структурою даних є товар. Він містить ідентифікатор, назву, категорію, опис, ціну, залишок, зображення, характеристики та, за потреби, масив варіантів. Для нових пристроїв варіанти можуть описувати різні кольори або обсяг пам’яті. Для БУ-товарів пам’ять, колір і стан є фіксованими характеристиками конкретної одиниці товару. Для аксесуарів варіанти найчастіше описують колір і відповідне зображення.

Однією з важливих задач є вибір правильного зображення товару. Якщо користувач не обрав конкретний варіант, система показує основне зображення товару. Якщо вибрано колір або інший варіант, система перевіряє, чи має цей варіант власне зображення, і відображає саме його. Якщо зображення варіанта відсутнє, використовується основне зображення товару. За умови зберігання варіантів у масиві складність пошуку становить O(k), де k - кількість варіантів товару. Оскільки кількість кольорів зазвичай невелика, такий підхід є достатнім.

Глобальний пошук реалізується як пошук по всіх товарних групах: новій техніці, БУ-товарах і аксесуарах. Для цього товари об’єднуються в єдину колекцію, після чого виконується фільтрація за пошуковим запитом. Пошук може враховувати назву, категорію, опис, модель, пам’ять, колір або інші характеристики. Перед порівнянням рядки доцільно нормалізувати: перевести у нижній регістр і прибрати зайві пробіли. Складність простого пошуку становить O(n), що є прийнятним для демонстраційного каталогу. У майбутньому для більшого обсягу даних можна використати індексацію або повнотекстовий пошук у PostgreSQL [38].

Фільтрація за категорією працює схожим чином. Якщо користувач обирає категорію, система показує лише товари з відповідним значенням. Якщо вибрано категорію «Усе», повертається повний список. Окремо реалізується форматування ціни: усі ціни повинні відображатися в єдиному форматі із символом «₴», наприклад «65 484 ₴». Винесення цієї логіки в окрему функцію

дозволяє уникнути різного написання цін у різних частинах сайту.

Кошик є однією з ключових структур даних у системі. Він може бути представлений як масив об’єктів CartItem. Кожен елемент містить productId, назву, ціну, кількість, вибраний колір, пам’ять, зображення варіанта, категорію та інші параметри. Важливо, що в кошику зберігається не просто товар, а конкретний вибраний варіант. Якщо користувач додає чохол у кольорі Clay, саме цей колір і відповідне зображення повинні відобразитися в кошику та замовленні.

Додавання товару до кошика виконується за таким принципом: система перевіряє, чи є вже позиція з таким самим productId і таким самим варіантом. Якщо така позиція є, збільшується кількість. Якщо немає - створюється новий елемент кошика. Для порівняння потрібно враховувати не лише ідентифікатор товару, а й колір або інші параметри варіанта, інакше товари різних кольорів можуть помилково об’єднатися в одну позицію. У випадку масиву складність операції становить O(m), де m - кількість позицій у кошику, що є прийнятним для типового користувацького сценарію.

Створення замовлення виконується на основі поточного стану кошика. Система формує об’єкт Order, який містить дату, користувача або контактні дані, список товарів, суму, спосіб доставки, вибране відділення або поштомат, спосіб оплати і статус. Якщо користувач обирає LiqPay, платіжні дані формуються на backend, оскільки приватний ключ платіжної системи не повинен передаватися у frontend [25]. Клієнтська частина отримує лише готові дані для переходу до оплати.

Алгоритм роботи з Новою поштою передбачає звернення backend до Nova Post API. Користувач вводить або вибирає місто, після чого сервер отримує список відділень або поштоматів [26]. Далі frontend показує ці точки у списку або на карті, а вибрана точка доставки зберігається в замовленні. Завдяки цьому система використовує актуальні дані служби доставки, а не статичний список адрес. Вибір відділення Нової пошти під час оформлення замовлення наведено на рисунку 4.4.

