МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії
програмного забезпечення

_____ Євген ДАВИДЕНКО

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

Вебзастосунок магазин одягу

Спеціальність 121 Інженерія програмного забезпечення
Освітня програма «Інженерія програмного забезпечення»

Здобувач _____ **Олександр ПІВОШЕНКО**

«__» _____ 20__ р.

Керівник роботи
доцентка,
PhD _____ **Катерина АНТІПОВА**

«__» _____ 20__ р.

Миколаїв – 2026

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп'ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступінь	Бакалавр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«___» _____ 2026 р.

ЗАВДАННЯ

на кваліфікаційну бакалаврську роботу здобувача

Півошенко Олександра

1. Тема кваліфікаційної роботи: Вебзастосунок магазин одягу затверджена наказом ректора ЧНУ ім. Петра Могили № 349 від «26» грудня 2025 р.

2. Строк представлення кваліфікаційної роботи «_» червня 2026 р.

3. Очікуваний результат роботи та початкові дані якщо такі потрібні: очікуваним результатом є вебзастосунок інтернет магазин одягу.

4. Перелік питань, що підлягають розробці:

- провести аналіз предметної області;
- зробити порівняльний аналіз аналогів;
- визначити вимоги та функціонал вебзастосунку;

– виконати моделювання та проєктування програмного застосунку;
– розробити клієнтську та серверну частини системи;
– провести тестування застосунку.

5. Перелік графічних матеріалів: слайди презентації.
6. Консультанти:

Консультант	**Кафедра (організація)**	**Частина роботи**

Дата видачі завдання «25» лютого 2026 р.

2026 р. Півошенко Олександр

Кафедра інженерії програмного забезпечення
Вебзастосунок магазин одягу

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: **Вебзастосунок магазин одягу**

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КБР	26.12.2025	03.02.2026	Виконано
2.	Огляд літератури за темою роботи	04.02.2026	10.02.2026	Виконано
3.	Складання календарного плану КБР	11.02.2026	12.02.2026	Виконано
4.	Аналіз предметної області	13.02.2026	16.02.2026	Виконано
5.	Розробка проєктних рішень	17.02.2026	23.02.2026	Виконано
6.	Моделювання та конструювання ПЗ	24.02.2026	18.03.2026	Виконано
7.	Кодування, тестування та апробація розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	19.03.2026	09.04.2026	Виконано
8.	Відгук керівника КБР	10.04.2026	14.04.2026	Виконано
9.	Оформлення КБР та презентації	15.04.2026	17.04.2026	Виконано
10.	Попередній захист			
11.	Рецензування			
12.	Завершення оформлення КБР та презентації			
13.	Захист кваліфікаційної роботи			

Здобувач _____ **Олександр ПІВОШЕНКО**

«__» _____ 20_26_ р.

Керівник роботи

доцентка, _____ **Катерина АНТІПОВА**

PhD

«__» _____ 20_26_ р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

«Вебзастосунок магазин одягу»

Здобувач 408 гр.: Півошенко Олександр

Керівник: PhD, доцент Антіпова Катерина

Серед різноманіття інтернет-магазинів вагоме місце займають саме платформи з продажу одягу. Потреба в одязі виникає з моменту народження людини та супроводжує її протягом усього життя. З розвитком суспільства одяг перестав виконувати виключно захисну функцію від зовнішніх факторів, таких як холод або спека, і набув значно ширшого значення. Зокрема, він використовується як спеціалізований елемент у професійній діяльності в різних галузях виробництва та сфері обслуговування.

У сучасному світі для багатьох людей, незалежно від віку, одяг є важливим засобом самовираження. Він дозволяє підкреслити індивідуальність та сформувати власний стиль.

З розвитком інформаційних технологій інтернет-магазини одягу досягли високого рівня зручності у виборі та придбанні товарів. Серед їхніх переваг можна виділити зручну навігацію, можливість пошуку за категоріями, систему сповіщень та персоналізовані рекомендації. Водночас існує потреба у подальшому вдосконаленні таких сервісів.

Метою створення інтернет-магазину є підвищення якості обслуговування користувачів, розширення асортименту товарів та забезпечення комфортної взаємодії з системою. Особлива увага приділяється створенню простого, зрозумілого та естетичного інтерфейсу без надмірного навантаження рекламою та акційними елементами.

Науково-практичне значення кваліфікаційної роботи полягає у дослідженні архітектури fullstack-застосунків. Застосуванні сучасних технологій такі як React, Node.js, Express, MSSQL, Sequelize. Реалізації бізнес-логіки інтернет-магазину та побудові масштабованої та підтримуваної

системи. Результат роботи може бути використаний як основа для створення реального комерційного продукту.

Об’єктом роботи є процеси електронної комерції інтернет-магазинів.

Предметом роботи є інструменти розробки fullstack вебзастосунків.

Метою кваліфікаційної роботи є розробка вебзастосунку інтернет-магазину одягу для автоматизації основних функцій електронної комерції.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків та списку використаних джерел.

У вступі обґрунтовано актуальність теми, визначено об’єкт, предмет і мету дослідження.

У першому розділі здійснено аналіз предметної області, розглянуто особливості функціонування інтернет-магазинів та проаналізовано існуючі аналоги.

В другому розділі сформульована специфікація вимог до системи та обґрунтувано вибір технологій розробки.

У третьому розділі представлене моделювання системи за допомогою UML-діаграм та описана архітектура вебзастосунку.

У четвертому розділі наведено опис процесів розробки та тестування вебзастосунку.

У висновках підсумовано результати виконаної роботи та окреслено перспективи подальшого розвитку системи.

Кваліфікаційна робота викладена на 80 сторінках друкованого тексту, містить вступ, чотири розділи, загальні висновки, список використаних джерел із 24 найменувань та __ додатків. Робота включає 8 таблиць і 28 рисунків.

Ключові слова: база даних, вебзастосунок, веброзробка, електронна комерція, інтернет-магазин, користувацький інтерфейс, продаж одягу.

Кафедра інженерії програмного забезпечення

Вебзастосунок магазин одягу

ABSTRACT

for the Bachelor's Qualification Thesis

«Web Application Clothing Store»

Student of group 408: Pivoshenko Oleksandr

Supervisor: PhD, Associate Professor Antipova Kateryna

Among the wide variety of online stores, platforms specializing in clothing sales occupy a significant place. The need for clothing arises from birth and accompanies a person throughout their entire life. With the development of society, clothing has ceased to serve solely as protection against environmental factors such as cold or heat and has gained a much broader significance. In particular, it is used as a specialized element in professional activities across various industries and service sectors.

In the modern world, for many people regardless of age, clothing serves as an important means of self-expression. It allows individuals to emphasize their personality and create their own unique style.

With the advancement of information technologies, online clothing stores have reached a high level of convenience in terms of product selection and purchasing. Among their advantages are user-friendly navigation, category-based search, notification systems, and personalized recommendations. At the same time, there remains a need for further improvement of such services.

The purpose of developing this online store is to improve the quality of user service, expand the product range, and ensure comfortable interaction with the system. Special attention is paid to creating a simple, intuitive, and aesthetically pleasing interface without excessive advertising and promotional overload.

The scientific and practical significance of the work lies in the study of fullstack application architecture, the use of modern technologies such as React, Node.js, Express, MSSQL, and Sequelize, as well as the implementation of business logic for an online store and the development of a scalable and maintainable system. The results of this work can be used as a foundation for creating a real commercial product.

2026 р. Півошенко Олександр

The object of the study is the processes of e-commerce in online stores.

The subject of the study is the tools and technologies used for developing fullstack web applications.

The aim of the qualification thesis is to develop a web application for an online clothing store to automate the core functions of e-commerce.

The qualification thesis consists of an introduction, four chapters, conclusions, and a list of references.

The introduction substantiates the relevance of the topic and defines the object, subject, and purpose of the research.

The first chapter provides an analysis of the subject area, examines the features of online store operation, and reviews existing analogues.

The second chapter formulates the system requirements specification and justifies the choice of development technologies.

The third chapter presents the system modelling using UML diagrams and describes the web application architecture.

The fourth chapter presents a description of the development and testing processes of the web application.

The conclusions summarize the results of the work and outline possible directions for further system development.

The thesis is presented on 80 pages of typed text and includes an introduction, four chapters, general conclusions, a list of references containing 24 sources, and __ appendices. The work contains 8 tables and 28 figures.

Keywords: clothing sales, database, e-commerce, online store, user interface, web application, web development

ЗМІСТ

ВСТУП

У сучасному світі електронна комерція є одним із найдинамічніших напрямів розвитку інформаційних технологій. Онлайн-магазини дозволяють автоматизувати процеси продажу, обліку товарів, взаємодії з клієнтами та аналітики. За останні роки купівля через інтернет перетворилася зі зручної альтернативи на звичну для більшості споживачів модель поведінки, а веб-застосунок нерідко стає основним, а подеколи й єдиним каналом збуту для торгової марки. Це зміщує акцент із простої присутності в мережі на якість самого програмного продукту – його швидкодію, надійність та зручність використання.

Особливо актуальною є розробка веб-застосунків для інтернет-магазинів одягу, оскільки зростає кількість онлайн-покупок. Специфіка одягу як товарної категорії полягає в тому, що покупець не може примірити річ перед оформленням замовлення, а отже, вирішального значення набувають детальний опис, наявність розмірів і кольорів та зрозумілий механізм повернення. Користувачі очікують зручний інтерфейс, швидкий пошук товарів, персоналізовані рекомендації та безпечну обробку даних. Помітне зростання частки замовлень зі смартфонів додатково підвищує вимоги до адаптивності й швидкості відображення сторінок. У свою чергу, бізнес потребує ефективних інструментів для управління товарами, замовленнями та клієнтською базою.

Аналіз сучасного стану електронної комерції показує, що існує велика кількість готових рішень, проте багато з них є або надто складними для адаптації, або потребують значних фінансових витрат. Готові платформи пришвидшують запуск, однак обмежують гнучкість і прив'язують власника до сторонньої інфраструктури, тоді як індивідуальна розробка дає повний контроль над функціональністю ціною більших початкових зусиль. Крім того, не всі системи забезпечують достатню гнучкість та масштабованість. Сучасні тенденції розвитку вебтехнологій орієнтовані на використання fullstack-

підходу, мікросервісної архітектури, API-орієнтованих рішень та інтерактивних користувацьких інтерфейсів. Дедалі ширше застосовується архітектура односторінкового застосунку (SPA) із чітким розмежуванням клієнтської та серверної частин, які обмінюються даними через REST API, що забезпечує плавність роботи інтерфейсу без повного перезавантаження сторінок.

Науково-практичне значення роботи полягає у дослідженні архітектури fullstack-застосунків. Застосуванні сучасних технологій такі як React, Node.js, Express, MSSQL, Sequelize. Реалізації бізнес-логіки інтернет-магазину та побудові масштабованої та підтримуваної системи. Результат роботи може бути використаний як основа для створення реального комерційного продукту.

Метою роботи є розробка вебзастосунку інтернет-магазину одягу для автоматизації основних функцій електронної комерції.

Для досягнення поставленої мети необхідно виконати такі завдання:

1) виконати порівняльний аналіз існуючих інтернет-магазинів одягу;
2) виконати проєктування архітектури застосунку (Frontend, Backend, Database частини);
3) виконати моделювання системи;
4) розробити структуру бази даних;
5) реалізувати функціонал:
   - перегляду товарів;
   - фільтрації та пошуку;
   - роботи з кошиком;
   - оформлення замовлень;
   - реалізувати систему email-сповіщень (вітальний лист, підтвердження замовлення, відновлення пароля, статус повернення);
   - реалізувати інтеграцію з платіжним шлюзом Stripe для онлайн-оплати замовлень;
   - реалізувати механізм оформлення повернень товарів;

- реалізувати перемикання світлої та темної теми оформлення інтерфейсу;
- додавання відгуків та обраного.

6) реалізувати REST API для взаємодії клієнта та сервера;
7) реалізувати систему аутентифікації та авторизації (JWT);
8) реалізувати рольову модель (User / Admin);
9) провести тестування роботи системи;

Об’єктом роботи є процеси електронної комерції інтернет-магазинів одягу.

Предметом роботи є інструменти розробки fullstack вебзастосунків.

Обґрунтування вибору проєктних рішень полягає у використанні сучасного стеку технологій, який забезпечує високу продуктивність, масштабованість та зручність підтримки. Зокрема, застосування React дозволяє створити динамічний користувацький інтерфейс, Node.js та Express – ефективно реалізувати серверну логіку, а MSSQL у поєднанні з ORM Sequelize – забезпечити надійне зберігання та обробку даних.

Сфера застосування результатів роботи охоплює розробку та впровадження інтернет-магазинів одягу, а також може бути розширена для інших напрямів електронної комерції, де необхідна автоматизація процесів продажу та управління товарами.

1 АНАЛІЗ СФЕРИ ЕЛЕКТРОННОЇ КОМЕРЦІЇ

1.1 Характеристика предметної області

Електронна комерція (e-commerce) сьогодні є головним рушієм трансформації світового ритейлу. Це не просто купівля-продаж товарів, а складна екосистема, що охоплює фінансові транзакції, логістичні ланцюги та маркетингові комунікації. За останні п’ять років частка онлайн-продажів у загальному обсязі світової торгівлі зросла майже вдвічі, що підтверджує актуальність переходу бізнесу у цифрове середовище.

У науковій та діловій практиці системи електронної комерції прийнято класифікувати за типом взаємодії учасників [2]:

- B2B (Business-to-Business): оптова торгівля між компаніями;
- C2C (Consumer-to-Consumer): майданчики для торгівлі між приватними особами;
- B2C (Business-to-Consumer): пряма взаємодія бренду з кінцевим споживачем, що є предметом даного дослідження.

Специфіка ніші Fashion e-commerce (торгівля одягом) вимагає від розробника особливої уваги до архітектури системи через високу динамічність даних. На відміну від статичних товарів, асортимент одягу змінюється щосезону, що вимагає гнучких інструментів управління каталогом.

Основними технічними викликами предметної області є:

- управління залишками (Stock Management): необхідність синхронізації бази даних з реальним складом, щоб уникнути ситуації, коли покупець оплачує товар, якого немає в наявності;
- візуальна презентація: впровадження високоякісних медіа-галерей та відео-оглядів, що потребує оптимізації завантаження контенту (Lazy Loading) для збереження швидкодії;
- персоналізація: формування індивідуальних підбірок на основі попередніх переглядів користувача, що підвищує конверсію.

Специфіка предметної області магазинів одягу зумовлена наступними факторами:

- висока візуальна складова: на відміну від побутової техніки чи книг, покупка одягу значною мірою базується на естетичному сприйнятті, що вимагає від системи здатності обробляти та якісно відображати велику кількість медіаконтенту;
- складність атрибуції товарів: кожен товар у цій ніші має специфічну сітку характеристик –розміри (S, M, L тощо), кольори, матеріали, сезонність. Це створює навантаження на архітектуру бази даних;
- проблема вибору та повернень: через відсутність фізичного контакту з товаром користувачі потребують детальних описів, таблиць розмірів та систем відгуків, щоб мінімізувати ризик помилки.

Основними учасниками системи інтернет-магазину є:

- покупець (клієнт): основний користувач, чия поведінка характеризується етапами пошуку, порівняння, вибору та безпосередньої транзакції. Для цієї ролі критично важливими є швидкість завантаження сторінок, інтуїтивна навігація та безпека платежів;
- адміністратор (менеджер): виконує функції операційного управління. Він відповідає за актуальність асортименту, модерацію контенту, обробку замовлень та контроль складських залишків.

Важливим технічним аспектом функціонування сучасних систем є продуктивність. Згідно з аналітичними даними, затримка завантаження інтерфейсу навіть на одну секунду може призвести до зниження конверсії на 7-10% [14]. Тому розробка має базуватися на принципах асинхронності та оптимізації обміну даними між клієнтом і сервером.

Окрім суто функціональних можливостей, сучасний інтернет-магазин має відповідати принципам адаптивного дизайну [6]. З огляду на тенденцію "Mobile First", система повинна однаково коректно працювати на настільних ПК, планшетах та смартфонах, забезпечуючи цілісність користувацького досвіду незалежно від пристрою.

1.2 Аналіз існуючих програмних рішень

З метою визначення основних підходів до реалізації інтернет-магазинів одягу було проведено аналіз рішень провідних світових компаній. Особливу увагу приділено функціональним можливостям та інтерфейсним рішенням, які стали галузевим стандартом у сфері Fashion e-commerce.

Платформа Nike є еталоном реалізації концепції «Digital Experience». Її інтерфейс (Рисунок 1.1) розроблений таким чином, щоб створювати ефект повного занурення через використання високоякісного медіаконтенту, який динамічно адаптується під інтереси конкретного користувача. Технічно це реалізовано за допомогою складних алгоритмів рекомендацій та гнучкого фронтенду на базі фреймворків React та Next.js.

Особлива увага в системі приділяється інструментам персоналізації: сервіс кастомізації взуття інтегрований безпосередньо в процес вибору товару, що вимагає високої синхронізації між графічним візуалізатором та модулем управління замовленнями. Це перетворює платформу з простої вітрини на інтерактивний інструмент взаємодії, де швидкість відгуку інтерфейсу є критично важливою для збереження конверсії [15].

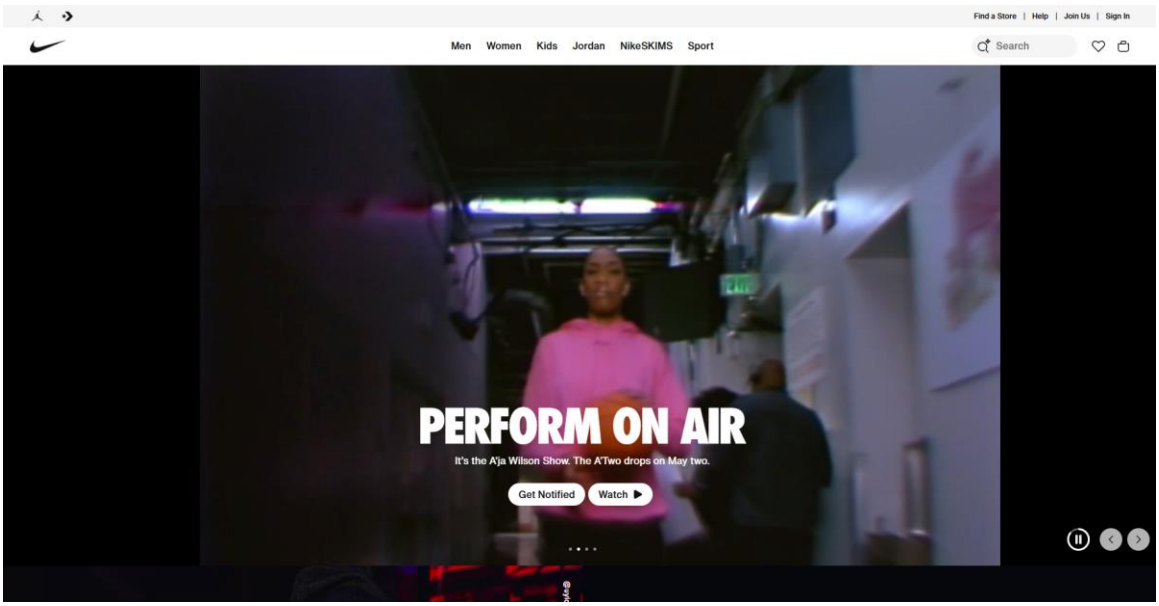

Рисунок 1.1 – Інтерфейс головної сторінки інтернет-магазину Nike

Система ASOS вирізняється своєю архітектурною здатністю обробляти колосальні обсяги даних. Оскільки асортимент платформи налічує понад 85 000 одиниць товарів від сотень різних брендів, ключовим елементом інтерфейсу є потужна система інтелектуальної фільтрації (Рисунок 1.2). Вона побудована за принципом фасетної навігації, що дозволяє користувачеві миттєво відсіювати зайве не лише за базовими параметрами (розмір, ціна), а й за специфічними атрибутами: типом крою, принтом чи матеріалом.

Важливою інновацією ASOS є впровадження AR-технологій та систем машинного навчання для підбору розміру («See My Fit»), які допомагають візуалізувати посадку одягу на різних типах фігур. З точки зору розробки, це вимагає складної структури бази даних та оптимізованих API-запитів, щоб забезпечити безперебійну роботу фільтрів при величезній кількості одночасних сесій [5].

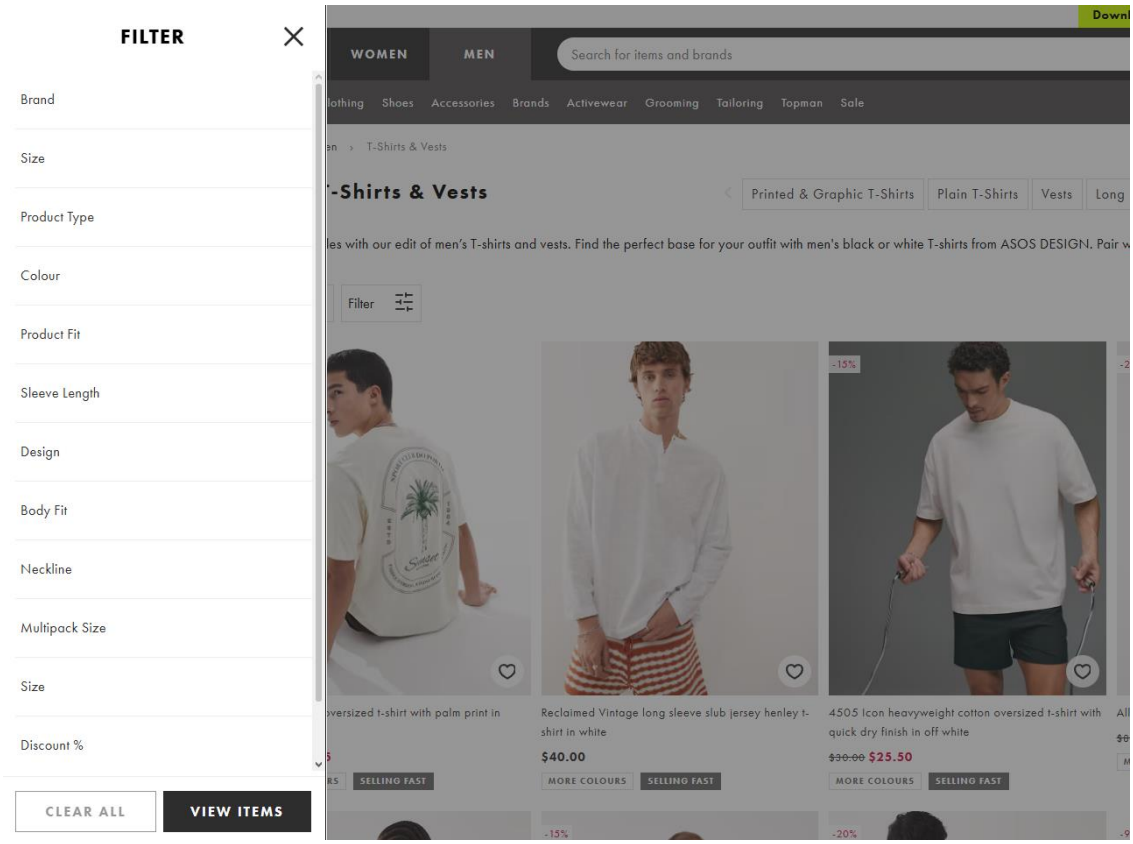

Рисунок 1.2 – Реалізація системи фільтрів на сайті ASOS

Платформа Adidas фокусується на поєднанні зручної навігації та глибокої інтеграції з екосистемою бренд-лояльності. Як зображено на рисунку

1.3, система категорій реалізована максимально наочно: користувач одразу бачить поділ не лише за типом одягу, а й за колекціями та видами спорту. Це дозволяє максимально скоротити «шлях клієнта» (Customer Journey) до цільової дії.

Особливістю архітектури Adidas є використання гібридного підходу, де фронтенд-частина забезпечує швидкий рендеринг складних елементів вибору (розмірна сітка, варіативність кольорів), а бекенд-система в реальному часі обробляє запити до програм лояльності. Такий підхід гарантує безшовний досвід переходу від перегляду каталогу до фіналізації покупки, що є вкрай важливим для систем, орієнтованих на велику аудиторію [4].

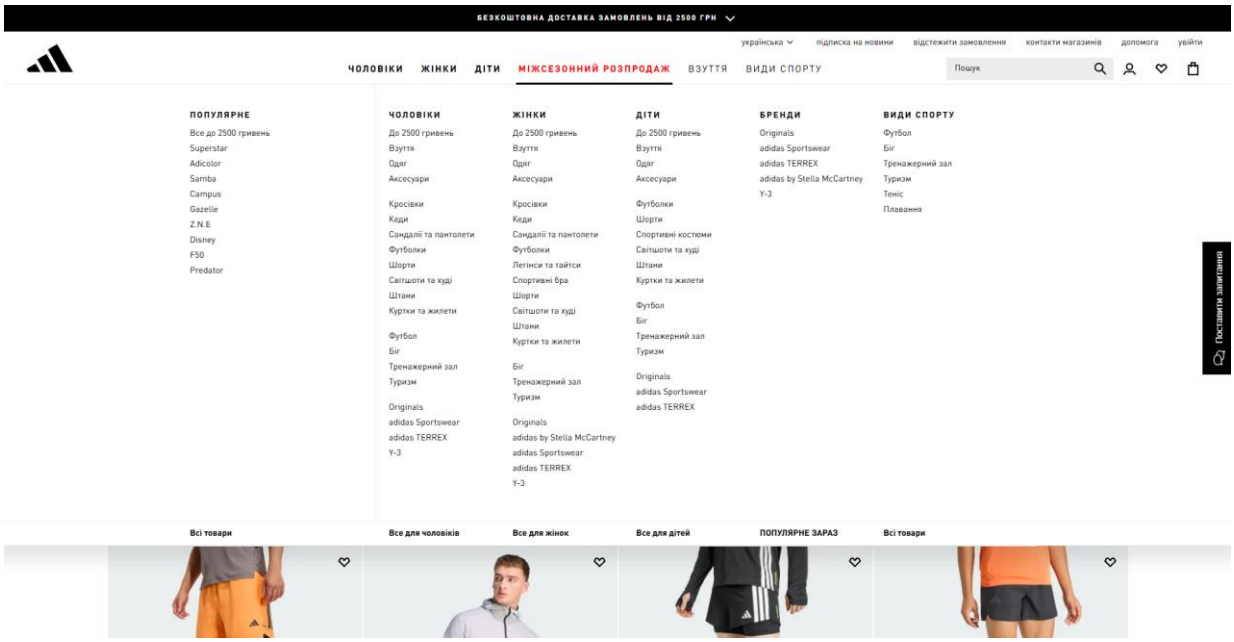

Рисунок 1.3 – Реалізація вибору категорій товару на сайті Adidas

Для систематизації отриманих даних та вибору оптимального шляху розробки було проведено порівняльний аналіз технічних характеристик зазначених систем. Критеріями порівняння обрано архітектурні підходи, використані технології та ключові переваги, що безпосередньо впливають на користувацький досвід. Результати аналізу наведено в таблиці 1.1.

Таблиця 1.1 – Приклади аналогічних систем

Назва	**Nike**	**Adidas**	**ASOS**
Виробник	Nike, Inc.	Adidas AG	ASOS Plc
Архітектура	Сучасна мікросервісна архітектура. Використовує хмарні сервіси (AWS), що дозволяє обробляти величезні навантаження під час запуску нових колекцій. Включає окремі сервіси для каталогу, кошика, користувачів, замовлень.	Гібридна архітектура. Використовує комерційні платформи електронної комерції, інтегровані з власними сервісами та хмарними рішеннями.	Мікросервісна архітектура на базі хмарних технологій (AWS). Побудована навколо швидкого масштабування та обробки величезного асортименту.
Мова реалізації	Java, Node.js , Javascript (React, Next.js) , Swift, Kotlin.	Java, PHP, Javascript (React) , Swift.	.NET Core, Node.js, Javascript (React)
Основні функції	Індивідуалізація. Функція Nike By You для кастомізації взуття та одягу. SNKRS App система розподілу лімітованих продуктів. Push-сповіщення про початок продажів, знижки, статус замовлення. Синхронізація з додатками для тренувань Nike Training Club та Nike Run Club. Збереження товару на обмежений час. Розширена система фільтрів: по полу, виду спорту, розміру, кольору, технології. Nike Membership: програма лояльності з безкоштовною	Миттєві покупки (Speedflow): оптимізований процес оформлення замовлення в кілька кліків. Попереднє замовлення: можливість замовити продукт до його надходження на склад. Віртуальний примерювач: AR-функція для примеряння взуття через камеру смартфона. Застосунок Creators Club: програма лояльності з різними рівнями (отримання бонусів, доступ до обмежених колекцій, персональні знижки). Інтеграція з додатком Runtastic: для відстеження активності та	Величезний асортимент: понад 85 000 продуктів від власного бренду та сотень інших брендів. ASOS Marketplace: платформа для незалежних продавців. Технологія "See My Fit": AR-функція, яка показує, як сидить одяг на моделях різних розмірів і пропорцій. Потужна система фільтрів: за розміром, кольором, брендом, принтом, довжиною рукава тощо. Збереження розмірів: система запам'ятовує ваші розміри для різних брендів. Передплата ASOS Premium: безкоштовна доставка наступного дня та повернення протягом року. Можливість створювати та

	доставкою, ексклюзивним контентом та доступом до продуктів.	отримання пропозицій щодо екіпірування. Збереження списку бажань (Wishlist). Зчитування QR-кодів з фізичних товарів у магазинах для отримання додаткової інформації.	переглядати списки бажань.

Кінець таблиці 1.1

Переваги	Найвища продуктивність та стабільність навіть під час пікових навантажень. Потужний мобільний застосунок з унікальними функціями (SNKRS). Сильний бренд та ексклюзивні продукти, що створюють попит. Відмінна персоніфікація та рекомендаційна система. Зручний і інтуїтивно зрозумілий інтерфейс	Швидкий і оптимізований процес покупки. Інноваційні функції (AR-примерка, інтеграція з магазинами). Ефективна програма лояльності, що мотивує до повторних покупок. Широкий асортимент, що включає спорт, повсякденний одяг і взуття.	Найширший асортимент серед конкурентів. Дуже гнучка політика повернення (до 28 днів). Сильний акцент на технологіях для примерки та пошуку. Цілодобова підтримка клієнтів. Доступні ціни та часті розпродажі.
Недоліки	Висока цінова категорія. Відсутність в наявності популярних розмірів через високий попит. Складність архітектури робить систему дуже дорогою в розробці та підтримці	Інтерфейс іноді може бути перевантаженим. Система може працювати повільніше за Nike під час високого навантаження. Менш агресивна маркетингова стратегія для дропів, ніж у Nike.	Якість власного бренду може бути нестабільною. Через величезну кількість товарів іноді складно знайти щось конкретне. Можливі проблеми з доставкою та наявністю через логістичну складність.
Джерело інформації	[15]	[4]	[5]

Проведений аналіз показав, що сучасні інтернет-магазини одягу використовують переважно мікросервісну або гібридну архітектуру, що забезпечує масштабованість та високу продуктивність систем. Значна увага

приділяється персоналізації користувацького досвіду, впровадженню рекомендаційних систем, а також інтеграції з мобільними застосунками.

Разом з тим, існуючі рішення мають певні недоліки, зокрема високу складність реалізації, значні витрати на підтримку та обмежену гнучкість для адаптації під конкретні бізнес-потреби.

Отже, доцільним є розроблення власного вебзастосунку інтернет-магазину, який поєднуватиме основні переваги існуючих систем та водночас буде більш гнучким, масштабованим і адаптованим до конкретних умов використання.

1.3 Аналіз бізнес-процесів та вимоги до розроблюваної системи

Функціонування інтернет-магазину одягу базується на сукупності бізнес-процесів, які забезпечують повний цикл взаємодії з клієнтом – від першого візиту до отримання замовлення. Автоматизація цих процесів є ключовим завданням при розробці вебзастосунку.

Розглянемо життєвий цикл основних бізнес-процесів детальніше:

- управління каталогом товарів: процес включає створення карток товарів, групування їх за категоріями та брендами. Важливим елементом є підтримка динамічних характеристик, що дозволяє адміністратору оперативно вносити зміни в асортимент;
- пошук і вибір товару: реалізується через систему багаторівневої фільтрації та сортування. Процес має бути спроєктований так, щоб мінімізувати кількість кліків, необхідних для знаходження потрібної речі;
- формування кошика та управління замовленням: кошик є проміжною ланкою, де відбувається акумуляція обраних позицій, розрахунок загальної вартості та перевірка наявності товару на складі в режимі реального часу;
- процес оплати та безпека: інтеграція з платіжними шлюзами дозволяє автоматизувати фінансову складову. Бізнес-процес вимагає високого рівня захисту персональних даних та використання шифрованих з'єднань.

Процес взаємодії користувача із системою починається з перегляду каталогу товарів. Користувач може застосовувати фільтри, виконувати пошук та переглядати детальну інформацію про товар. Після вибору товар додається до кошика, де формується замовлення.

Наступним етапом є оформлення замовлення, під час якого користувач вводить необхідні дані. Після підтвердження замовлення інформація передається до серверної частини для подальшої обробки.

Адміністратор системи виконує функції управління товарами, категоріями, користувачами та замовленнями, що забезпечує актуальність і коректність роботи системи.

Таким чином, ефективність роботи інтернет-магазину значною мірою залежить від правильної організації та автоматизації зазначених бізнес-процесів.

На основі проведеного аналізу предметної області та існуючих рішень можна сформулювати основні вимоги до розроблюваної системи.

Функціональні вимоги визначають перелік можливостей, які повинна забезпечувати система:

- перегляд каталогу товарів;
- пошук та фільтрація;
- додавання товарів до кошика;
- оформлення замовлення;
- реєстрація та авторизація користувачів;
- система відгуків;
- управління товарами та замовленнями адміністратором.