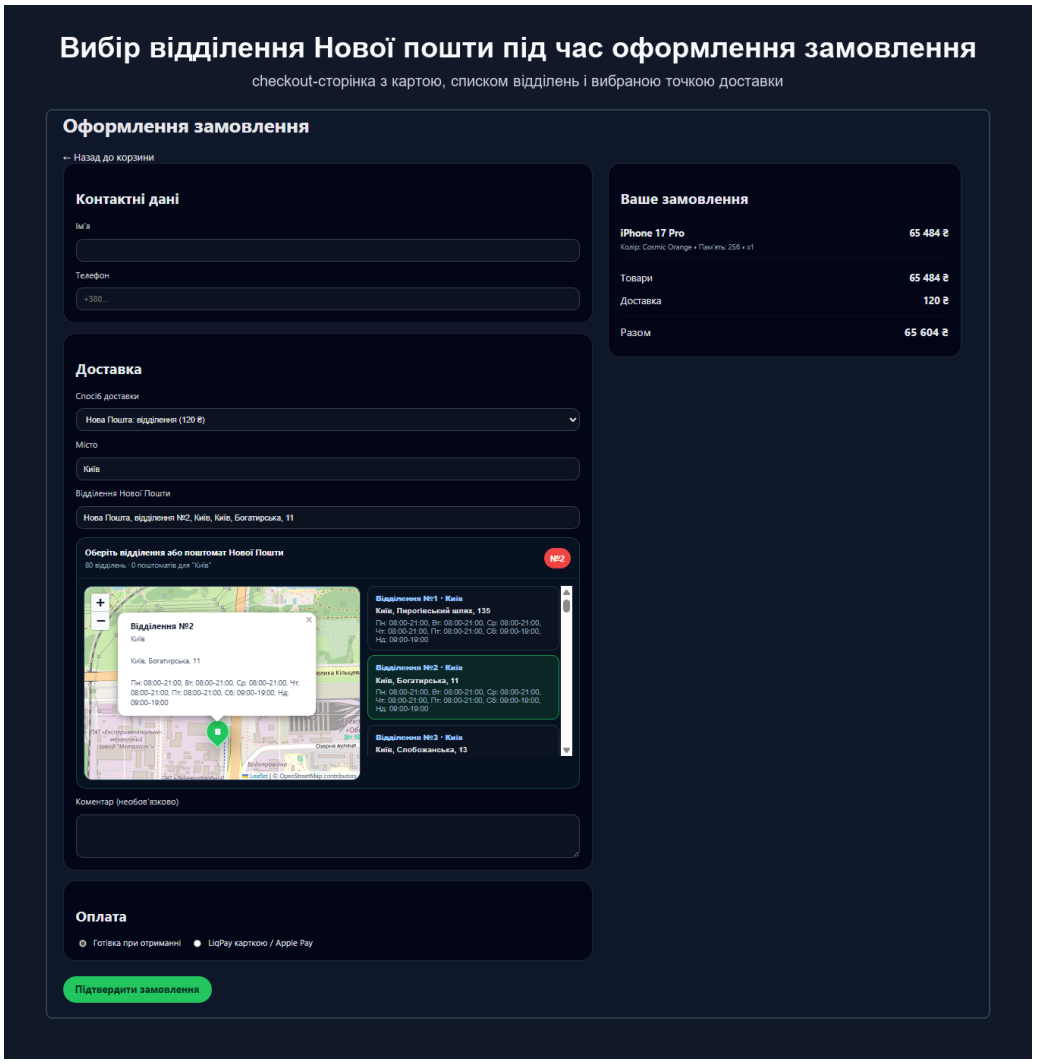

Рисунок 4.4 – Вибір відділення Нової пошти під час оформлення замовлення

Trade In-заявка є окремою структурою даних. Вона містить userId, модель старого пристрою, стан, бажаний новий пристрій, контактні дані, коментар і дату створення. Перед створенням заявки система перевіряє, чи користувач авторизований. Якщо ні, заявка не створюється. Така перевірка реалізує бізнес-вимогу, згідно з якою Trade In не повинен бути анонімним процесом.

Для адміністративної панелі реалізовано CRUD-операції над товарами. Адміністратор може створити товар, переглянути список, змінити характеристики, оновити зображення або видалити недоречні дані. Для відгуків передбачено видалення, для Trade In - перегляд заявок, а для замовлень - зміна статусу оплати. Також у системі використовуються тимчасові повідомлення:

після додавання товару до кошика або надсилання заявки повідомлення показується користувачу і автоматично зникає через кілька секунд.

Таким чином, реалізовані алгоритми Apple Shop мають переважно лінійну складність, що є прийнятним для дипломного проєкту і демонстраційного обсягу даних. Водночас структура системи залишає можливість подальшої оптимізації: перехід до бази даних, індексація пошуку, використання Map для кошика, кешування зовнішніх API-запитів або винесення окремих модулів у самостійні сервіси.

4.3. Реалізація основних модулів, інтеграцій та адміністративної панелі

У межах практичної реалізації Apple Shop було створено основні модулі, які забезпечують роботу інтернет-магазину: каталог товарів, глобальний пошук, сторінку товару, кошик, checkout, авторизацію, Trade In, зовнішні інтеграції та адміністративну панель. Кожен модуль має власну функціональну роль, але всі вони взаємодіють між собою через спільні дані, API-запити та стан клієнтського застосунку.

Публічна частина сайту поділена на кілька основних розділів: каталог нової Apple-техніки, БУ-товари та аксесуари. Для нових товарів реалізовано вибір пам’яті й кольору, для БУ-товарів характеристики відображаються як фіксовані, а для аксесуарів передбачено поділ на категорії та зміну зображення залежно від вибраного кольору. Такий підхід дозволяє врахувати особливості різних груп товарів і зробити інтерфейс зрозумілішим для користувача.

Для відображення товарів використовується єдиний підхід до побудови карток. У картці подаються зображення, назва, категорія, ціна та короткі характеристики товару. Це дає змогу зберігати однаковий стиль на головній сторінці, у каталозі, розділі БУ-техніки, аксесуарах і результатах пошуку. Уніфікація товарних карток спрощує супровід frontend-частини, оскільки зміни у стилі або логіці відображення можна вносити централізовано.

Важливим елементом системи є глобальний пошук. Він працює не лише по основному каталогу, а й по БУ-техніці та аксесуарах. Завдяки цьому користувач може знайти потрібний товар незалежно від того, у якому розділі він розміщений. Це особливо важливо для інтернет-магазину з кількома товарними групами, оскільки покупець не завжди знає точну категорію потрібної позиції.

Модуль сторінки товару відповідає за відображення детальної інформації: назви, ціни, опису, категорії, зображення та доступних характеристик. Для товарів із кількома варіантами реалізовано зміну зображення при виборі кольору. Для БУ-товарів пам’ять і колір не подаються як варіанти вибору, оскільки такий товар уже існує в конкретній конфігурації. Це відповідає реальній логіці продажу вживаної техніки.

Окремо реалізовано модуль кошика. Він зберігає не лише назву, ціну та кількість товару, а й конкретний вибраний варіант: колір, пам’ять і відповідне зображення. Це дає змогу уникнути ситуації, коли користувач обирає один варіант товару, а в кошику або замовленні відображається інший. Відображення товару конкретного кольору в кошику наведено на рисунку 4.5.

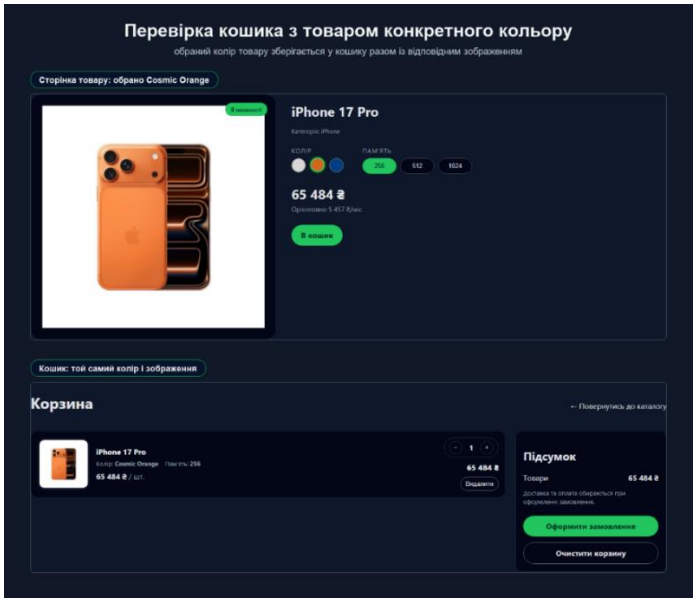

Рисунок 4.5 – Відображення товару конкретного кольору в кошику

Модуль checkout забезпечує оформлення замовлення. Користувач вводить

контактні дані, вибирає спосіб доставки та оплати. Для доставки передбачено самовивіз із магазину та доставку Новою поштою. Якщо користувач обирає самовивіз, система не вимагає введення міста, адреси або відділення. Якщо обрано Нову пошту, користувач може вибрати відділення або поштомат, що отримуються через Nova Post API.

Під час створення замовлення важливо передати на backend не лише загальну суму, а й склад кошика, контактні дані, спосіб доставки, спосіб оплати та статус замовлення. Це дозволяє адміністратору надалі переглядати замовлення, бачити вибрані товари й керувати їх обробкою. Для замовлень, які очікують оплати, передбачено можливість адміністративного підтвердження платежу.

Для онлайн-оплати реалізовано інтеграцію з LiqPay. Платіжні параметри формуються на backend, оскільки приватні ключі не повинні зберігатися у frontend-частині. Також реалізовано авторизацію через email/password і Google-акаунт. Google-вхід спрощує доступ користувача до персональних функцій, зокрема до оформлення замовлень і Trade In-заявок.

Trade In-модуль дозволяє користувачу подати заявку на обмін старого iPhone на новий. У формі враховуються модель старого пристрою, пам’ять, стан, рівень акумулятора та модель нового iPhone. Ціна нового пристрою залежить від вибраної пам’яті й береться відповідно до каталогу. Неавторизований користувач не може надіслати заявку, що дозволяє прив’язати звернення до конкретного облікового запису.

Адміністративна панель реалізована у вигляді окремих розділів для товарів, відгуків, Trade In-заявок і замовлень. Адміністратор може додавати й редагувати товари, завантажувати локальні зображення, видаляти відгуки, переглядати заявки на обмін техніки та підтверджувати оплату замовлень. Поділ панелі на вкладки робить її зручнішою, оскільки адміністратору не потрібно працювати з однією довгою сторінкою.

Окремо важливою є можливість локального керування зображеннями товарів. Адміністратор може завантажувати файли через адміністративну панель,

а система зберігає шлях до них і використовує ці зображення у каталозі, на сторінці товару та в кошику. Це робить наповнення магазину гнучкішим, оскільки для оновлення фотографій не потрібно безпосередньо змінювати програмний код. Приклади реалізованого коду основних модулів Apple Shop наведено в додатку А.

Таким чином, реалізовані модулі забезпечують основну бізнес-логіку Apple Shop. Публічна частина відповідає за взаємодію з покупцем, інтеграції забезпечують оплату, доставку й авторизацію, а адміністративна панель дає змогу керувати товарами, відгуками, заявками та замовленнями. Це робить систему придатною для демонстрації, тестування й подальшого розвитку.

4.4. Висновки по розділу

У четвертому розділі було описано практичну реалізацію веб-застосунку Apple Shop. Розглянуто структуру репозиторію, організацію frontend- і backend-частин, роботу з конфігураційними файлами, локальними зображеннями, адміністративною панеллю та deploy-оточенням. Визначено, що система має чітке розділення відповідальності: клієнтська частина відповідає за інтерфейс, сторінки сайту та взаємодію з користувачем, а серверна частина забезпечує API, бізнес-логіку, авторизацію, збереження даних і роботу із зовнішніми сервісами.

Було проаналізовано основні алгоритми й структури даних, використані у проєкті. До них належать глобальний пошук по всіх розділах сайту, фільтрація товарів за категоріями, вибір зображення залежно від кольору, додавання товару до кошика з урахуванням характеристик, підрахунок вартості, створення замовлення, формування LiqPay-платежу, отримання відділень Нової пошти, обробка Trade In-заявок і CRUD-операції адміністратора. Такі рішення забезпечують повний цикл роботи інтернет-магазину: від перегляду товару до оформлення покупки.

Окрему увагу приділено реалізації товарних варіантів і локальних зображень. Для товарів з різними кольорами система зберігає вибраний варіант і

відображає відповідне зображення на сторінці товару, у кошику та в замовленні. Це особливо важливо для Apple-техніки й аксесуарів, де колір є суттєвою характеристикою товару. Завдяки цьому користувач бачить саме той варіант, який обрав перед додаванням до кошика.

Також описано адміністративну панель, яка побудована не як одна довга сторінка, а як набір окремих розділів для товарів, замовлень, відгуків і Trade In-заявок. Адміністратор може додавати й редагувати товари, завантажувати локальні зображення, видаляти відгуки, переглядати заявки на обмін техніки та підтверджувати оплату замовлень, що очікують оплати. Така структура робить керування магазином зручнішим і зрозумілішим.

У розділі також було розглянуто інтеграцію із зовнішніми сервісами. Google Identity використовується для авторизації користувачів через Google-акаунт, LiqPay - для підготовки онлайн-оплати, а Nova Post API - для вибору відділення або поштомата під час оформлення замовлення. Наявність цих інтеграцій наближає прототип до реального комерційного вебзастосунку.

У результаті реалізації було створено функціональний прототип інтернет-магазину Apple-техніки, який підтримує каталоги нових і БУ-товарів, аксесуари, глобальний пошук, сторінку товару, кошик, checkout, авторизацію, Trade In, оплату через LiqPay, доставку Новою поштою та адміністративне керування. Реалізована структура може бути розширена у майбутньому шляхом підключення повноцінної бази даних, CI/CD, промокодів, аналітики продажів і додаткових способів доставки.

Отримані результати четвертого розділу підтверджують практичну реалізованість обраної архітектури та технологічного стеку. Вони є основою для подальшого тестування системи, перевірки її працездатності та оцінки готовності до впровадження, що розглядається у п’ятому розділі.

РОЗДІЛ 5. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ

5.1. Стратегії та методи тестування веб-застосунку

Тестування веб-застосунку Apple Shop виконувалося з метою перевірки відповідності системи функціональним і нефункціональним вимогам, сформульованим у другому розділі. Оскільки проєкт містить frontend, backend, адміністративну панель, кошик, замовлення, авторизацію та зовнішні інтеграції, перевірка не може обмежуватися лише ручним переглядом сторінок. Необхідно оцінити роботу окремих модулів, взаємодію між клієнтською і серверною частинами, коректність бізнес-логіки, безпеку доступу та готовність системи до запуску через Docker.

Загальна стратегія тестування поєднує ручний та автоматизований підходи. Ручне тестування застосовується для перевірки візуального вигляду сторінок, навігації, відображення зображень, роботи форм і адміністративної панелі. Автоматизоване тестування доцільне для повторюваних сценаріїв: форматування ціни, пошуку, додавання товару до кошика, створення замовлення, перевірки API та основних користувацьких шляхів. ISTQB визначає тестування як процес оцінювання якості програмного продукту та зниження ризиків через виявлення дефектів [49].

У межах Apple Shop використовуються кілька рівнів перевірки: модульне, компонентне, API, end-to-end, навантажувальне, безпекове, accessibility та deployment-тестування. Модульне тестування перевіряє окремі функції, компонентне - поведінку React-компонентів, API-тестування - маршрути backend, E2E-тестування - повні сценарії користувача, а deployment-тестування - запуск системи в контейнерному середовищі.

Для модульного тестування JavaScript-функцій може використовуватися Vitest, який підтримує assertions, mock-функції та інтеграцію з Vite-проєктами [39]. У Apple Shop такі тести доцільні для перевірки форматування ціни, нормалізації пошуку, вибору зображення товарного варіанта, підрахунку суми

кошика та логіки додавання товару. Для тестування React-компонентів може використовуватися Testing Library, яка перевіряє інтерфейс через дії, близькі до поведінки реального користувача [40].