Нефункціональні вимоги включають [9]:

- продуктивність –швидка обробка запитів користувачів;
- масштабованість –можливість розширення системи;
- безпека –захист даних користувачів;
- зручність використання –інтуїтивно зрозумілий інтерфейс;
- надійність –стабільна робота системи.

Врахування зазначених вимог є необхідною умовою для створення якісного вебзастосунку.

Для створення конкурентоспроможного продукту необхідно враховувати глобальні тренди розробки, які стають стандартом де-факто:

- статистика свідчить, що понад 70% транзакцій у сфері моди здійснюються з мобільних пристроїв. Це вимагає від веброзробки використання Progressive Web App (PWA) підходів або, як мінімум, глибокої оптимізації інтерфейсу під сенсорне керування;
- оскільки проєкт передбачає інтеграцію зі Stripe, важливо розуміти принцип "Tokenization". Система не повинна зберігати CVV-коди чи повні номери карток у власній базі даних. Використання зовнішніх шлюзів дозволяє перекласти відповідальність за безпеку транзакцій на сертифікованого провайдера [18, 23], що є критичним для легітимної роботи магазину;
- сучасні користувачі не готові чекати перезавантаження всієї сторінки при переході між категоріями. Саме тому використання фреймворків типу React [20] стає необхідністю, а не перевагою, оскільки це дозволяє реалізувати плавну зміну контенту, характерну для нативних мобільних додатків.

Висновки до розділу 1

У першому розділі проведено комплексне дослідження сфери електронної комерції з акцентом на нішу онлайн-ритейлу одягу. Було визначено, що даний сегмент ринку є одним із найбільш технологічно вимогливих, що зумовлено специфікою товарів та високими очікуваннями користувачів щодо візуалізації та швидкодії систем.

В результаті аналізу діяльності провідних світових платформ, таких як Nike, Adidas та ASOS, виявлено галузеві стандарти: використання мікросервісної архітектури [13], асиметричні підходи до навігації, акцент на мобільних користувачах та персоналізація контенту. Водночас було встановлено, що для малих та середніх підприємств існуючі готові рішення

часто є надлишковими за складністю або занадто дорогими в обслуговуванні, що підтверджує доцільність розробки індивідуального гнучкого вебзастосунку.

Детальний аналіз бізнес-процесів дозволив виокремити ключові етапи взаємодії користувача з системою та сформулювати вичерпний перелік функціональних і технічних вимог. Це стало підґрунтям для обґрунтування вибору сучасного технологічного стеку (React, Node.js, MSSQL), який здатний забезпечити необхідний рівень продуктивності, безпеки та масштабованості. Таким чином, результати розділу формують повну специфікацію для подальшого проєктування та програмної реалізації системи. Окремо візначимо, що зіставлення функціональних можливостей Nike, Adidas та ASOS [4, 5, 15] виявило характерну закономірність: розвинені рекомендаційні й логістичні підсистеми цих платформ розраховані на асортимент у сотні тисяч позицій і потребують відповідної інфраструктури, тоді як для нішевого магазину одягу надлишкова складність радше шкодить, ніж допомагає. Саме тому сформований перелік вимог свідомо обмежено ядром електронної комерції – каталогом, кошиком, оплатою та обробкою повернень, – що дає змогу досягти повноти функціоналу без невиправданих витрат на розроблення й супровід.

ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ

2.1 Обґрунтування технологічного інструментарію та методів розробки

У сучасних умовах розробка високонавантажених вебзастосунків для електронної комерції базується на використанні інструментарію, що забезпечує високу швидкість обробки даних та гнучкість архітектури. Аналіз сучасних наукових публікацій підтверджує, що найбільш ефективним підходом для створення систем онлайн-торгівлі є використання компонентно-орієнтованих бібліотек та асинхронних серверних середовищ.

Зокрема, у дослідженні [24], присвяченому розробці адаптивних вебзастосунків для магазинів одягу (Teras Thrifting Store), автори наголошують, що для забезпечення конкурентоспроможності критично важливою є висока швидкість відгуку інтерфейсу та його адаптивність під різні типи пристроїв. Це підтверджує доцільність вибору бібліотеки React для даного проєкту, оскільки її механізм Virtual DOM дозволяє оновлювати лише окремі компоненти сторінки (наприклад, фільтри товарів або кошик) без повного перезавантаження, що значно покращує користувацький досвід (UX).

Вибір React диктує перехід до архітектури SPA. На відміну від традиційних багатосторінкових застосунків (MPA), де кожен клік ініціює повне оновлення сторінки з боку сервера, SPA завантажує необхідні ресурси лише один раз. У подальшому клієнтська частина взаємодіє з сервером через асинхронні запити (AJAX/Fetch), отримуючи дані у форматі JSON.

Для магазину одягу це означає миттєву зміну кольору моделі або розміру без “миготіння” екрана, що є критичним для утримання уваги покупця.

Продуктивність: перенесення рендерингу на сторону клієнта знижує навантаження на сервер, дозволяючи йому зосередитися виключно на бізнес-логіці та роботі з базою даних.

Для забезпечення цілісності даних на фронтенді (наприклад, синхронізація кількості товарів у хедері та на сторінці кошика) у проєкті

використовується React Context API. Це дозволяє уникнути "prop drilling" (прокидання даних через усі рівні компонентів) і створює єдине джерело істини для станів авторизації та кошика.

Використання React у проєкті обґрунтовано наявністю наступних технологічних рішень [20]:

- декларативний підхід: Розробник описує кінцевий стан інтерфейсу, а бібліотека сама піклується про його оновлення. Це зменшує кількість помилок при маніпуляціях з DOM-деревом;
- JSX-синтаксис: Поєднання логіки JavaScript та розмітки HTML в одному файлі дозволяє створювати візуально зрозумілі компоненти, що прискорює підтримку коду;
- екосистема Hooks (useEffect, useState): Дозволяє керувати станом компонентів та життєвим циклом застосунку без використання громіздких класів.

Для об’єктивності вибору React було проведено порівняльний аналіз із альтернативними фреймворками, такими як Vue.js та Angular. Хоча Angular пропонує повний набір інструментів «з коробки», його високий поріг входження та надмірна складність для e-commerce проєкту середнього масштабу є недоцільними. Vue.js, у свою чергу, є легким, проте екосистема React (зокрема бібліотеки для роботи зі станом та маршрутизацією) є більш зрілою та має ширшу підтримку спільноти, що гарантує стабільність розробки інтернет-магазину.

Таблиця 2.1 – Порівняння фронтенд-фреймворків для розробки проєкту

Параметр	React	Vue.js	Angular
Архітектура	Компонентна (Virtual DOM)	Компонентна (Virtual DOM)	MVC (Real DOM)
Крива навчання	Середня	Низька	Висока

Продуктивність	Висока	Висока	Середня
Масштабованість	Дуже висока	Висока	Дуже висока

Крім того, наукова праця [21], яка розглядає впровадження алгоритмів рекомендацій для магазинів одягу на основі історії транзакцій, вказує на важливість структурованого збереження даних та можливості масштабування серверної частини. Це обґрунтовує використання платформи Node.js та фреймворку Express, які завдяки подійно-орієнтованій неблокуючій моделі вводу-виводу забезпечують високу продуктивність при одночасній обробці запитів від багатьох покупців.

На відміну від традиційних потокових серверів (наприклад, Apache), Node.js використовує Event Loop [16]. Це дозволяє серверу не очікувати завершення операції читання з бази даних або файлу, а переходити до обробки наступного запиту. Для e-commerce системи, де одночасно сотні користувачів можуть переглядати каталог, така архітектура запобігає зависанню сервера під навантаженням.

Фреймворк Express дозволяє реалізувати ланцюжок проміжних обробників (Middleware) [7], що є критичним для e-commerce:

- Logging Middleware (Morgan): для відстеження запитів та діагностики помилок;
- Security Middleware (Helmet, CORS): для захисту заголовків та обмеження доступу до API з неавторизованих доменів;
- Error Handling: централізована обробка помилок гарантує, що клієнт отримає зрозумілу відповідь навіть при критичному збої на сервері.

Для збереження даних про товари, замовлення та результати транзакцій обрано MSSQL [11]. На відміну від NoSQL-рішень, ця реляційна СКБД гарантує сувору цілісність даних (ACID), що є обов'язковим для фінансових операцій.

Транзакційність у разі збою під час оформлення замовлення, MSSQL гарантує, що дані або будуть записані повністю, або не будуть записані зовсім, що виключає появу "завислих" замовлень.

Sequelize – як інструмент абстракції: Взаємодія з базою реалізується через ORM Sequelize [22], що забезпечує надійний рівень абстракції та захист від SQL-ін'єкцій [17]. Використання моделей JavaScript замість прямих SQL-запитів робить код більш читабельним та дозволяє легко впроваджувати міграції –версійність структури бази даних.

При використанні MSSQL особлива увага приділяється нормалізації даних (до третьої нормальної форми), що дозволяє уникнути дублювання інформації про бренди та категорії одягу, забезпечуючи високу швидкість виконання JOIN-запитів.

Таблиця 2.2 – Порівняльний аналіз СКБД для проєкту

Параметр	MSSQL (обрано)	MySQL	PostgreSQL
Тип	Реляційна	Реляційна	Реляційна
Підтримка транзакцій	Повна (ACID)	Повна	Повна
Інтеграція з корпоративним ПЗ	Висока	Середня	Середня
Інструменти безпеки	Розширені (Row-level security)	Стандартні	Високі

Додатково, для забезпечення безпечних платежів інтегрується Stripe API [23], що дозволяє відповідати міжнародному стандарту PCI DSS, оскільки платіжні дані обробляються на стороні сервісу, не зберігаючись на сервері магазину.

Метод Tokenization замість передачі реальних даних картки через наш сервер, Stripe генерує унікальний токен. Це нівелює ризики крадіжки фінансової інформації користувачів у разі злому бази даних магазину.

Stripe Webhooks дозволяють серверу отримувати автоматичні сповіщення про статус платежу (успішно/відмова), що автоматизує зміну статусів замовлень у базі даних без втручання адміністратора.

Застосування вебхуків дозволяє системі автоматично реагувати на зовнішні події. Наприклад, як тільки Stripe підтверджує успішне списання коштів, сервер магазину миттєво змінює статус замовлення в базі MSSQL на «Оплачено» та ініціює процес бронювання товару на складі. Для тестування транзакційних сповіщень на етапі розробки використовується сервіс Mailtrap.io, який перехоплює вихідні листи й дозволяє перевіряти їхні шаблони, не надсилаючи повідомлень на реальні поштові скриньки користувачів.

Безпека системи гарантується використанням протоколу HTTPS [3] та авторизації на основі JSON Web Tokens (JWT) [10].

JWT механізм після успішної авторизації сервер генерує токен, який зберігається у клієнта (наприклад, у LocalStorage або HttpOnly Cookies). Це дозволяє реалізувати систему Stateless, коли серверу не потрібно зберігати стан сесії в пам'яті, що значно полегшує масштабування системи при збільшенні кількості серверних вузлів.

Хешування для захисту паролів користувачів використовується бібліотека bcrypt [19], яка застосовує алгоритм соління (salting), що унеможливлює злом методом перебору або за допомогою "райдужних таблиць".

Процес розробки базується на гнучких методологіях (Agile/Scrum) та принципі розділення відповідальностей (Separation of Concerns).

Scrum-підхід дозволяє розбити розробку на двотижневі спринти. Це забезпечує можливість постійного тестування функціоналу та швидкого внесення правок на основі проміжних результатів.

SOLID принципи при розробці серверної логіки застосовуються принципи SOLID [1]. Наприклад, принцип єдиної відповідальності (Single Responsibility) гарантує, що модуль, який відповідає за розрахунок знижок, не займається відправкою електронних листів. Це робить код легким для тестування та майбутнього розширення (наприклад, додавання нових платіжних систем або методів доставки).

REST API використання архітектурного стилю REST [8] забезпечує уніфікований інтерфейс для взаємодії між клієнтом та сервером, що в майбутньому дозволить легко підключити мобільний застосунок до існуючого бекенду без його переробки.

Для забезпечення цілісності вихідного коду та можливості командної розробки обрано розподілену систему керування версіями Git. Використання моделі розгалуження (наприклад, GitFlow) дозволяє ізолювати розробку нових функцій (feature-branches) від стабільної версії продукту (main branch), що мінімізує ризики появи критичних помилок у "продакшн" середовищі. Це особливо важливо при інтеграції платіжних шлюзів, де будь-яка помилка в коді може призвести до фінансових втрат.

2.2 Специфікація вимог до програмного забезпечення

1) Призначення та межі проєкту

1.1) Призначення системи

Розроблюваний вебзастосунок призначений для організації процесу продажу одягу через мережу Інтернет. Система забезпечує можливість перегляду каталогу товарів, пошуку та фільтрації продукції, формування кошика, оформлення замовлень, а також управління даними з боку адміністратора.

Застосунок орієнтований на спрощення взаємодії між продавцем і покупцем, а також автоматизацію основних бізнес-процесів інтернет-магазину.

1.2) Погодження, ухвалені в програмній документації

- використання клієнт-серверної архітектури;
- застосування REST API для обміну даними;
- використання сучасних вебтехнологій для забезпечення адаптивності;
- обмеження функціоналу базовими можливостями електронної комерції.

1.3) Межі проєкту

- система включає вебінтерфейс користувача та серверну частину;
- передбачено реалізацію основного функціоналу інтернет-магазину;
- передбачено інтеграцію з платіжною системою Stripe для здійснення онлайн-оплати;
- не включає розробку мобільного застосунку;
- не реалізується логістика доставки.

2) Загальний опис

2.1) Сфера застосування

Система використовується для організації онлайн-продажу одягу та може бути адаптована для інших типів товарів.

2.2) Характеристика користувачів

- користувачі (покупці):

мають базові навички роботи з вебсайтами, здійснюють перегляд товарів та покупки;

- адміністратори:

мають розширені права доступу, здійснюють управління товарами, замовленнями та користувачами;

- рівень підготовки:

користувачі – базовий; адміністратори – середній.

2.3) Загальна структура системи

Система складається з трьох основних компонентів:

- клієнтська частина (інтерфейс користувача);

- серверна частина (обробка запитів);
- база даних (зберігання інформації).

2.4) Загальні обмеження

- необхідність підключення до Інтернету;
- залежність швидкодії від серверного навантаження;
- обмеження на обсяг даних, що обробляються одночасно.

3) Функції системи

3.1) Реєстрація та авторизація

Опис функції

Забезпечує створення облікового запису користувача та доступ до системи.

Вхідні дані

Логін, пароль, email.

Вихідні дані

Обліковий запис користувача, токен доступу.

Функціональні вимоги

- перевірка коректності введених даних;
- збереження паролів у зашифрованому вигляді;
- підтримка сесії користувача.

3.2) Перегляд та пошук товарів

Опис функції

Надає можливість перегляду каталогу товарів.

Вхідні дані

Запит користувача, параметри фільтрації

Вихідні дані

Список товарів

Функціональні вимоги

- відображення товарів;
- фільтрація за категоріями, ціною, брендом;
- сортування результатів.

3.3) Робота з кошиком

Опис функції

Дозволяє додавати товари до кошика та змінювати їх кількість.

Вхідні дані

ID товару, кількість

Вихідні дані

Оновлений кошик.

Функціональні вимоги

- додавання та видалення товарів;
- зміна кількості;
- збереження стану кошика.

3.4) Оформлення замовлення

Опис функції

Забезпечує створення замовлення користувачем.

Вхідні дані

Дані користувача, товари з кошика.

Вихідні дані

Створене замовлення.

Функціональні вимоги

- формування замовлення;
- збереження даних у базі;
- відображення статусу замовлення.

3.5) Управління системою (адміністратор)

Опис функції

Надає можливість адміністрування системи.

Функціональні вимоги

- додавання та редагування товарів;
- управління замовленнями;
- управління користувачами.

3.6) Оплата замовлення

Опис функції

Забезпечує можливість оплати замовлення користувачем за допомогою інтеграції з платіжною системою Stripe.

Вхідні дані

ID замовлення, сума, платіжні дані користувача;

Вихідні дані

Статус оплати (успішно / помилка), ідентифікатор транзакції.

Функціональні вимоги

- створення платіжної сесії;
- передача даних до платіжного сервісу;
- обробка результату платежу;
- оновлення статусу замовлення після оплати.

3.7) Система фільтрації та сортування товарів

Опис функції

Дозволяє користувачу швидко знайти потрібний одяг за специфічними характеристиками.

Вхідні дані

Категорія товару, діапазон цін, розмір, колір, бренд.

Вихідні дані

Відфільтрований список товарів у реальному часі.

Функціональні вимоги

- підтримка множинного вибору параметрів;
- динамічне оновлення кількості знайдених товарів;
- можливість скидання всіх фільтрів одним натисканням.

4) Вимоги до інформаційного забезпечення

4.1) Джерела даних

- дані вводяться користувачами;
- дані створюються адміністратором;
- дані про платежі (статус, сума, ідентифікатор транзакції).

4.2) Довідкова інформація

- категорії товарів;
- бренди;
- статуси замовлень.

4.3) Організація даних

- використання реляційної бази даних;
- забезпечення цілісності даних;
- резервне копіювання.