Для end-to-end тестування можна використовувати Selenium WebDriver або Playwright. Selenium WebDriver забезпечує автоматизацію браузера [46], а Playwright призначений для перевірки сучасних веб-застосунків у браузері [41]. У межах Apple Shop E2E-тести мають перевіряти повний шлях користувача: відкриття головної сторінки, пошук товару, вибір кольору, додавання до кошика, оформлення замовлення та перевірку статусу.

API-тестування може виконуватися через Postman або автоматизовані backend-тести. Postman дозволяє створювати запити до API, додавати test scripts і запускати перевірки відповідей [50]. Для Apple Shop API-тестування охоплює маршрути товарів, пошуку, замовлень, авторизації, Trade In, LiqPay і Нової пошти. Наприклад, потрібно перевірити, що пошук повертає товари з усіх категорій, а створення Trade In-заявки недоступне без авторизації.

Для навантажувального тестування можуть застосовуватися Apache JMeter або Grafana k6. JMeter є відкритим інструментом для перевірки навантаження [45], а k6 використовується для performance-тестування, зокрема stress і spike tests [48]. У Apple Shop такі перевірки доцільні для основних API-запитів: отримання каталогу, пошуку товарів, створення замовлення та роботи з адміністративними даними.

Для оцінки frontend-частини може використовуватися Lighthouse, який містить аудити продуктивності, доступності, best practices і SEO [47]. Це корисно для Apple Shop, оскільки система має багато зображень, товарних карток і динамічних сторінок. Додатково тестування безпеки виконується з урахуванням OWASP Top 10: перевіряється контроль доступу до адміністративної панелі, заборона Trade In-заявок без авторизації, серверне формування LiqPay-платежу та збереження секретних ключів у змінних середовища [23].

Схему рівнів тестування Apple Shop наведено на рисунку 5.1.

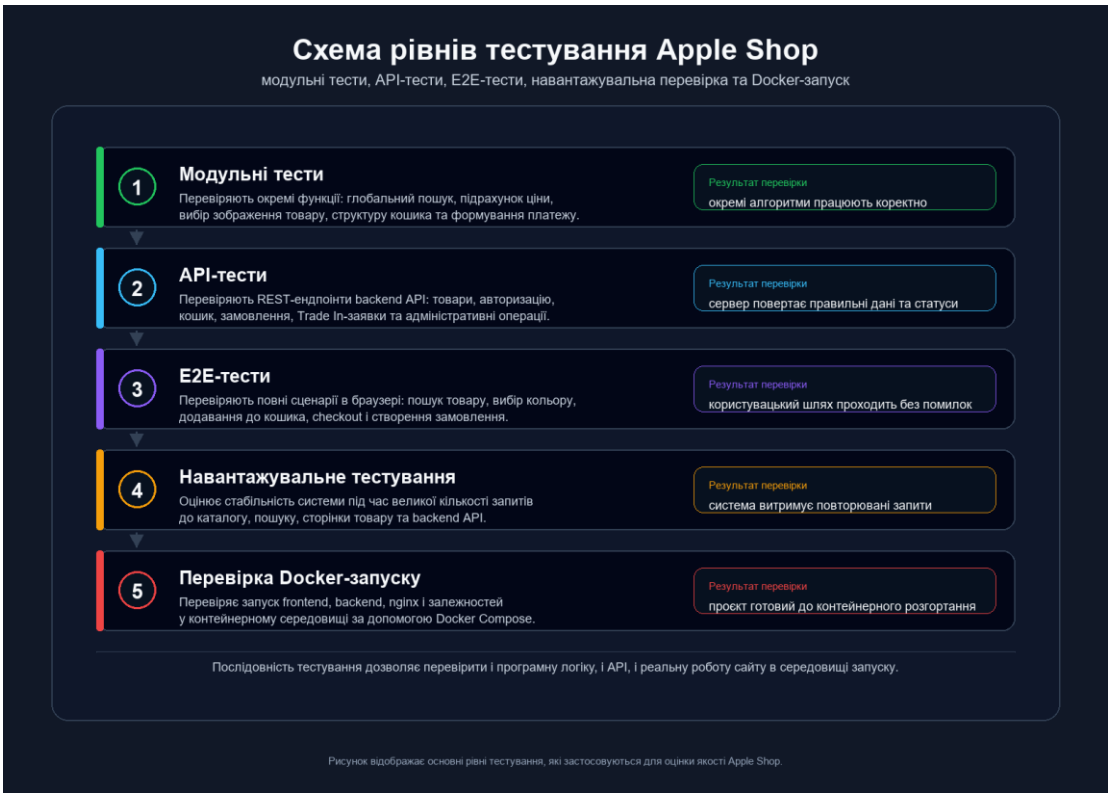

Рисунок 5.1 – Схема рівнів тестування Apple Shop

5.2. Результати тестування системи та роботи її сервісів

Під час тестування Apple Shop було перевірено основні функції системи: публічні сторінки, каталог, пошук, сторінку товару, кошик, checkout, авторизацію, Trade In, адміністративну панель, зовнішні інтеграції та Docker-запуск. Тестування охоплювало не лише окремі елементи інтерфейсу, а й повні користувацькі сценарії.

Загалом було сформовано 82 тестові випадки. Після первинної перевірки успішно виконано 71 тест, а 11 потребували виправлення. Після усунення недоліків успішно виконано 80 тестів із 82, що становить 97,6 %. Два випадки залишилися залежними від зовнішніх умов: повна перевірка реального LiqPay-платежу та стабільність відповіді API Нової пошти.

Результати, наведені в таблиці 5.1, підтверджують, що основні сценарії роботи Apple Shop виконуються коректно та можуть бути використані для оцінки ефективності впровадження системи.

Таблиця 5.1

Основні результати функціонального тестування Apple Shop

№	Сценарій	Очікуваний результат	Результат
1	Відкриття головної сторінки	Відображається слайдер, добірки товарів і категорії	Виконано
2	Глобальний пошук товарів	Пошук знаходить товари з каталогу, БУ-техніки та аксесуарів	Виконано
3	Вибір варіанта товару	При виборі кольору або пам’яті змінюються відповідні дані товару	Виконано
4	Додавання товару до кошика	У кошику зберігаються назва, ціна, кількість, колір і зображення товару	Виконано
5	Оформлення замовлення	Користувач може ввести контактні дані, вибрати доставку й оплату	Виконано
6	Доставка Новою поштою	Система дозволяє вибрати відділення або поштомат	Виконано
7	Формування LiqPay-платежу	Backend формує параметри для переходу до оплати	Виконано
8	Подання Trade In-заявки	Неавторизований користувач блокується, авторизований може створити заявку	Виконано
9	Робота адміністративної панелі	Адміністратор керує товарами, відгуками, заявками та замовленнями	Виконано
10	Docker-запуск системи	Frontend, backend і nginx запускаються через Docker Compose	Виконано

Функціональне тестування показало, що головна сторінка, каталоги нової техніки, БУ-товарів, аксесуарів, кошик, checkout, Trade In, сторінка входу, сторінка замовлень і адміністративна панель відкриваються коректно. Глобальний пошук працює по всіх групах товарів, зокрема по основному каталогу, БУ-техніці та аксесуарах.

Окремо було перевірено сторінку товару й кошик. Для нових товарів перевірялися варіанти кольору та пам’яті, для БУ-товарів - фіксовані характеристики, а для аксесуарів - зміна зображення при виборі кольору. У кошику зберігаються вибраний колір, пам’ять, ціна та відповідне зображення товару.

Checkout перевірявся з різними типами доставки та оплати. Для Нової пошти перевірено вибір відділення або поштомата, а для LiqPay - формування параметрів платежу на backend [25]. Авторизація тестувалася через email/password і Google-вхід [24]. Також перевірено, що Trade In-заявку може

подати лише авторизований користувач.

Адміністративна панель тестувалася за розділами товарів, відгуків, Trade In і замовлень. Адміністратор може додавати й редагувати товари, видаляти відгуки, переглядати заявки та підтверджувати оплату замовлення. Приклад перевірки розділу замовлень адміністративної панелі наведено на рисунку 5.2.

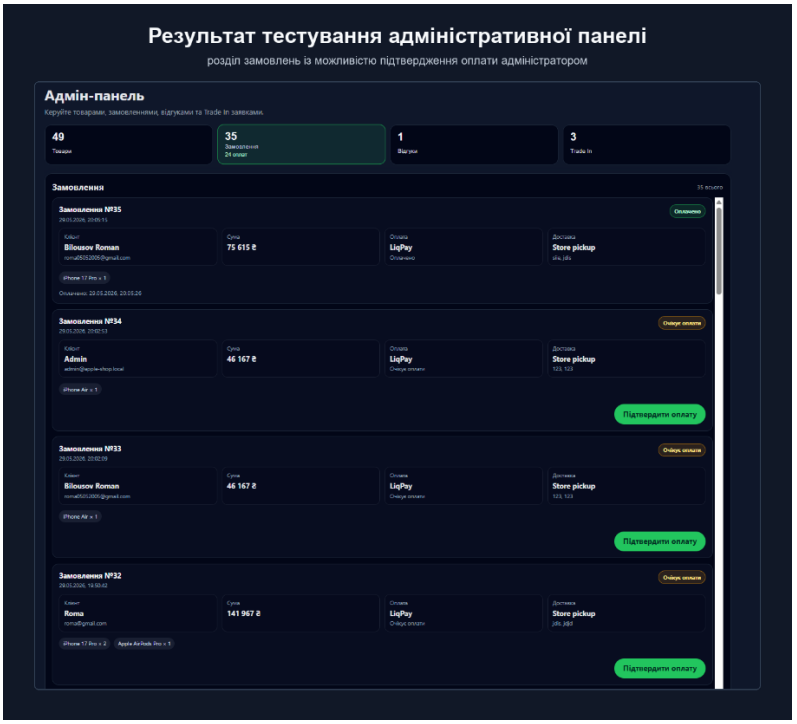

Рисунок 5.2 – Результат тестування адміністративної панелі

Під час тестування інтерфейсу було виявлено й виправлено недоліки: горизонтальну прокрутку, дублювання бейджів, некоректне відображення зображень, смикання сторінки при зміні кольору та неповну логіку глобального пошуку. Після виправлення інтерфейс став стабільнішим і зручнішим.

Також було виконано базову перевірку продуктивності та розгортання. Для навантажувального тестування можуть застосовуватися JMeter або k6 [45], [48], а якість frontend-частини може перевірятися через Lighthouse [47]. Docker-тестування показало, що frontend, backend і nginx можуть запускатися як окремі сервіси через Docker Compose [21], [22].

Отже, результати тестування підтвердили працездатність основних

сценаріїв Apple Shop. Система виконує ключові функції інтернет-магазину, а виявлені недоліки були переважно пов’язані з інтерфейсом, передаванням варіантів товару та зовнішніми інтеграціями.

5.3. Аналіз ефективності впровадження

Впровадження Apple Shop передбачає підготовку середовища запуску, налаштування змінних середовища, запуск frontend- і backend-частин, перевірку API, підключення зовнішніх сервісів, наповнення каталогу товарами та підготовку адміністратора до роботи із системою. Оскільки проєкт підтримує Docker-розгортання, базовий запуск може виконуватися швидше, ніж у випадку ручного встановлення всіх залежностей. Docker Compose дозволяє запускати кілька сервісів як єдину систему [21].

Підготовка інфраструктури включає встановлення Node.js, Docker, налаштування .env, перевірку портів, підготовку зображень товарів і запуск контейнерів. У production-середовищі додатково потрібні домен, HTTPS, резервне копіювання, журналювання, моніторинг і стабільна база даних. У межах дипломного проєкту система розглядається як демонстраційний прототип, але її архітектура дозволяє надалі перейти до промисловішої конфігурації.

Міграція даних у поточному проєкті полягає у перенесенні товарів, категорій, аксесуарів, БУ-пристроїв, слайдів і зображень у єдину структуру. Для нових товарів зберігаються доступні варіанти, для БУ-товарів пам’ять і колір є фіксованими характеристиками, а для аксесуарів із кольорами кожен варіант має відповідне зображення. У майбутньому при переході на PostgreSQL ці дані можуть бути перенесені у таблиці Product, ProductVariant, Order, User, Review і TradeInRequest [38].

Ефективність впровадження можна оцінити за кількома критеріями: зручність пошуку товару, швидкість оформлення замовлення, зменшення кількості ручних дій адміністратора, можливість самостійного оновлення каталогу, прозорість статусів замовлень і готовність до подальшого розвитку.

Для користувача це проявляється у глобальному пошуку, правильному відображенні товарних варіантів, єдиному кошику та зручному checkout. Для адміністратора - у можливості керувати товарами, відгуками, заявками Trade In і замовленнями без редагування коду.

Позитивним результатом впровадження є також покращення візуальної цілісності сайту. Після виправлень товарні картки мають узгоджений вигляд, ціни відображаються в одному форматі, зображення не створюють горизонтальної прокрутки, а головна сторінка містить слайдер, добірки товарів і категорії. Підсумковий вигляд головної сторінки після впровадження наведено на рисунку 5.3.