5) Вимоги до технічного забезпечення

- персональний комп’ютер або сервер;
- доступ до мережі Інтернет;
- сучасний браузер.

6) Вимоги до програмного забезпечення

6.1) Архітектура

Клієнт-серверна

6.2) Системне ПЗ

Windows / Linux

6.3) Мережне ПЗ

HTTP / HTTPS

6.4) База даних

MSSQL

6.5) Технології

- React;
- Node.js;
- Express;
- MSSQL;
- Sequelize;
- Stripe API.

7) Вимоги до зовнішніх інтерфейсів

7.1) Інтерфейс користувача

- адаптивний дизайн;

- зручна навігація;
- простий доступ до функцій;
- інтеграція з зовнішнім платіжним API (Stripe).

7.2) Апаратний інтерфейс

Стандартні пристрої (ПК, смартфони).

7.3) Програмний інтерфейс

REST API

7.4) Протокол

HTTP / HTTPS

8) Властивості програмного забезпечення

8.1) Доступність

Система повинна бути доступною більшу частину часу.

8.2) Супроводжуваність

Модульна структура коду.

8.3) Переносимість

Можливість розгортання на різних серверах.

8.4) Продуктивність

Швидка обробка запитів.

8.5) Надійність

Стабільна робота без збоїв.

8.6) Безпека

- захист даних користувачів;
- використання захищених платіжних протоколів;
- передача платіжних даних через Stripe без збереження їх у системі.

8.7) Масштабованість та відмовостійкість

Система повинна підтримувати горизонтальне масштабування серверної частини (Node.js). Використання Stateless JWT дозволяє запускати декілька екземплярів сервера за балансувальником навантаження (наприклад, Nginx), що забезпечує безперебійну роботу магазину під час сезонних розпродажів або пікових навантажень.

9) Інші вимоги

- можливість розширення функціоналу;
- ведення логів;
- підтримка оновлень.

Висновки до розділу 2

У результаті виконання другого розділу було проведено комплексний аналіз та формування вимог до програмного забезпечення інтернет-магазину одягу. На основі аналізу сучасних тенденцій та наукових джерел [21, 24] було науково та технічно обґрунтовано вибір технологічного стеку, що базується на JavaScript-екосистемі (React, Node.js, Express) та реляційній системі управління базами даних MSSQL.

Ключові результати розділу включають:

- Обґрунтування архітектури: Встановлено, що для забезпечення високого показника UX (User Experience) у сфері Fashion e-commerce найбільш доцільним є використання архітектури Single Page Application (SPA). Це дозволяє мінімізувати час відгуку інтерфейсу при складних маніпуляціях з каталогом товарів;
- Вибір інструментів безпеки: Визначено, що поєднання JWT-авторизації, хешування паролів через bcrypt та інтеграції платіжного шлюзу Stripe забезпечує комплексний захист персональних та фінансових даних користувачів згідно з міжнародними стандартами (PCI DSS);
- Формування специфікації вимог: Сформовано детальний перелік функціональних та нефункціональних вимог. Визначено межі проєкту, які зосереджені на створенні повного циклу онлайн-продажів: від реєстрації клієнта до автоматизованої обробки платежу;
- Аналіз даних: Обґрунтовано використання MSSQL та ORM Sequelize як засобів забезпечення цілісності даних за стандартом ACID, що є критично важливим для уникнення помилок при одночасному оформленні замовлень великою кількістю користувачів.

Сформована специфікація вимог та обґрунтований технологічний інструментарій створюють надійний фундамент для подальших етапів розробки. Зокрема, результати цього розділу будуть використані в наступній частині роботи для детального проєктування структури бази даних, створення діаграм послідовностей та безпосередньої програмної реалізації модулів системи. Обрані методи розробки (Scrum, SOLID) гарантують, що створюваний продукт буде не лише функціональним, а й легким у супроводі та масштабуванні в умовах динамічного ринку електронної комерції. Визначені вимоги окреслюють функціональні межі проєкту, серед яких – обов'язкова інтеграція з платіжним шлюзом Stripe та впровадження рольової моделі доступу. Отримані результати є фундаментальною базою для переходу до етапу безпосереднього проєктування архітектури та бази даних. Слід наголосити, що обґрунтований технологічний стек – React із клієнтським станом на MobX, серверний прошарок на Express та доступ до даних через Sequelize поверх Microsoft SQL Server – добирався не за принципом популярності, а з огляду на однорідність мови розроблення та зрілість екосистеми. Єдина мова JavaScript на клієнті й сервері знижує когнітивне навантаження на розробника та спрощує повторне використання логіки валідації, а наявність докладної документації кожного з компонентів [7, 12, 20, 22] зменшує ризики на етапі реалізації. Такий виважений вибір інструментів закладає підґрунтя для подальшої підтримуваності й масштабованості системи.

МОДЕЛЮВАННЯ ВЕБЗАСТОСУНКУ

3.1 Розробка архітектури програмного забезпечення

Проєктування архітектури є фундаментом для створення масштабованої, безпечної та зручної в обслуговуванні інформаційної системи. Для розробки інтернет-магазину одягу було обрано трирівневу клієнт-серверну архітектуру, яка забезпечує чіткий розподіл відповідальності між інтерфейсом користувача, бізнес-логікою та рівнем збереження даних.

Архітектура системи базується на принципі Model-View-Controller на серверній стороні та компонентному підходу на клієнтській стороні:

1) клієнтський рівень (Frontend): Побудований на бібліотеці React. Це забезпечує високу швидкість відгуку інтерфейсу та можливість створення динамічних сторінок (Single Page Application). Взаємодіє з сервером здійснюється через REST API шляхом надсилання асинхронних HTTP-запитів;

2) рівень бізнес-логіки (Backend): Реалізований на середовищі виконання Node.js із застосуванням фреймворку Express. Цей рівень забезпечує обробку запитів, авторизацію користувачів, роботу з платіжним шлюзом Stripe та сервісом автоматичних розсилок Mailtrap.io;

3) рівень збереження даних (Database): Використовується Microsoft SQL Server. Для взаємодії з базою даних застосовується об’єктно-реляційне відображення (Sequelize ORM), що дозволяє керувати даними через об’єктні моделі, забезпечуючи цілісність інформації та безпеку запитів.

Вибір інструментарію зумовлений потребами продуктивності та безпеки:

- JWT (JSON Web Token): Використовується для забезпечення безпечної сесії користувача. Токен, що передається в заголовку Authorization, мінімізує навантаження на сервер, дозволяючи виконувати автентифікацію без постійного звернення до БД;

- Stripe SDK: Інтеграція платіжної системи через тестовий режим забезпечує безпечну обробку фінансових транзакцій, гарантуючи отримання статусу оплати перед зміною стану замовлення;
- Mailtrap.io: Застосовується як середовище для тестування транзакційних електронних листів, що дозволяє відлагоджувати процеси сповіщення клієнтів (підтвердження замовлення, реєстрація) без фактичного надсилання листів на реальні поштові скриньки.

Архітектура проєкту суворо дотримується принципів цілісності даних, зокрема:

Розмежування сутностей: Уникання надлишковості шляхом розділення Product та ProductVariant. Логіка зберігання товарів у кошику та замовленнях виключно через ProductVariant запобігає аномаліям оновлення цін чи описів;

Ізоляція даних замовлення: Після створення замовлення, дані про ціну OrderItem стають імутабельними, що гарантує збереження історії фінансових операцій навіть при зміні цін у каталозі товарів;

Сервісно-орієнтований підхід: Уся бізнес-логіка інкапсульована у відповідних контролерах та сервісах, що забезпечує легкість тестування та подальшого розширення системи (наприклад, додавання модулів аналітики чи локалізації).

3.2 Моделювання функцій та інформаційних потоків

Проєктування архітектури сучасного вебзастосунку неможливе без створення моделей, що формалізують функціональні вимоги та потоки даних. У даному підрозділі представлено комплексний опис логіки роботи інтернет-магазину за допомогою мови UML. Моделювання дозволяє на етапі проєктування виявити «вузькі місця» в бізнес-процесах, уникнути суперечностей у функціоналі та забезпечити високу якість програмної реалізації. Всі розроблені моделі базуються на специфікації вимог та враховують особливості роботи з сутностями товарів, кошиком та фінансовими транзакціями.

Діаграма (рис. 3.1) візуалізує межі системи "ClothingStore" та визначає ролі користувачів. Ключовим аспектом є розділення прав доступу: клієнт має доступ до транзакційних функцій (пошук, фільтрація, кошик, замовлення), тоді як адміністратор наділений розширеними правами на управління контентом (CRUD-операції з товарами) та модерацію відгуків. Використання зв’язку «include» для прецеденту «Оплата» підкреслює, що процес завершення замовлення є невід'ємною частиною оформлення покупки. Своєю чергою, відношення «extend» для прецеденту «Повернення товару» вказує на те, що дана функція є ініційованою подією, доступною лише для замовлень зі статусом «Виконано». Це дозволяє уникнути некоректних дій з боку користувача на етапах до фактичного отримання товару.

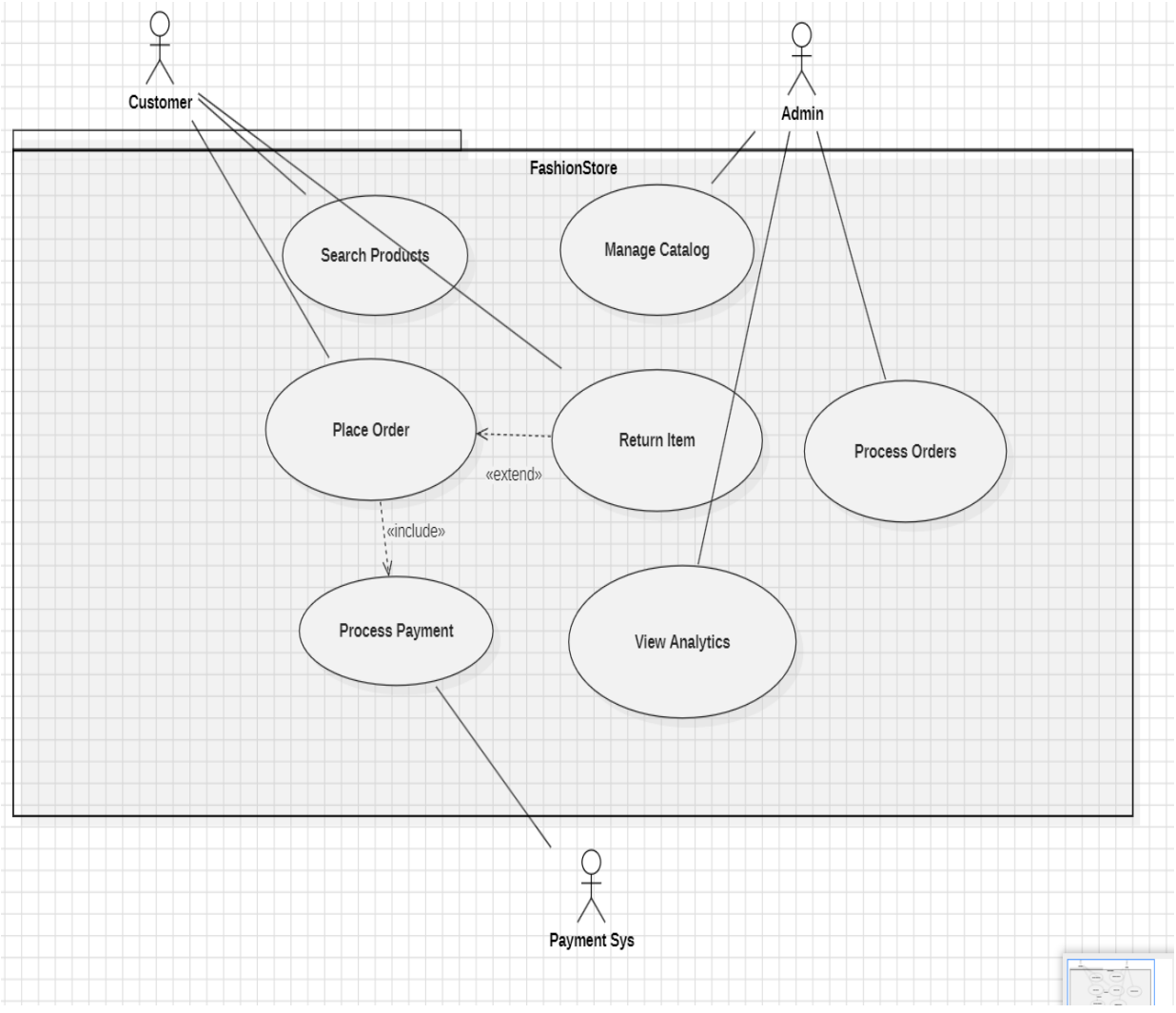

Рисунок 3.1 – Діаграма прецедентів

Діаграма класів (рис. 3.2) є основою для подальшої реалізації бази даних та архітектури серверної частини. . Вона деталізує структуру даних і демонструє зв’язки між сутностями User, Role, Product, ProductVariant, Cart, Order та OrderItem. Особливу увагу на діаграмі приділено розмежуванню Product та ProductVariant. Це рішення виключає надлишковість даних та дозволяє гнучко керувати параметрами (колір, розмір) без зміни опису товару. Також важливою є композиція Order -> OrderItem. Дана модель забезпечує імутабельність фінансових даних: ціна товару, зафіксована в OrderItem на момент створення замовлення, залишається незмінною, навіть якщо в основному каталозі Product ціна буде змінена. Це є критично важливим для забезпечення цілісності фінансової звітності та довіри користувачів. Крім того, діаграма ілюструє рольову модель, де Role пов'язана з User за принципом (1:N), що забезпечує гнучкість при управлінні правами доступу.

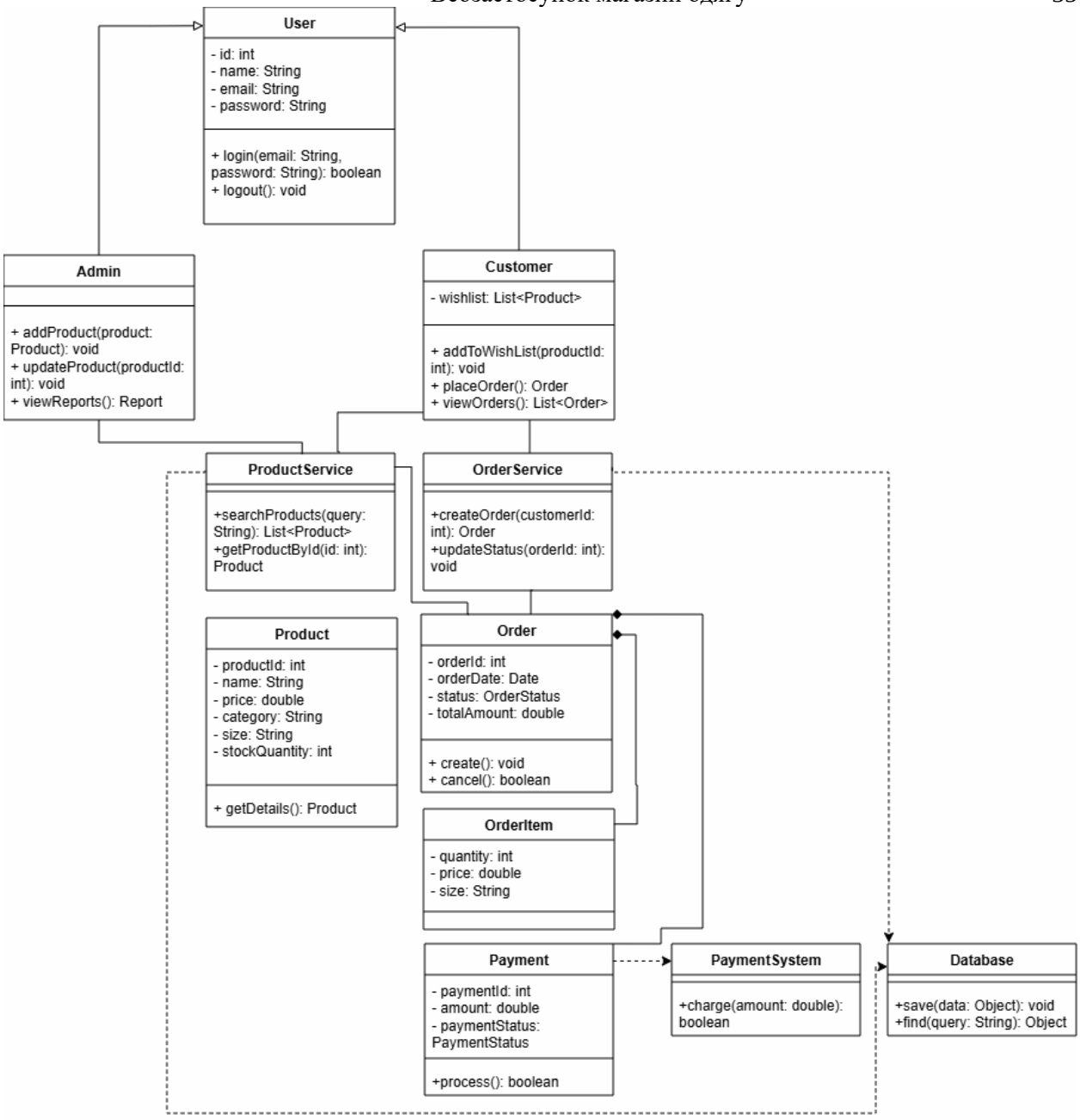

Рисунок 3.2 – Діаграма класів

Система динамічного моделювання включає серію діаграм послідовності та станів, які описують поведінку програмного забезпечення у часі та в залежності від дій користувача.

Процеси (Sequence Diagrams): Діаграми оформлення замовлення, оплати (Stripe) та повернення товару(див. рис 3.3 – 3.5) ілюструють покрокову передачу даних між WebUI, контролерами та сервісами.

Життєвий цикл (State Diagrams): Діаграми станів (див. рис 3.6 – 3.8) описують логіку зміни статусів замовлень та платежів, що є критично важливим для коректної роботи бізнес-логіки.

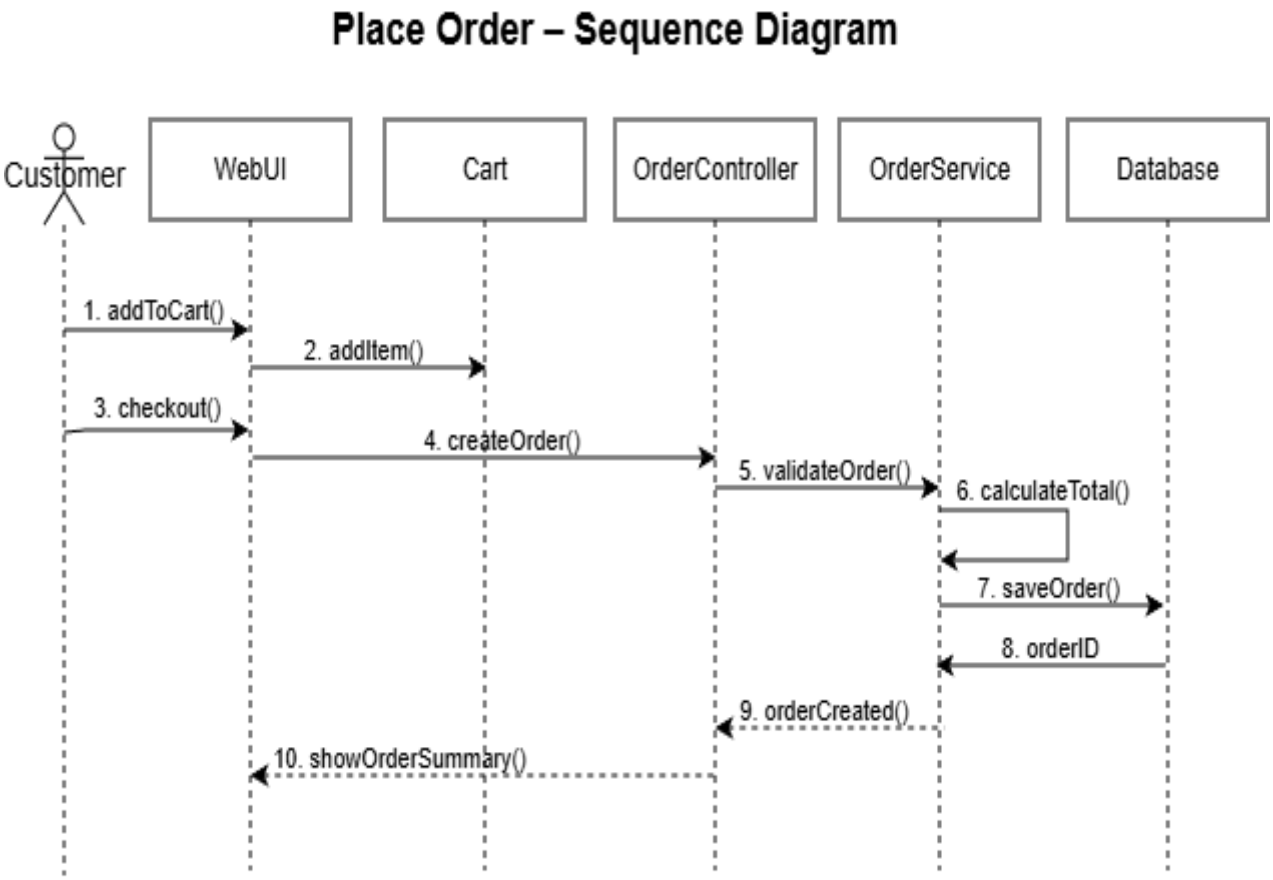

Рисунок 3.3 – Діаграма взаємодії процес оформлення замовлення.

Дана діаграма демонструє покрокову взаємодію клієнта з OrderController. Вона описує процес перевірки наявності товару на складі, формування запису в БД через OrderService та очищення кошика після успішного підтвердження. Це забезпечує атомарність операції оформлення замовлення.

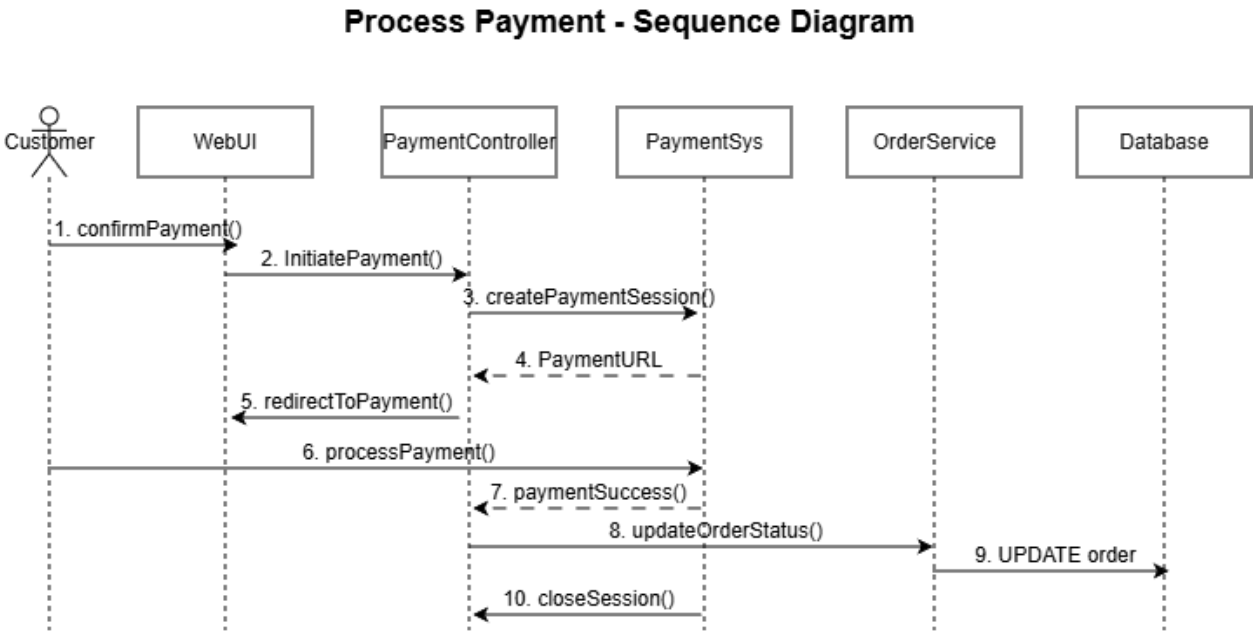

Рисунок 3.4 – Діаграма взаємодії процес оплати замовлення.

Діаграма фокусується на інтеграції зі Stripe API. Вона відображає обмін повідомленнями: запит на створення платіжної сесії, передачу JWT-токена для авторизації та обробку асинхронного callback-повідомлення (webhook). Модель наочно демонструє, як система обробляє відповідь від зовнішнього сервера, синхронізуючи статус транзакції зі станом замовлення.

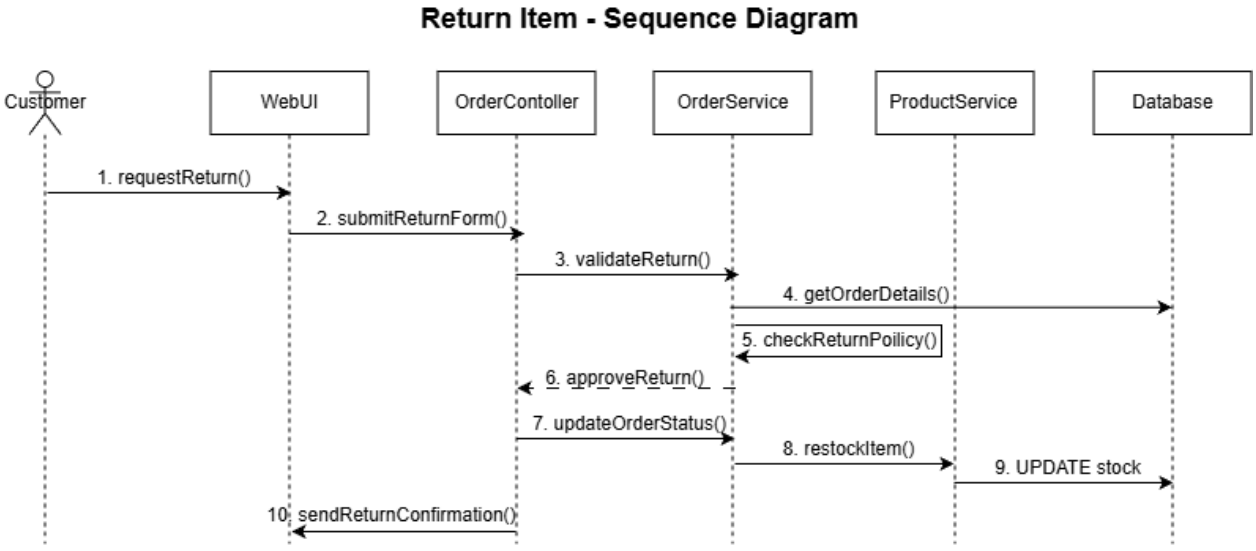

Рисунок 3.5 – Діаграма взаємодії процес повернення замовлення.

Модель описує алгоритм дій адміністратора. Вона відображає ланцюжок викликів: скасування транзакції в платіжному шлюзі, коригування залишків в ProductVariant та оновлення статусу в OrderService. Ця діаграма допомагає виявити всі необхідні сервіси, які мають бути сповіщені про зміну стану замовлення.

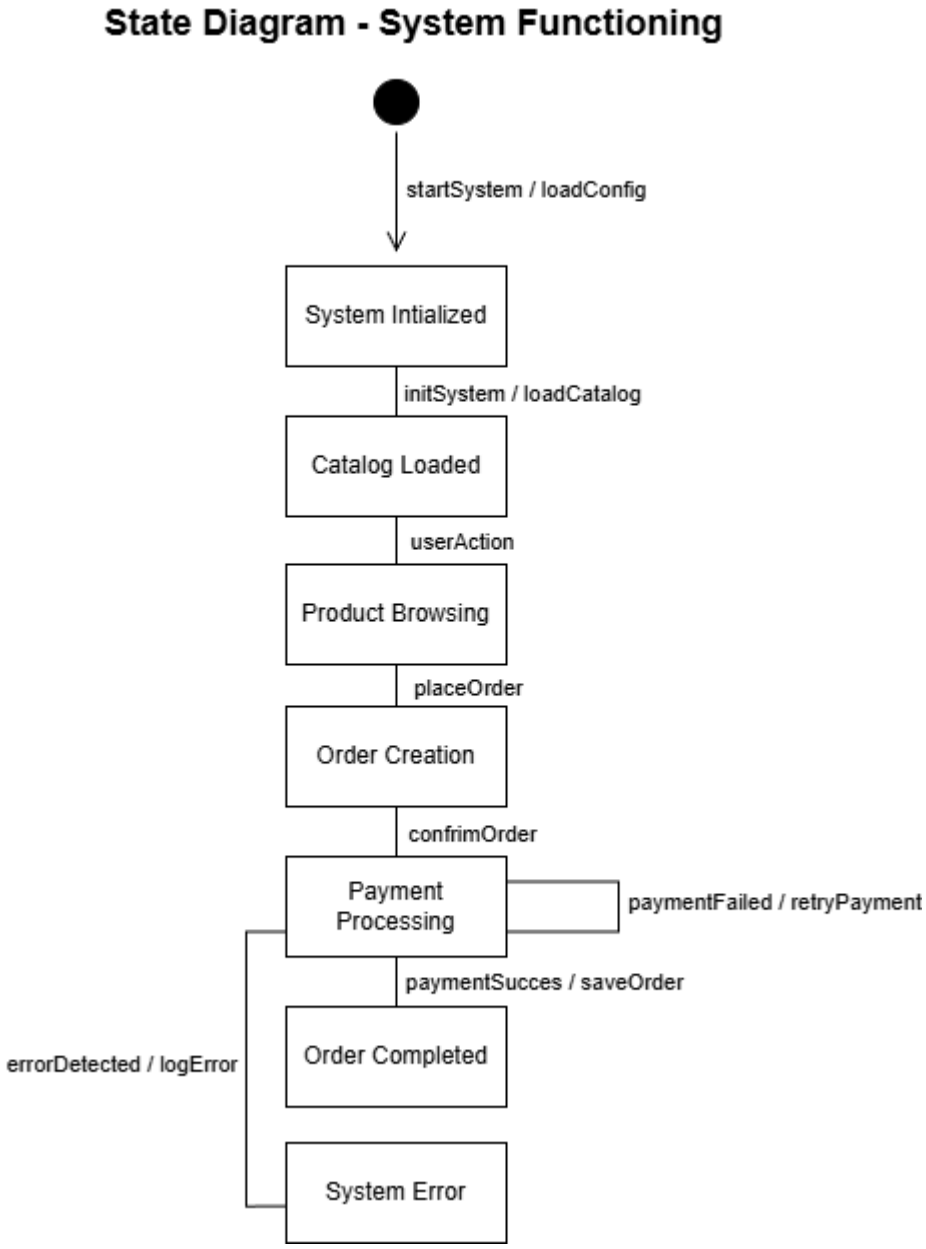

Рисунок 3.6 – Діаграма станів процес функціонування системи.

Діаграма формалізує життєвий цикл застосунку від ініціалізації до завершення сесії. Вона описує умови переходу між станами: «Анонімний

користувач» → «Авторизація» → «Клієнт» або «Адміністратор», що дозволяє уникнути несанкціонованого доступу.

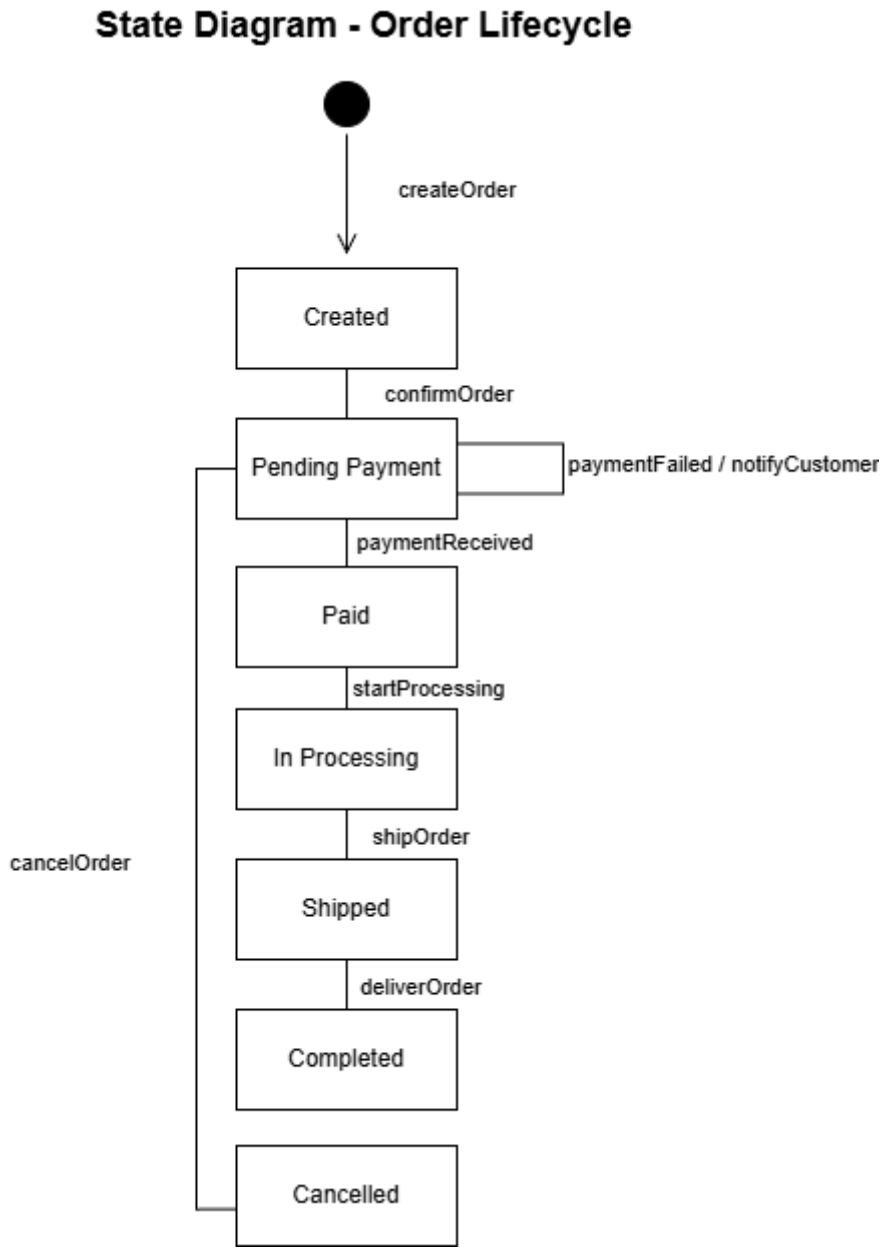

Рисунок 3.7 – Діаграма станів життєвий цикл замовлення.