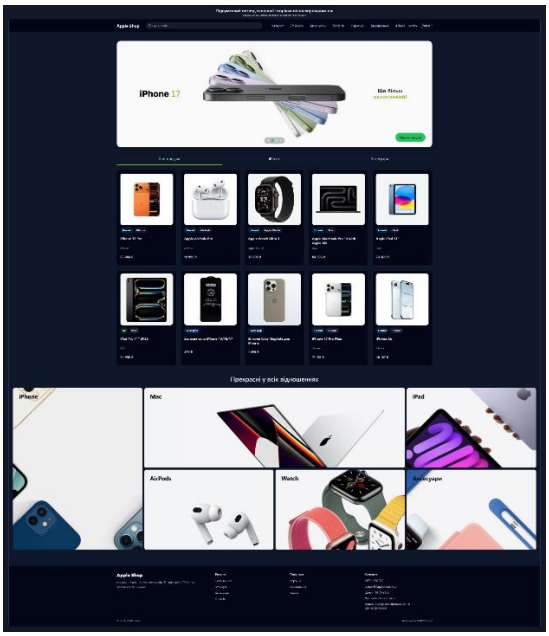

Рисунок 5.3 – Підсумковий вигляд головної сторінки після впровадження

Водночас для реального впровадження потрібно врахувати певні обмеження. Production-версія потребує повноцінної бази даних, резервного копіювання, HTTPS, стабільного домену, коректних callback URL для LiqPay [25], правильного origin для Google-авторизації [24] та врахування залежності від API Нової пошти [26]. Також необхідно посилити безпеку: контроль доступу, журналювання адміністративних дій, обмеження частоти запитів і перевірку

введених даних відповідно до рекомендацій OWASP Top 10 [23].

Загалом ефективність впровадження Apple Shop можна оцінити як достатню для демонстраційного та навчального використання. Система реалізує повний цикл електронної комерції: перегляд товарів, пошук, вибір варіанта, кошик, замовлення, доставку, оплату, авторизацію, Trade In і адміністрування. Крім того, проєкт має зрозумілу архітектуру, що дозволяє розширювати його у майбутньому.

5.4. Висновки по розділу

У п’ятому розділі було розглянуто системне тестування та впровадження вебзастосунку Apple Shop. Визначено загальну стратегію тестування, яка охоплює модульне, компонентне, API, end-to-end, навантажувальне, безпекове, accessibility та deployment-тестування. Такий підхід дозволяє перевірити систему як на рівні окремих функцій, так і на рівні повних користувацьких сценаріїв: від перегляду каталогу до оформлення замовлення й роботи адміністратора.

Було сформовано 82 тестові випадки, які охоплюють публічні сторінки, каталоги, глобальний пошук, сторінку товару, кошик, checkout, авторизацію, Trade In, адміністративну панель, зовнішні інтеграції, Docker-запуск і базові аспекти безпеки. Після первинного тестування було виявлено 11 проблем, більшість із яких стосувалися інтерфейсу, передавання товарних варіантів, логіки кошика та зручності адміністративної панелі. Після виправлень 80 тестових випадків виконуються повністю, а 2 залишаються залежними від зовнішніх сервісів.

Під час тестування було підтверджено працездатність ключових функцій системи. Глобальний пошук працює по всіх товарних групах, БУ-товари відображають фіксовані характеристики без зайвого вибору, аксесуари з кольоровими варіантами змінюють зображення відповідно до вибраного кольору, а кошик зберігає правильний варіант товару, його назву, ціну та зображення. Також перевірено, що Trade In-заявку не може подати

неавторизований користувач, а адміністратор має доступ до керування товарами, відгуками, заявками та замовленнями.

Окремо перевірено зовнішні інтеграції та розгортання: для LiqPay - формування платіжних параметрів на backend, для Google Identity - авторизацію через Google-акаунт, для Nova Post API - вибір відділення або поштомата, а також запуск системи через Docker Compose. Це підтвердило, що проєкт може працювати як цілісний вебзастосунок із frontend, backend, reverse proxy та зовнішніми сервісами.

Таким чином, тестування підтвердило відповідність Apple Shop основним функціональним і нефункціональним вимогам. Система є завершеним демонстраційним прототипом інтернет-магазину Apple-техніки та може бути використана як основа для подальшого розвитку з підключенням промислової бази даних, HTTPS, журналювання, моніторингу й резервного копіювання.

ВИСНОВКИ

Дана розробка представлена у форматі робочого прототипу інтернет-магазину Apple-техніки, що об’єднує клієнтську частину на React, серверну частину на Node.js, REST API, Docker-розгортання, адміністративну панель, інтеграцію з LiqPay, Google Identity та Nova Post API. Система орієнтована на продаж нової техніки, БУ-пристроїв та аксесуарів, а також підтримує Trade In-заявки, локальне керування зображеннями, кошик, оформлення замовлень і керування статусами оплати.

Архітектура Apple Shop побудована за клієнт-серверним підходом. Frontend відповідає за відображення сторінок, товарних карток, пошуку, кошика, checkout, авторизації та адміністративної панелі. Backend забезпечує обробку запитів, бізнес-логіку, роботу з товарами, замовленнями, відгуками, Trade In-заявками, оплатою та доставкою.

У процесі роботи було реалізовано основні сценарії електронної комерції. Користувач може переглядати головну сторінку, каталог нової техніки, розділ БУ-товарів, аксесуари, виконувати глобальний пошук по всіх товарних групах, відкривати сторінку товару, обирати варіант кольору, додавати товар до кошика та оформлювати замовлення. Для БУ-техніки пам’ять, колір і стан відображаються як фіксовані характеристики, що відповідає логіці продажу конкретної одиниці товару.

Окрему увагу приділено роботі з товарними варіантами та зображеннями. Якщо користувач обирає конкретний колір аксесуара або пристрою, система змінює зображення на сторінці товару та зберігає цей вибір у кошику й замовленні. Це дозволяє уникнути ситуації, коли користувач додає один варіант товару, а в кошику або замовленні бачить інший.

Адміністративна панель реалізована як окремий інструмент керування магазином. Адміністратор може додавати й редагувати товари, завантажувати локальні зображення, переглядати замовлення, підтверджувати оплату, видаляти відгуки та працювати з Trade In-заявками. Панель поділена на окремі розділи,

тому є зручнішою для щоденної роботи.

Інтеграції із зовнішніми сервісами наближають прототип до реального комерційного веб-застосунку. Google Identity використовується для швидкої авторизації користувача через Google-акаунт. LiqPay застосовується для підготовки онлайн-оплати, а Nova Post API використовується для вибору відділення або поштомата під час оформлення замовлення.

Фаза тестування охопила основні функціональні сценарії системи. Було перевірено роботу каталогів, глобального пошуку, сторінки товару, вибору кольору, кошика, checkout, Trade In, авторизації, адміністративної панелі, підтвердження оплати та Docker-запуску. Під час тестування було виявлено й виправлено недоліки, пов’язані з відображенням зображень, передаванням кольору в кошик, форматом цін, горизонтальною прокруткою та структурою адміністративної панелі.