Дана модель описує чітку послідовність станів замовлення: Створено → Оплачено → В обробці → Виконано. Діаграма жорстко обмежує можливі переходи, унеможливлюючи логічні помилки (наприклад, неможливість скасування відправленого замовлення). Це є механізмом захисту бізнес-логіки від "людського фактора".

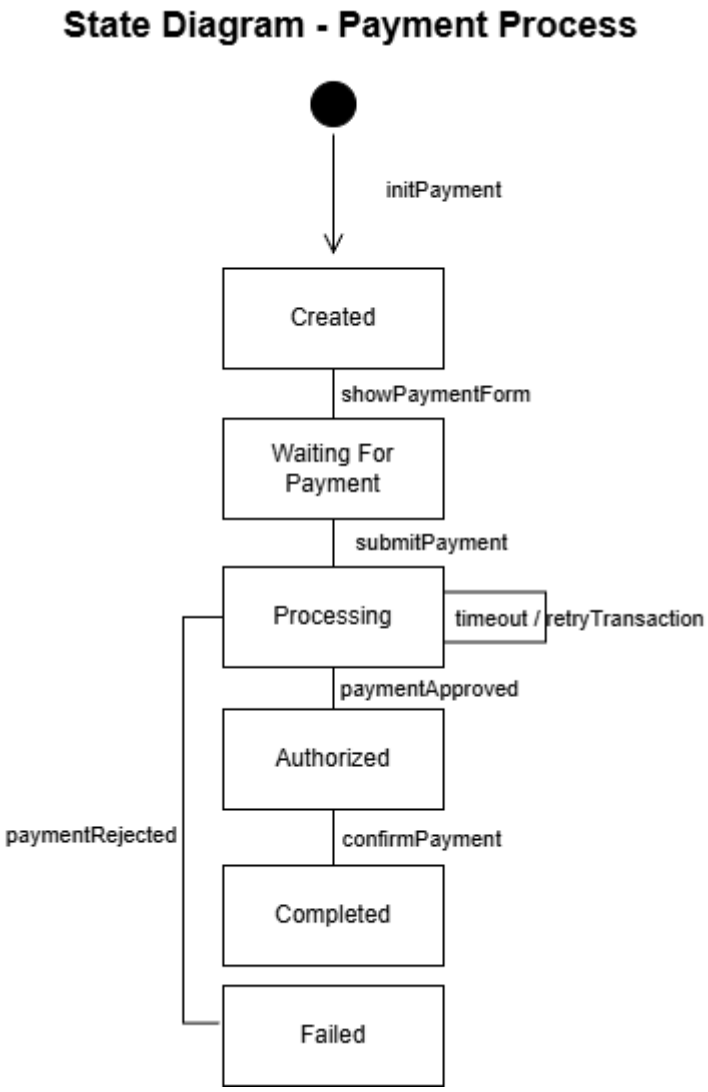

Рисунок 3.8 – Діаграма станів процес оплати замовлення.

Діаграма фокусується виключно на фінансовому стані транзакції (Pending -> Success -> Failed). Використання цієї моделі дозволяє обробити виключні ситуації, наприклад, коли оплата не пройшла через недостатність коштів, автоматично повертаючи замовлення у стан «Очікування оплати».

Моделі наведені у цьому розділі ілюструють покрокові сценарії роботи ключових функцій:

Процес пошуку (рис. 3.9): Описує шлях від введення запиту до відображення детальної інформації, враховуючи розвилки (якщо товар не знайдено).

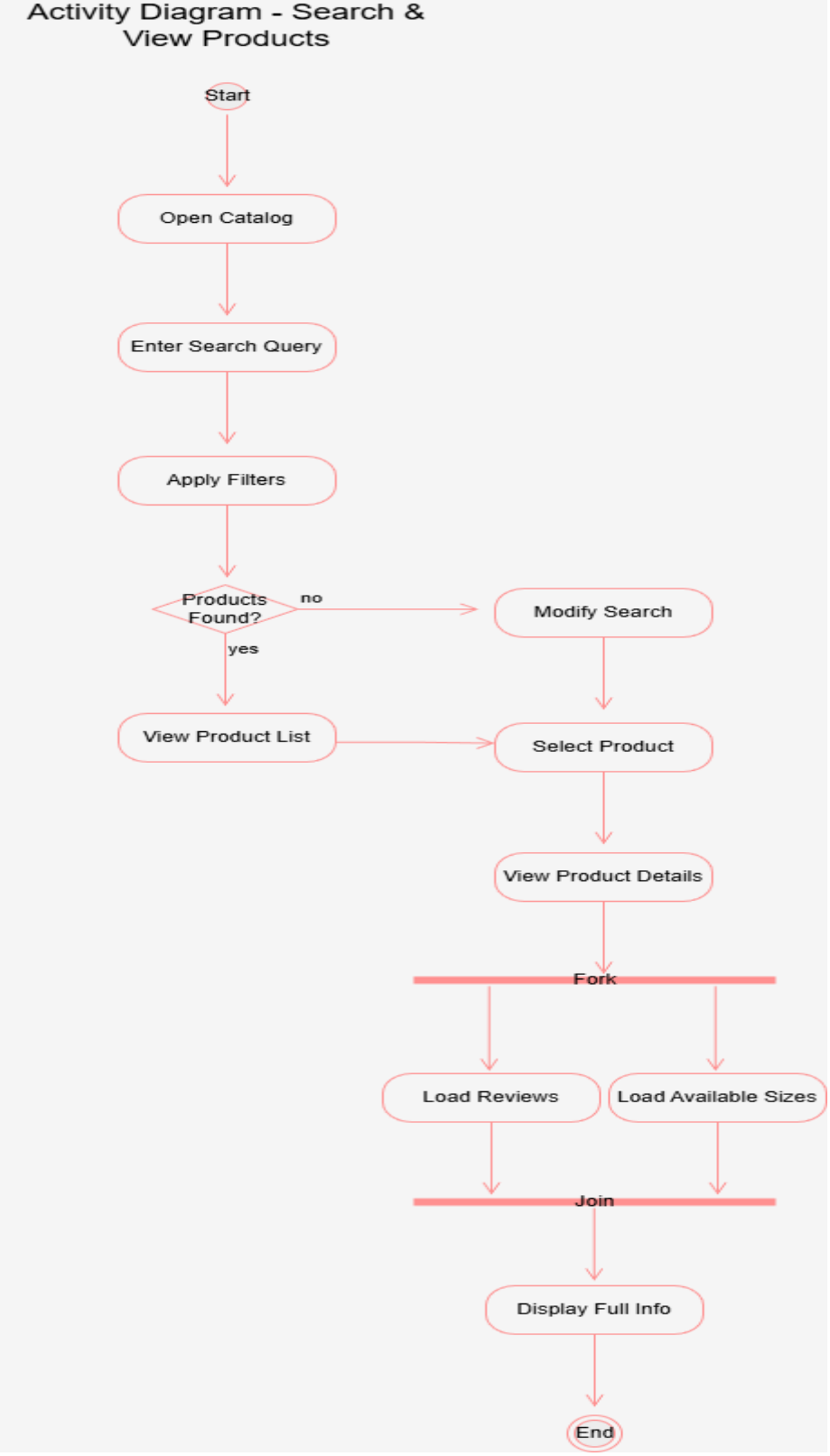

Рисунок 3.9 – Діаграма діяльності процес пошуку та перегляду товарів.

Процес оформлення замовлення (рис. 3.10): Візуалізує послідовність дій від кошика до резервування товарів. Використання елементів Fork/Join

наочно показує паралельність обчислень (розрахунок суми та резервування товару).

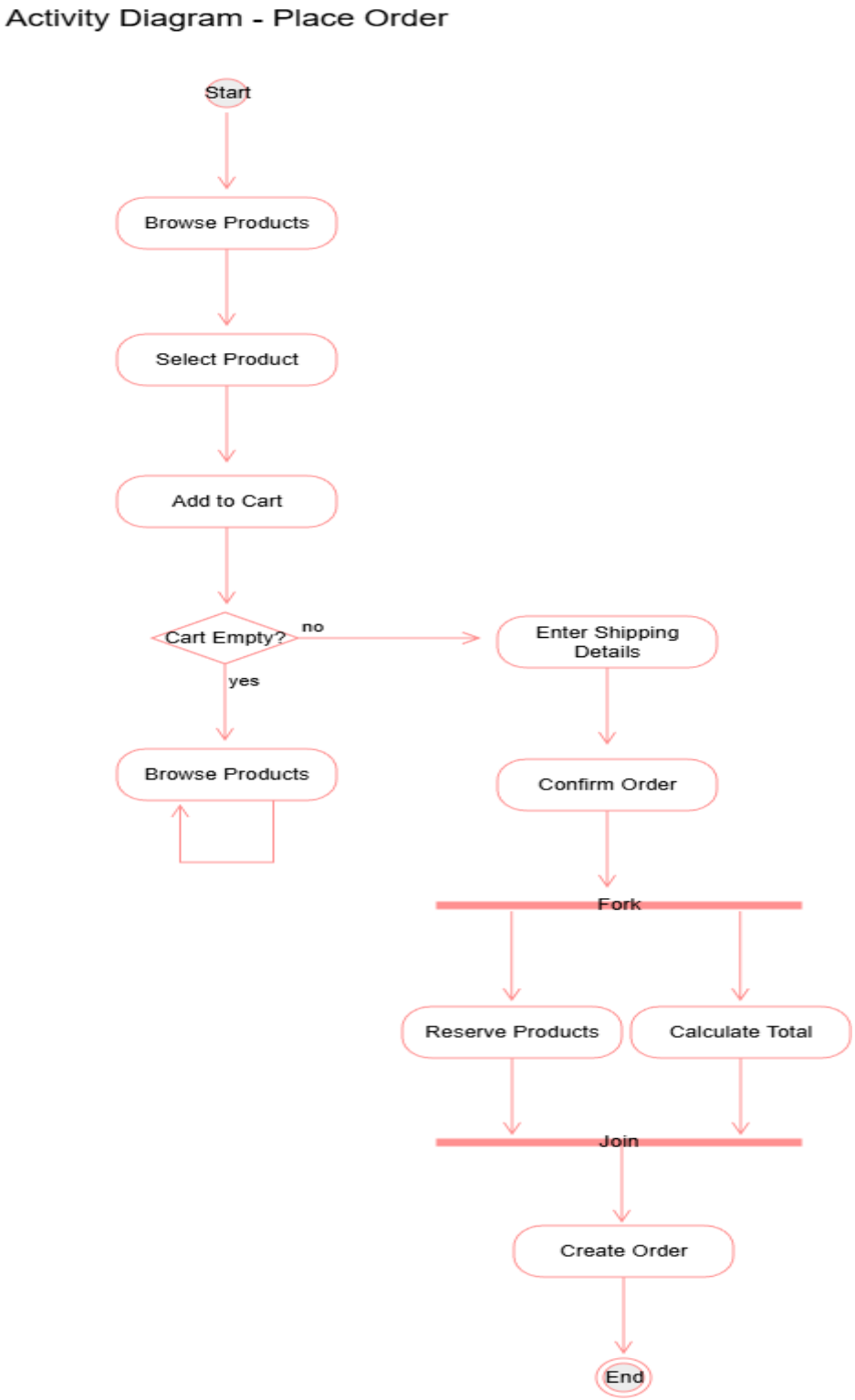

Рисунок 3.10 – Діаграма діяльності процес оформлення замовлення.

Процес оплати (рис. 3.11): Фокусується на бізнес-логіці транзакцій, включаючи цикли повторної спроби (Retry Payment) у разі невдачі.

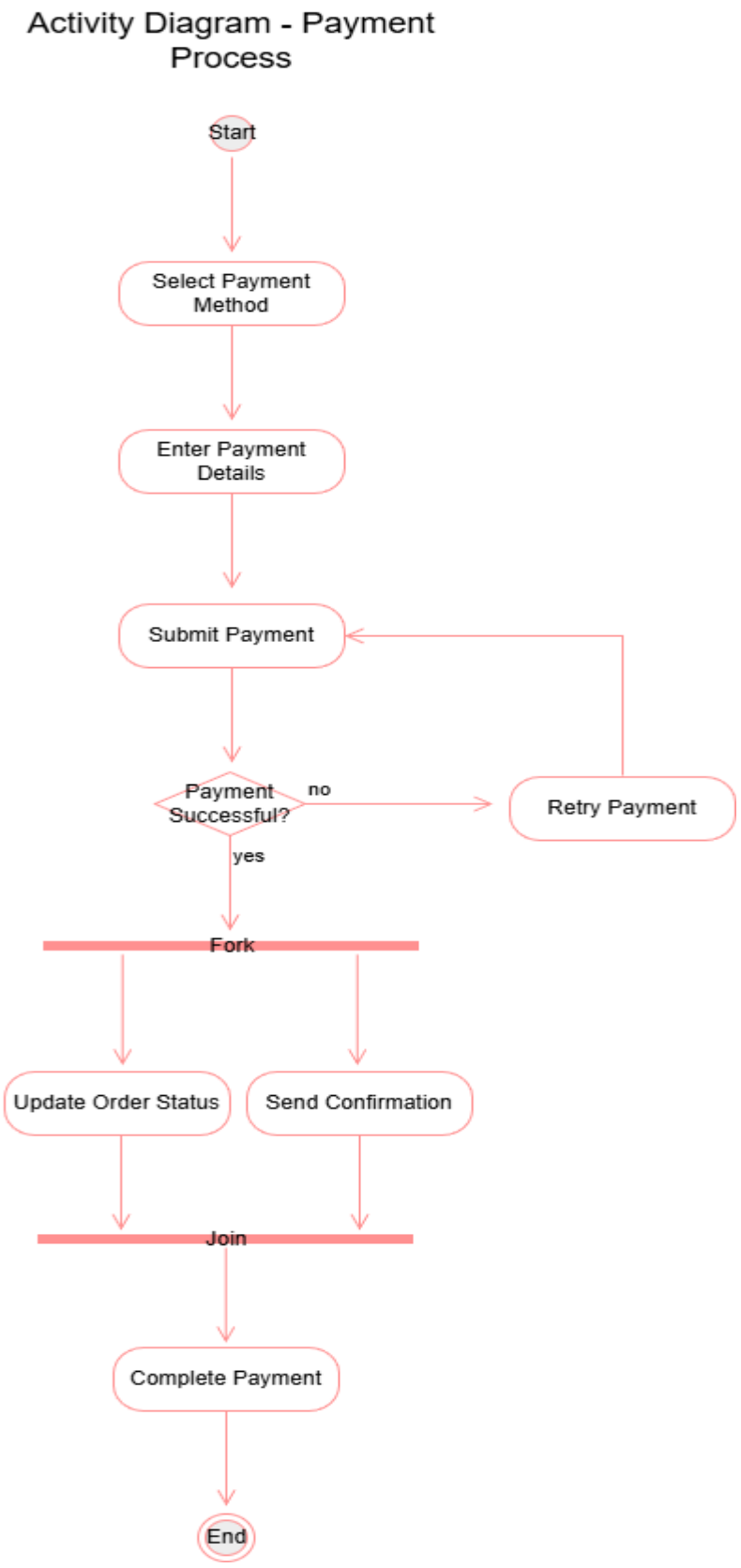

Рисунок 3.11 – Діаграма діяльності процес оплати замовлення.

3.3 Компонентна архітектура та модель розгортання системи

Технологічний стек, обґрунтований у підрозділі 2.1, покладено в основу проєктування архітектури програмного забезпечення. У цьому підрозділі увагу зосереджено не на повторному виборі інструментів, а на тому, як обрані технології організовано у вигляді взаємопов'язаних компонентів і як ці компоненти розміщуються в середовищі виконання. Статичну структуру системи описано за допомогою діаграми компонентів, а її фізичне розгортання – за допомогою діаграми розгортання.

На рисунку 3.12 представлено діаграму компонентів, яка відображає фізичну структуру програмного забезпечення та логічні зв’язки між його складовими частинами. Архітектура застосунку спроєктована за модульним принципом, що забезпечує чітке розмежування відповідальності між шарами системи.