Обмеження поточної версії системи є усвідомленими. Для промислового впровадження доцільно підключити повноцінну базу даних, налаштувати HTTPS, резервне копіювання, журналювання адміністративних дій, моніторинг, CI/CD та стабільне production-середовище. У майбутньому також можна додати промокоди, аналітику продажів, розширені фільтри, рекомендації товарів і додаткові способи доставки.

Попри ці обмеження, розроблений веб-застосунок виконує основні функції інтернет-магазину Apple-техніки та демонструє практичну реалізацію повного циклу електронної комерції. Apple Shop може бути використаний як основа для подальшого розвитку реального магазину, оскільки має зрозумілу архітектуру, розділені frontend і backend, підтримку зовнішніх сервісів, адміністративне керування та можливість розгортання через Docker.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Закон України «Про електронну комерцію» - Верховна Рада України (нормативно-правовий акт) – [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/675-19 - Дата звернення: 03.04.2026.

2. Закон України «Про захист прав споживачів» - Верховна Рада України (нормативно-правовий акт) – [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/1023-12 - Дата звернення: 09.04.2026.

3. Закон України «Про захист персональних даних» - Верховна Рада України (нормативно-правовий акт) – [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2297-17 - Дата звернення: 16.04.2026.

4. DataReportal. Digital 2025: Global Overview Report - DataReportal (аналітичний звіт про цифрову економіку) – [Електронний ресурс]. – Режим доступу: https://datareportal.com/reports/digital-2025-global-overview-report - Дата звернення: 22.04.2026.

5. UNCTAD. Digital Economy Report 2024 - UNCTAD (міжнародний звіт про цифрову економіку) – [Електронний ресурс]. – Режим доступу: https://unctad.org/publication/digital-economy-report-2024 - Дата звернення: 05.05.2026.

6. Baymard Institute. Product Listing UX: Information - Baymard Institute (UX-дослідження товарних списків) – [Електронний ресурс]. – Режим доступу: https://baymard.com/blog/product-listing-information - Дата звернення: 11.05.2026.

7. Baymard Institute. E-Commerce Checkout UX Research - Baymard Institute (UX-дослідження оформлення замовлення) – [Електронний ресурс]. – Режим доступу: https://baymard.com/blog/checkout-2024-launch - Дата звернення: 18.05.2026.

8. Baymard Institute. E-Commerce UX Research - Baymard Institute (дослідницький ресурс з UX електронної комерції) – [Електронний ресурс]. – Режим доступу: https://baymard.com/ - Дата звернення: 27.05.2026.

9. ISO/IEC/IEEE 12207:2017. Systems and software engineering - Software life cycle processes - ISO (міжнародний стандарт життєвого циклу ПЗ) – [Електронний ресурс]. – Режим доступу: https://www.iso.org/standard/63712.html - Дата звернення: 07.04.2026.

10. ISO/IEC/IEEE 29148:2018. Systems and software engineering - Life cycle processes - Requirements engineering - ISO (міжнародний стандарт інженерії вимог) – [Електронний ресурс]. – Режим доступу: https://www.iso.org/standard/72089.html - Дата звернення: 14.04.2026.

11. ISO/IEC 25010:2023. Systems and software Quality Requirements and Evaluation - Product quality model - ISO (міжнародний стандарт якості ПЗ) – [Електронний ресурс]. – Режим доступу: https://www.iso.org/standard/78176.html - Дата звернення: 25.04.2026.

12. Fielding R. Architectural Styles and the Design of Network-based Software Architectures - University of California, Irvine (дисертаційна робота про REST) – [Електронний ресурс]. – Режим доступу: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm - Дата звернення: 02.05.2026.

13. Mesbah A., van Deursen A. Migrating Multi-page Web Applications to Single-page AJAX Interfaces - arXiv (наукова публікація про SPA) – [Електронний ресурс]. – Режим доступу: https://arxiv.org/abs/cs/0610094 - Дата звернення: 09.05.2026.

14. Fowler M., Lewis J. Microservices - Martin Fowler (технічна стаття про мікросервіси) – [Електронний ресурс]. – Режим доступу: https://martinfowler.com/articles/microservices.html - Дата звернення: 21.05.2026.

15. Використання React та Redux під час розробки вебзастосунків - Інформаційні технології і суспільство (наукова стаття) – [Електронний ресурс]. – Режим доступу: https://infotech-soccult.knukim.edu.ua/article/view/335535 - Дата звернення: 29.04.2026.

16. Реалізація вебзастосунку для книжкового магазину-маркетплейсу - Науковий вісник НЛТУ України (наукова стаття) – [Електронний ресурс]. –

Режим доступу: https://nv.nltu.edu.ua/index.php/journal/article/view/2760 - Дата звернення: 15.05.2026.

17. React Documentation - React (офіційна документація frontend-бібліотеки) – [Електронний ресурс]. – Режим доступу: https://react.dev/ - Дата звернення: 01.04.2026.

18. Vite Documentation - Vite (офіційна документація інструмента збірки) – [Електронний ресурс]. – Режим доступу: https://vite.dev/guide/ - Дата звернення: 08.04.2026.

19. Node.js About - Node.js (офіційний ресурс серверного середовища виконання) – [Електронний ресурс]. – Режим доступу: https://nodejs.org/en/about/ - Дата звернення: 19.04.2026.

20. Express.js Documentation - Express.js (офіційна документація backend-фреймворку) – [Електронний ресурс]. – Режим доступу: https://expressjs.com/ - Дата звернення: 28.04.2026.

21. Docker Compose Documentation - Docker (офіційна документація контейнерного запуску) – [Електронний ресурс]. – Режим доступу: https://docs.docker.com/compose/ - Дата звернення: 04.05.2026.

22. NGINX Reverse Proxy Documentation - NGINX (офіційна документація reverse proxy) – [Електронний ресурс]. – Режим доступу: https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/ - Дата звернення: 12.05.2026.

23. OWASP Top 10:2021 - OWASP (рекомендації з безпеки веб-застосунків) – [Електронний ресурс]. – Режим доступу: https://owasp.org/Top10/2021/ - Дата звернення: 20.05.2026.

24. Google Identity Services: Sign in with Google - Google Developers (офіційна документація Google-авторизації) – [Електронний ресурс]. – Режим доступу: https://developers.google.com/identity/gsi/web/guides/display-button - Дата звернення: 26.04.2026.

25. LiqPay Checkout API - LiqPay (документація платіжного API) – [Електронний ресурс]. – Режим доступу:

https://www.liqpay.ua/en/doc/api/internet_acquiring/checkout - Дата звернення: 06.05.2026.

26. Nova Post API Documentation - Nova Post (документація логістичного API) – [Електронний ресурс]. – Режим доступу: https://api-portal.novapost.com/en/about-api/general/ - Дата звернення: 24.05.2026.