Основні вузли архітектури включають:

- Web Client (React): Клієнтська частина, яка реалізує інтерфейс користувача та логіку взаємодії з клієнтом (State Management, маршрутизація);
- Backend API (Node.js/Express): Серверний компонент, що виконує роль центрального вузла обробки бізнес-логіки, валідації запитів та керування сесіями;
- Database (MS SQL Server): Реляційна система управління базами даних, що забезпечує транзакційну цілісність, зберігання даних про користувачів, товари та замовлення. Взаємодія з БД здійснюється через Sequelize ORM, що підвищує безпеку запитів;
- Mail Service (Mailtrap.io): Зовнішній сервіс, інтегрований у систему для автоматизації транзакційних сповіщень та розсилки листів (реєстрація, підтвердження замовлення);
- Payment Service (Stripe API): Зовнішній компонент платіжної інфраструктури, що забезпечує безпечну обробку фінансових транзакцій.

Інтеграція реалізована через SDK, що дозволяє ізолювати логіку оплати від внутрішніх сервісів магазину.

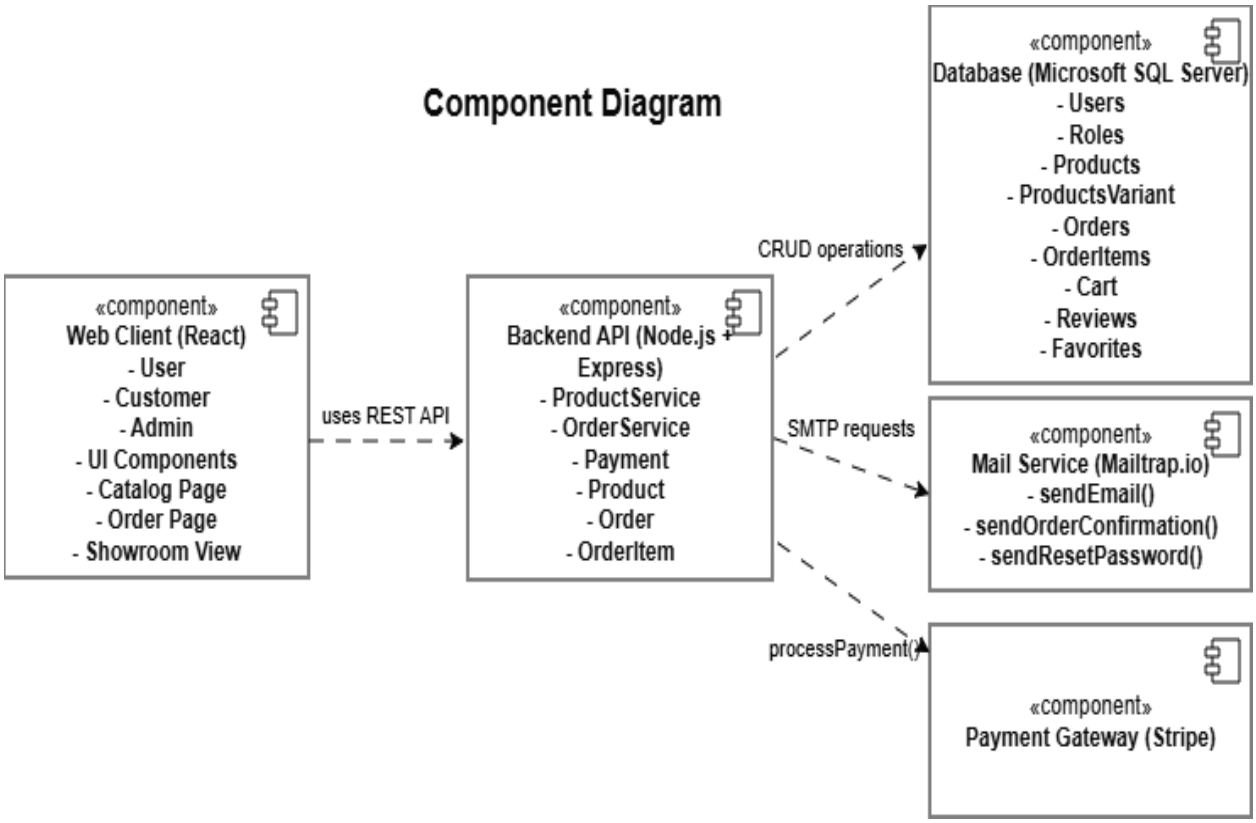

Рисунок 3.12 – Діаграма компонентів системи.

Дана модель підтверджує модульність системи: кожен компонент має визначений інтерфейс взаємодії, що дозволяє проводити оновлення окремих частин (наприклад, зміну платіжного провайдера чи сервісу сповіщень) без необхідності повної перебудови архітектури всього програмного забезпечення.

Діаграма розгортання рисунок 3.13 ілюструє фізичне розміщення програмних компонентів на апаратному та мережевому рівнях. Вона показує, як система взаємодіє з оточенням під час виконання.

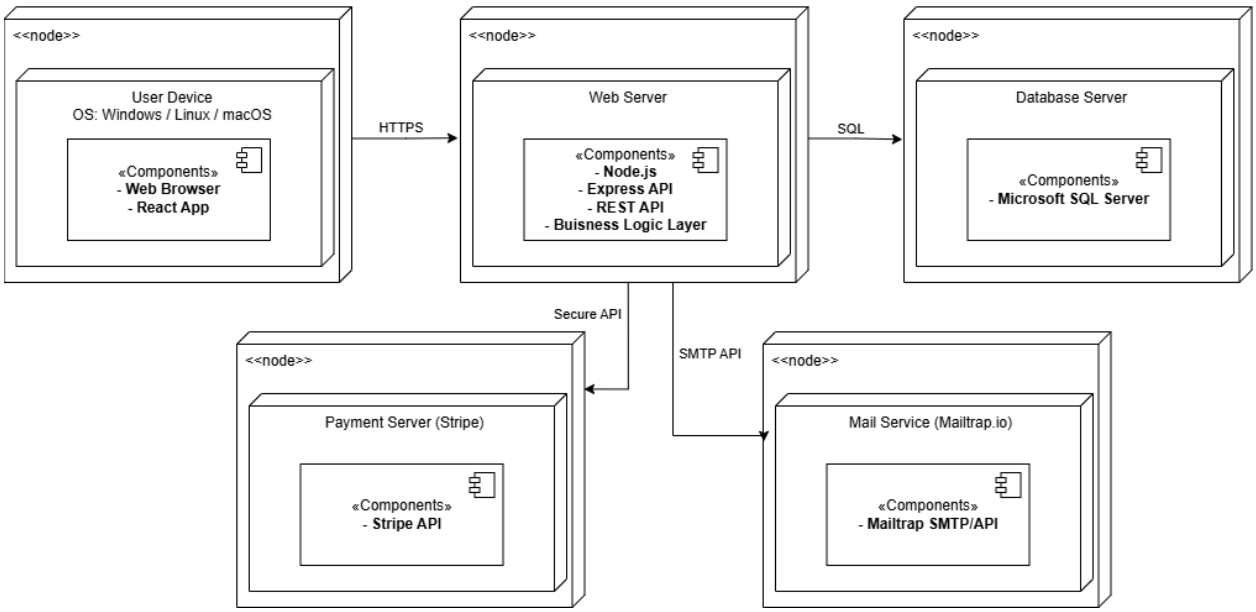

Рисунок 3.13 – Діаграма розгортання ПЗ

Архітектура розподілена між вузлами:

− Client Node (Web Browser): Вузол кінцевого користувача, де виконується клієнтська частина застосунку (React SPA). Взаємодія з сервером відбувається через протокол HTTPS;

− Application Server: Серверне середовище виконання Node.js, на якому розгорнуто API-сервер. Цей вузол виступає посередником, що приймає запити від клієнта та обробляє їх;

− Database Server (MS SQL Server): Виділений сервер бази даних, з яким API-сервер підтримує стабільне з'єднання через драйвери, що забезпечує ізоляцію даних та підвищену продуктивність;

− Mail Gateway (Mailtrap.io): Зовнішній вузол, до якого сервер звертається через SMTP або API для надсилання сповіщень;

− Payment Gateway (Stripe Server): Зовнішній вузол платіжної системи. Взаємодія здійснюється через зашифрований HTTPS-канал для ініціалізації сесій оплати та отримання підтверджень про транзакції.

Використання такої діаграми демонструє, що архітектура є готовою до масштабування. Поділ на логічні вузли дозволяє в майбутньому перенести базу даних або API-сервер на окремі потужності (Cloud instances), що забезпечує стабільну роботу інтернет-магазину при зростанні навантаження.

3.4 Опис інтерфейсів ПЗ

Інтерфейс користувача розроблений з урахуванням принципів інтуїтивності та швидкості навігації.

Каталог товарів: Реалізований як динамічний компонент, що підтримує «ліниве» завантаження (lazy loading). Кожна картка товару містить інтерактивні елементи для вибору розміру та кольору, які динамічно змінюють стан ProductVariant у внутрішній пам'яті.

Кошик: Інтерфейс кошика реалізує патерн Observer –при зміні кількості товару, загальна вартість перераховується миттєво без потреби оновлення сторінки.

Адмін-панель: Це окремий модуль застосунку, доступ до якого обмежений на рівні Middleware. Інтерфейс адмін-панелі включає табличне представлення даних з можливістю сортування та фільтрації, що дозволяє адміністратору ефективно керувати асортиментом та обробляти вхідні замовлення.

Висновки до розділу 3

У третьому розділі було розроблено проєктні рішення, що забезпечують реалізацію функціональних вимог до програмного забезпечення інтернет-магазину одягу.

По-перше, обґрунтовано вибір трирівневої архітектури, яка забезпечує високу масштабованість системи та чітке розмежування відповідальності між клієнтською та серверною частинами.

По-друге, проведено UML-моделювання системи, що дозволило формалізувати бізнес-процеси та структуру даних, мінімізуючи ризики виникнення логічних помилок на етапі реалізації.

По-третє, обрано сучасний технологічний стек, який задовольняє вимоги до безпеки (JWT, Stripe) та стабільності роботи (MS SQL Server, Sequelize).

По-четверте, на рівні моделі даних закладено механізми збереження цілісності інформації: розмежування сутностей Product та ProductVariant усуває надлишковість і дає змогу керувати розміром та кольором незалежно від опису товару, тоді як фіксація ціни в OrderItem на момент оформлення робить історію фінансових операцій незмінною навіть за подальшої зміни цін у каталозі.

Розроблені моделі та архітектурні рішення створюють надійну базу для подальшого програмного впровадження системи, що дозволяє її легко модифікувати та розширювати новим функціоналом, таким як аналітичні модулі чи інтеграція зі службами доставки.

Розроблені діаграми послідовності та станів слугували безпосередньою інструкцією для написання коду контролерів та сервісів. Завдяки детальному моделюванню станів замовлення та оплати, вдалося досягти високої стабільності бізнес-логіки, що мінімізує помилки при взаємодії з користувачем та зовнішніми платіжними сервісами.

4.1 Опис розроблених класів та модулів

Розроблений програмний застосунок “ClothingStore” є вебзастосунком електронної комерції, який реалізовано за клієнт-серверною архітектурою з розподілом на три логічні рівні. Рівень подання (React-клієнт), рівень бізнес-логіки (Node.js, Express-сервер) та рівень збереження даних (СКБД Microsoft SQL Server, доступ через ORM-бібліотеку Sequelize).

Технологічний стек:

1) клієнтська частина:
   - React 19, React-Router-DOM 7 – побудова SPA та маршрутизація;
   - MobX 6, mobx-react-lite – управління глобальним станом застосунку [12];
   - React-Bootstrap 2, Bootstrap 5.3 – компонентна бібліотека UI;
   - Axios – HTTP-клієнт для взаємодії з REST API;
   - jwt-decode – декодування токенів автентифікації.
2) серверна частина:
   - Express 4 – HTTP-сервер та маршрутизація;
   - Sequelize 6 + tedious – ORM для Microsoft SQL Server;
   - bcrypt – хешування паролів;
   - jsonwebtoken (JWT) – токени автентифікації;
   - nodemailer – поштовий сервіс;
   - Stripe API – обробка платежів;
   - express-fileupload – приймання зображень товарів;
   - node-cron – періодичні фонові завдання.

Програмний застосунок реалізує предметну область інтернет-магазину одягу та підтримує функціональні можливості, описані у таблиці 4.1.

Таблиця 4.1 – Функціональні можливості застосунку

№	Функція	Роль користувача
1	Перегляд каталогу товарів з фільтрацією та сортуванням	Гість, користувач
2	Пошук товарів за назвою	Гість, користувач
3	Реєстрація та автентифікація (JWT)	Гість
4	Відновлення пароля через електронну пошту	Користувач
5	Управління кошиком (додавання, зміна, видалення)	Користувач
6	Оформлення замовлення з оплатою через Stripe	Користувач
7	Перегляд історії замовлень з деталізацією	Користувач
8	Запит на повернення товару	Користувач
9	Список обраних товарів	Користувач
10	Залишення відгуків і оцінок до товарів	Користувач
11	Перемикання світлої/темної теми оформлення	Гість, користувач
12	CRUD-операції над товарами, категоріями, брендами	Адміністратор
13	Управління відгуками користувачів	Адміністратор
14	Обробка запитів на повернення товарів	Адміністратор

Інформаційна модель застосунку складається з тринадцяти сутностей, описаних у вигляді ORM-моделей Sequelize. Опис сутностей наведено у таблиці 4.2.

Таблиця 4.2 – Сутності інформаційної моделі

Сутність	Призначення	Ключові атрибути
User	Користувач системи	id, email, password, name, resetToken, resetTokenExpiry

Кінець таблиці 4.2

Role	Роль користувача	id, name (USER/ADMIN)
Product	Товар каталогу	id, name, price, description, img
ProductVariant	Варіант товару (розмір/колір)	id, size, color, stock
Category	Категорія товарів	id, name
Brand	Бренд-виробник	id, name
Cart	Кошик користувача	id (зв'язок 1:1 з User)
CartItem	Позиція в кошику	id, quantity
Order	Замовлення	id, totalPrice, status, phone, address
OrderItem	Позиція замовлення	id, quantity, price
Favorite	Запис обраного	id (зв'язки User↔Product)
Review	Відгук про товар	id, text, rating
OrderReturn	Запит на повернення	id, reason, status, adminComment

Типи даних, використані у моделях:

- DataTypes.INTEGER – числові ідентифікатори, кількості;
- DataTypes.STRING – назви, адреси, телефони (до 255 символів);
- DataTypes.TEXT – довгі рядки (тексти відгуків, причини повернення);
- DataTypes.FLOAT – грошові суми;
- DataTypes.DATE – мітки часу;
- Symbol-ключі бібліотеки Sequelize (Op.gte, Op.lte, Op.like, Op.gt) – оператори у WHERE-умовах.

В якості констант у системі застосовуються:

- перелічувані статуси замовлень: PENDING, PAID, SHIPPED, DELIVERED, CANCELLED, RETURNED;

- статуси запитів на повернення: PENDING, APPROVED, REJECTED, REFUNDED;
- ролі користувачів: USER, ADMIN;
- проміжок дії токена скидання пароля – 1 година (3 600 000 мс);
- термін автоматичного скасування неоплачених замовлень – 24 години.

Решта значень (ціни, рейтинги, тексти) є динамічними і зберігаються у базі даних.

Клієнт надсилає HTTPS-запити до REST API серверу за префіксом /api/. Доступ до захищених маршрутів регулюється проміжним middleware authMiddleware, який перевіряє JWT-токен у заголовку Authorization: Bearer <token>. Адміністративні маршрути додатково проходять через checkRoleMiddleware('ADMIN').

Стан клієнта розділено на п'ять MobX-сховищ: UserStore (автентифікація), ProductStore (каталог), BasketStore (кошик), FavoritesStore (обране), ThemeStore (тема оформлення). Сховища поширюються через React Context.

Контролер ReviewController реалізує CRUD-операції над відгуками. Склад методів:

- create(req, res) – створення відгуку, перевіряє унікальність (один відгук на товар від одного користувача);
- getByProduct(req, res) – повернення списку відгуків для конкретного товару з даними автора;
- getAll(req, res) – повернення всіх відгуків (адміністративний);
- getLatest(req, res) – повернення останніх N відгуків для головної сторінки;
- update(req, res) – редагування з перевіркою прав (автор або адміністратор);
- delete(req, res) – видалення з аналогічною перевіркою прав.

Контролер OrderController обробляє життєвий цикл замовлення: створення з інтеграцією Stripe, отримання історії, скасування, обробка webhook-подій від платіжної системи, оновлення статусу.

Контролер ReturnController обробляє запити на повернення: створення (доступне лише для замовлень зі статусами PAID/SHIPPED/DELIVERED), перегляд власних запитів, адміністративний перегляд усіх, оновлення статусу з автоматичним надсиланням листа клієнту.

Сервіс MailService інкапсулює взаємодію з SMTP через nodemailer. Містить методи sendWelcomeMail, sendOrderMail, sendPasswordResetMail, sendReturnStatusMail, sendActivationMail. Усі методи використовують внутрішній _send, що обробляє помилки, не перериваючи основний потік виконання.

Модуль міграцій runMigrations виконує ідемпотентне додавання колонок до існуючих таблиць через QueryInterface.addColumn, обходячи обмеження sequelize.sync({alter:true}) на СКБД MSSQL.