27. Wiegers K., Beatty J. Software Requirements, 3rd Edition - Microsoft Press (навчальне видання з інженерії вимог) – [Електронний ресурс]. – Режим доступу: https://www.microsoftpressstore.com/store/software-requirements-9780735679665 - Дата звернення: 10.04.2026.

28. IIBA. BABOK Guide: Elicitation and Collaboration - IIBA (професійний стандарт бізнес-аналізу) – [Електронний ресурс]. – Режим доступу: https://www.iiba.org/knowledgehub/business-analysis-body-of-knowledge-babok-guide/4-elicitation-and-collaboration/ - Дата звернення: 17.04.2026.

29. Google web.dev. Web Vitals - Google web.dev (технічна документація з продуктивності вебсайтів) – [Електронний ресурс]. – Режим доступу: https://web.dev/articles/vitals - Дата звернення: 23.04.2026.

30. W3C. Web Content Accessibility Guidelines 2.2 - W3C (стандарт вебдоступності) – [Електронний ресурс]. – Режим доступу: https://www.w3.org/WAI/standards-guidelines/wcag/ - Дата звернення: 30.04.2026.

31. MDN Web Docs. Client-side form validation - MDN Web Docs (довідковий ресурс з frontend-розробки) – [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Forms/Form_validation - Дата звернення: 07.05.2026.

32. Nielsen Norman Group. 10 Usability Heuristics for User Interface Design - Nielsen Norman Group (UX-рекомендації) – [Електронний ресурс]. – Режим доступу: https://www.nngroup.com/articles/ten-usability-heuristics/ - Дата звернення: 13.05.2026.

33. Bass L., Clements P., Kazman R. Software Architecture in Practice, 4th Edition - Software Engineering Institute (навчальне видання з архітектури ПЗ) –

[Електронний ресурс]. – Режим доступу: https://www.sei.cmu.edu/library/software-architecture-in-practice-fourth-edition/ - Дата звернення: 28.05.2026.

34. Object Management Group. Unified Modeling Language Specification 2.5.1 - OMG (специфікація UML) – [Електронний ресурс]. – Режим доступу: https://www.omg.org/spec/UML/2.5.1 - Дата звернення: 02.04.2026.

35. C4 Model. Lightweight standard for visualizing software architecture - C4 Model (методика моделювання архітектури) – [Електронний ресурс]. – Режим доступу: https://c4model.com/ - Дата звернення: 15.04.2026.

36. GitHub Actions Documentation - GitHub Docs (офіційна документація CI/CD) – [Електронний ресурс]. – Режим доступу: https://docs.github.com/en/actions - Дата звернення: 24.04.2026.

37. The Twelve-Factor App: Config - Twelve-Factor App (методичні рекомендації з конфігурації застосунків) – [Електронний ресурс]. – Режим доступу: https://www.12factor.net/config - Дата звернення: 01.05.2026.

38. PostgreSQL Documentation - PostgreSQL (офіційна документація СУБД) – [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/current/ - Дата звернення: 10.05.2026.

39. Vitest Documentation - Vitest (офіційна документація інструмента модульного тестування) – [Електронний ресурс]. – Режим доступу: https://vitest.dev/ - Дата звернення: 19.05.2026.

40. Testing Library Documentation - Testing Library (офіційна документація тестування UI-компонентів) – [Електронний ресурс]. – Режим доступу: https://testing-library.com/docs/ - Дата звернення: 29.05.2026.

41. Playwright Documentation - Playwright (офіційна документація E2E-тестування) – [Електронний ресурс]. – Режим доступу: https://playwright.dev/docs/intro - Дата звернення: 05.04.2026.

42. ESLint Documentation - ESLint (офіційна документація статичного аналізу коду) – [Електронний ресурс]. – Режим доступу: https://eslint.org/docs/latest/ - Дата звернення: 13.04.2026.

43. Prettier Documentation - Prettier (офіційна документація форматування коду) – [Електронний ресурс]. – Режим доступу: https://prettier.io/docs/ - Дата звернення: 21.04.2026.

44. MDN Web Docs. Web Storage API - MDN Web Docs (довідковий ресурс з браузерного збереження даних) – [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API - Дата звернення: 02.05.2026.

45. Apache JMeter Documentation - Apache JMeter (документація інструмента навантажувального тестування) – [Електронний ресурс]. – Режим доступу: https://jmeter.apache.org/ - Дата звернення: 09.05.2026.

46. Selenium WebDriver Documentation - Selenium (документація інструмента браузерного тестування) – [Електронний ресурс]. – Режим доступу: https://www.selenium.dev/documentation/webdriver/ - Дата звернення: 16.05.2026.

47. Chrome Lighthouse Documentation - Chrome Developers (документація інструмента аудиту вебсторінок) – [Електронний ресурс]. – Режим доступу: https://developer.chrome.com/docs/lighthouse - Дата звернення: 23.05.2026.

48. Grafana k6 Documentation - Grafana k6 (документація інструмента навантажувального тестування) – [Електронний ресурс]. – Режим доступу: https://grafana.com/docs/k6/latest/ - Дата звернення: 30.05.2026.

49. ISTQB Certified Tester Foundation Level CTFL v4.0 - ISTQB (стандарт і навчальна програма з тестування ПЗ) – [Електронний ресурс]. – Режим доступу: https://www.istqb.org/certifications/certified-tester-foundation-level-ctfl-v4-0/ - Дата звернення: 06.04.2026.

50. Postman Docs: Run and automate API tests - Postman Learning Center (документація інструмента API-тестування) – [Електронний ресурс]. – Режим доступу: https://learning.postman.com/docs/tests-and-scripts/run-tests/run-tests/ - Дата звернення: 26.05.2026.

ДОДАТКИ

Додаток А

Лістинг А.1 – Функція отримання ціни товару залежно від обсягу пам’яті

```
export function getVariantPrice(product, selectedStorage) {
   const prices = product?.price_by_storage || PRICE_MAP[product?.title]
   if (!prices || !selectedStorage) {
      return Number(product?.price || 0)
   }
   return Number(prices[String(selectedStorage)] || product?.price || 0)
}
```

Лістинг А.2 – Визначення зображення товару залежно від вибраного кольору

```
function getVariantImage(product, selectedColor) {
   const variant = product.variants?.find((item) =>
      item.color === selectedColor
   )
   return variant?.image_url || product.image_url
}
```

Лістинг А.3 – Додавання товару до кошика з урахуванням варіанта

```
const cartItem = {
   product_id: product.id,
   title: product.title,
   price: selectedPrice,
   quantity: 1,
   color: selectedColor,
   storage: selectedStorage,
   image_url: selectedImage,
}

addToCart(cartItem)
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: “Веб-застосунок по продажу техніки Apple”

Обсяг пояснювальної записки: 74 аркуші

Дата закінчення роботи червня 2026р.

Підпис _____