Клас BasketStore містить приватне поле _basket (масив позицій кошика), методи setBasket, clear та геттери:

- basket: CartItem[] – повний список;
- count: number – сума quantity усіх позицій (для бейджа на іконці кошика);
- totalPrice: number – підсумкова сума.

Клас FavoritesStore зберігає список ID товарів, доданих в обране. Геттер has(productId) дає змогу будь-якій картці товару миттєво визначити стан "лайку".

Клас ThemeStore управляє темою оформлення. У конструкторі читає збережене значення з localStorage або визначає системну тему через window.matchMedia('(prefers-color-scheme: dark)'). Методи setDark(bool) та toggle() зберігають вибір користувача.

Клас UserStore містить дані автентифікованого користувача (_user, _isAuth) із геттерами user та isAuth.

4.2 Лістинги основних компонентів програмного забезпечення

У цьому розділі наведено чотири ключові лістинги, що відображають як серверну, так і клієнтську логіку системи. Вони демонтсрують реалізацію віладції даних, розмежування прав доступу, динамічного формування запитів до бази даних та реактивного керування станом.

Лістинг контроллеру відгуків відповідає за створення та видалення коментарів до товарів. Метод create отримує ідентифікатор автора з токена авторизації й виконує послідовну валідацію: перевіряє заповненість полів, коректність оцінки (від 1 до 5), наявність товару та відсутність раніше залишеного відгуку від цього ж користувача. Метод delete реалізує розмежування прав доступу – видалити запис може лише його автор або адміністратор. Обробку помилок організовано через конструкцію try–catch із поверненням відповідних HTTP-статусів, що дозволяє клієнтській частині коректно реагувати на кожну ситуацію.

Лістинг – Контролер відгуків reviewController.js

```
const { Review, User, Product } = require('../models');
class ReviewController {
  async create(req, res) {
    try {
      const userId = req.user.id;
      const { productId, text, rating } = req.body;
      if (!text || !rating || !productId)
        return res.status(400).json({ message: "Заповніть всі поля відгуку" });
      const numericRating = Number(rating);
      if (numericRating < 1 || numericRating > 5)
        return res.status(400).json({ message: "Оцінка має бути від 1 до 5" });
      const product = await Product.findByPk(productId);
```

```
            if (!product)
                return res.status(404).json({ message: "Товар не знайдено" });
            const existing = await Review.findOne({
                where: { UserId: userId, ProductId: productId }
            });
            if (existing)
                return res.status(400).json({ message: "Ви вже залишили відгук"
});
            const review = await Review.create({
                text, rating: numericRating,
                UserId: userId, ProductId: productId
            });
            const full = await Review.findByPk(review.id, {
                include: [{ model: User, attributes: ['id','email','name'] }]
            });
            return res.json(full);
        } catch (e) {
            res.status(500).json({ message: "Помилка при створенні відгуку"
});
        }
    }
    async delete(req, res) {
        const { id } = req.params;
        const review = await Review.findByPk(id);
        if (!review) return res.status(404).json({ message: "Не знайдено" });

        const isAuthor = review.UserId === req.user.id;
        const isAdmin  = req.user.role === 'ADMIN';
        if (!isAuthor && !isAdmin)
            return res.status(403).json({ message: "Немає доступу" });
```

```
        await review.destroy();
        return res.json({ message: "Відгук видалено" });
    }
}
module.exports = new ReviewController();
```

У лістингу наведено сервіс електронної пошти, винесений в окремий модуль згідно з патерном Service Layer. Параметри підключення до SMTP-сервера зчитуються зі змінних середовища, тому конфіденційні дані не зберігаються безпосередньо в коді. Допоміжний метод _send обгортає надсилання листа в try–catch: у разі збою помилка лише фіксується в журналі й не перериває основний бізнес-процес. Методи sendPasswordResetMail і sendReturnStatusMail формують HTML-шаблони листів, а словник labels зіставляє технічні статуси повернення зі зрозумілими користувачеві формулюваннями.

Лістинг – Сервіс електронної пошти mailService.js

```
const nodemailer = require('nodemailer');
const FROM = '"ClothingStore" <no-reply@clothingstore.com>';

class MailService {
    constructor() {
        this.transporter = nodemailer.createTransport({
            host: process.env.SMTP_HOST,
            port: process.env.SMTP_PORT,
            secure: false,
            auth: { user: process.env.SMTP_USER, pass: process.env.SMTP_PASSWORD }
        });
    }
    async _send(to, subject, html) {
```

```
        try {
            await this.transporter.sendMail({ from: FROM, to, subject, html });
            console.log(`[MAIL] sent "${subject}" -> ${to}`);
        } catch (e) {
            console.error(`[MAIL] failed "${subject}" -> ${to}:`, e.message);
        }
    }
    async sendPasswordResetMail(to, link) {
        await this._send(to, 'Відновлення пароля ClothingStore', `
            <div style="font-family: sans-serif; max-width: 560px;">
                <h2>Відновлення пароля</h2>
                <p>Натисніть кнопку, щоб задати новий пароль. Посилання діє
1 годину.</p>
                <a href="${link}" style="background:#111;color:#fff;
                    padding:10px 20px;border-radius:8px;text-decoration:none;">
                    Скинути пароль
                </a>
            </div>
        `);
    }
    async sendReturnStatusMail(to, returnId, orderId, status, comment) {
        const labels = {
            PENDING: 'отримано і очікує розгляду',
            APPROVED: 'схвалено',
            REJECTED: 'відхилено',
            REFUNDED: 'оброблено, кошти повернуто'
        };
        await this._send(to, `Запит №${returnId} – оновлення статусу`, `
            <h2>Статус повернення оновлено</h2>
            <p>Запит <strong>#${returnId}</strong> по замовленню
```

```
            <strong>#${orderId}</strong> ${labels[status] || status}.</p>
          ${comment ? `<p>Коментар: <em>${comment}</em></p>` : ''}
        `);
    }
}
module.exports = new MailService();
```

Лістинг ілюструє метод getAll, який поєднує фільтрацію, сортування та посторінкове виведення каталогу. Умова вибірки where формується динамічно залежно від переданих параметрів – бренду, категорії та часткового збігу назви. Фільтр за ціною використовує оператори Op.gte й Op.lte, а факт його застосування фіксується окремим прапорцем hasPrice, оскільки ключами цих умов є символи (Symbol), для яких стандартна перевірка властивостей об'єкта не спрацьовує. Відбір за наявністю на складі реалізовано приєднанням сутності ProductVariant з умовою stock > 0, а напрям сортування задається через словник orderMap. Виклик findAndCountAll із параметрами distinct: true та subQuery: false забезпечує правильний підрахунок записів за наявності приєднаних таблиць.

Лістинг – Фільтрація товарів productController.getAll

```
async getAll(req, res) {
    try {
        let { brandId, categoryId, limit, page, searchTerm,
              priceFrom, priceTo, sort, inStock } = req.query;
        page = Number(page) || 1;
        limit = Number(limit) || 9;
        const offset = page * limit - limit;
        const where = {};
        if (brandId)    where.BrandId    = brandId;
        if (categoryId) where.CategoryId = categoryId;
        if (searchTerm) where.name = { [Op.like]: `%${searchTerm}%` };
        const priceCondition = {};
```

```
        let hasPrice = false;
        if (priceFrom !== undefined && priceFrom !== '' &&
!isNaN(Number(priceFrom))) {
          priceCondition[Op.gte] = Number(priceFrom); hasPrice = true;
        }
        if (priceTo !== undefined && priceTo !== '' &&
!isNaN(Number(priceTo))) {
          priceCondition[Op.lte] = Number(priceTo); hasPrice = true;
        }
        if (hasPrice) where.price = priceCondition;
        const wantInStock = inStock === 'true' || inStock === '1';
        const include = [
          { model: Brand },
          { model: Category },
          { model: ProductVariant, required: wantInStock,
            where: wantInStock ? { stock: { [Op.gt]: 0 } } : undefined }
        ];
        const orderMap = {
          price_asc:  [['price','ASC']],  price_desc: [['price','DESC']],
          name_asc:   [['name','ASC']],   name_desc:  [['name','DESC']],
          newest:     [['createdAt','DESC']], oldest: [['createdAt','ASC']]
        };
        const order = orderMap[sort] || [['id','DESC']];
        const products = await Product.findAndCountAll({
          where, limit, offset, include, order,
          distinct: true, subQuery: false
        });
        return res.json(products);
      } catch (e) {
        res.status(500).json({ message: "Помилка при отриманні товарів" });}}
```

Лістинг демонтрує перемикання теми оформлення засобами бібліотеки MobX. Стан зберігається у спостережуваному сховищі ThemeStore, тож його зміна автоматично оновлює всі підписані компоненти. У конструкторі початкова тема визначається за збереженим вибором у localStorage, а за його відсутності – за системним налаштуванням через медіазапит prefers-color-scheme. Метод setDark одночасно оновлює стан і зберігає його, завдяки чому вибір теми не втрачається між сеансами. Наведений фрагмент useEffect застосовує тему до DOM, додаючи клас dark та атрибут data-bs-theme, що керує стилізацією компонентів Bootstrap.

Лістинг – Сховище теми ThemeStore.js та його застосування

```
import { makeAutoObservable } from "mobx";
const STORAGE_KEY = 'cs_theme';
export default class ThemeStore {
  constructor() {
    this._isDark = false;
    try {
      const saved = localStorage.getItem(STORAGE_KEY);
      if (saved === 'dark') this._isDark = true;
      else if (saved === 'light') this._isDark = false;
      else if (window.matchMedia('(prefers-color-scheme: dark)').matches)
        this._isDark = true;
    } catch (e) { /* ignore */ }
    makeAutoObservable(this);
  }
  setDark(bool) {
    this._isDark = bool;
    try { localStorage.setItem(STORAGE_KEY, bool ? 'dark' : 'light'); }
catch(e){}
  }
  toggle() { this.setDark(!this._isDark); }
```

```
    get isDark() { return this._isDark; }
}
useEffect(() => {
    const root = document.documentElement;
    const body = document.body;
    if (theme.isDark) {
        body.classList.add('dark');
        root.setAttribute('data-bs-theme', 'dark');
    } else {
        body.classList.remove('dark');
        root.setAttribute('data-bs-theme', 'light');
    }
}, [theme.isDark]);
```

4.3 Тестування програмного забезпечення

Для перевірки коректності функціонування застосунку обрано комбінований підхід, що включає три рівні тестування:

1) модульне (Unit) тестування – для перевірки окремих функцій валідації даних (наприклад, перевірка діапазону рейтингу $1 \leq rating \leq 5$, перевірка достатності полів при створенні запиту).

2) інтеграційне тестування API – для перевірки взаємодії маршрутів, контролерів та СКБД. Виконувалося із застосуванням HTTP-клієнта Postman за принципом «чорної скриньки»: формувалися запити з різними наборами вхідних параметрів та перевірялися отримані відповіді.

3) ручне функціональне тестування користувацького інтерфейсу – за сценаріями користувача в браузерах Chrome 130 та Mozilla Firefox 132. Перевірялись реакція інтерфейсу, коректність переходів, стан кошика, дії з обраним, оплата.

Сформовано 24 тестових сценарії, що покривають критичні бізнес-функції. Перелік основних сценаріїв та їх результати наведено в таблиці 4.3.

Таблиця 4.3 – Результати функціонального тестування

№	Тестовий сценарій	Очікуваний результат	Фактичний	Статус
Т-01	Реєстрація з валідним email і паролем	Створення акаунта, отримання JWT, welcome-лист	Збігається	Виконано
Т-02	Реєстрація з існуючим email	HTTP 400, повідомлення про дублікат	Збігається	Виконано
Т-03	Авторизація з невірним паролем	HTTP 400	Збігається	Виконано
Т-04	Запит скидання пароля	HTTP 200, лист зі унікальним токеном	Збігається	Виконано
Т-05	Перехід за посиланням скидання (валідний токен)	Сторінка зміни пароля	Збігається	Виконано
Т-06	Перехід за простроченим токеном (>1 год)	HTTP 400, відмова	Збігається	Виконано
Т-07	Додавання товару в кошик	Запис у CartItem, оновлення лічильника	Збігається	Виконано
Т-08	Зміна кількості товару	Оновлення quantity, перерахунок суми	Збігається	Виконано
Т-09	Фільтрація по ціні від=500, до=1500	Лише товари у діапазоні	Збігається	Виконано
Т-10	Сортування price_desc	Сортування за спаданням ціни	Збігається	Виконано
Т-11	Фільтр «Тільки в наявності»	Лише товари з ProductVariant.stock > 0	Збігається	Виконано
Т-12	Оформлення замовлення	Створення Order, перехід на Stripe Checkout, лист	Збігається	Виконано
Т-13	Успішна оплата (webhook)	Зміна статусу на PAID	Збігається	Виконано

Кінець таблиці 4.3

Т-14	Скасування замовлення PENDING	Зміна статусу на CANCELLED	Збігається	Виконано
Т-15	Скасування замовлення PAID	HTTP 400, відмова	Збігається	Виконано
Т-16	Запит на повернення для PAID-замовлення	Створення OrderReturn зі статусом PENDING	Збігається	Виконано
Т-17	Адмін схвалює повернення (APPROVED)	Оновлення статусу, лист користувачу	Збігається	Виконано
Т-18	REFUNDED повернення	Order.status → RETURNED	Збігається	Виконано
Т-19	Додавання товару в обране	Запис у Favorite, бейдж +1	Збігається	Виконано
Т-20	Додавання відгуку (валідні дані)	Створення Review, оновлення середньої оцінки	Збігається	Виконано
Т-21	Спроба додати другий відгук	HTTP 400	Збігається	Виконано
Т-22	Видалення чужого відгуку звичайним користувачем	HTTP 403	Збігається	Виконано
Т-23	Видалення відгуку адміністратором	HTTP 200, видалення	Збігається	Виконано
Т-24	Перемикання теми та збереження вибору	Тема зберігається при перезавантаженні	Збігається	Виконано

Усі 24 тестових сценарії завершилися успішно. Виявлені на ранніх ітераціях розробки помилки, які були виправлені до фінального тестування:

1) Помилка фільтрації за ціною. Умова Object.keys(priceCondition).length повертала 0, оскільки Op.gte/Op.lte є Symbol-ключами, а метод Object.keys повертає лише string-ключі. Виправлення: введено явний прапорець hasPrice.

2) Відсутність асоціації Product↔OrderItem. При відображенні історії замовлень запит з include: [Product] падав з виключенням Sequelize. Додано двосторонню асоціацію та виконано міграцію OrderItems.ProductId через QueryInterface.addColumn.

3) Несумісність sync({alter:true}) з MSSQL. Sequelize генерував невалідний SQL ALTER COLUMN ... UNIQUE, не підтримуваний Microsoft SQL Server. Замінено на ідемпотентний модуль runMigrations, що додає колонки через QueryInterface.

4) Відсутність id товару у відповіді API кошика. Атрибут attributes: ['name','price','img'] не містив id, через що клік на зображення товару у кошику призводив до переходу за маршрутом /product/undefined. Додано id у перелік атрибутів.

5) CSS-змінні теми не застосовувалися. Файл index.css не імпортувався у точку входу index.js. Виправлено імпортом.

4.4 Результати рішення

Таблиця 4.4 – Метрики розробленого програмного забезпечення

Показник	Значення
Кількість рядків коду серверної частини	≈ 1 350
Кількість рядків коду клієнтської частини	≈ 3 100
Кількість REST-ендпоінтів	32
Кількість сутностей бази даних	13
Кількість сторінок (React-маршрутів)	11

Кінець таблиці 4.4

Кількість MobX-сховищ	5
Кількість шаблонів електронних листів	4
Розмір production-збірки клієнта (gzip)	≈ 163 КБ JS + 32 КБ CSS

Виміри виконано локально, час відображає лише серверну обробку (без мережевих затримок). Тестова вибірка містила 100 товарів, 50 замовлень, 30 відгуків.

Таблиця 4.5 – Середній час відповіді ключових ендпоінтів

Ендпоінт	Метод	Середній час, мс
/api/product (без фільтрів)	GET	18
/api/product (всі фільтри + сортування)	GET	27
/api/cart	GET	22
/api/order (історія)	GET	35
/api/reviews/:productId	GET	14
/api/order (створення + Stripe)	POST	420

Час створення замовлення обумовлений мережевим запитом до Stripe API.

Розроблений застосунок повною мірою реалізує заплановані функціональні можливості. Інтерфейс адаптивний (Bootstrap grid), підтримує дві теми оформлення зі збереженням вибору користувача. Захист API забезпечується JWT-автентифікацією та middleware перевірки ролей. Обробка платежів делегована сертифікованому провайдеру Stripe, що знімає вимоги PCI DSS. Реалізовано асинхронне надсилання електронних листів, що не блокує відповідь API.

4.5 Керівництво користувача

Розділ описує порядок роботи зі застосунком для трьох ролей: гостя, авторизованого користувача та адміністратора.

Для початку роботи необхідно у браузері відкрити URL http://localhost:3000. Завантажується головна сторінка з банером, рекомендованими товарами та відгуками покупців (Рисунок 4.1).

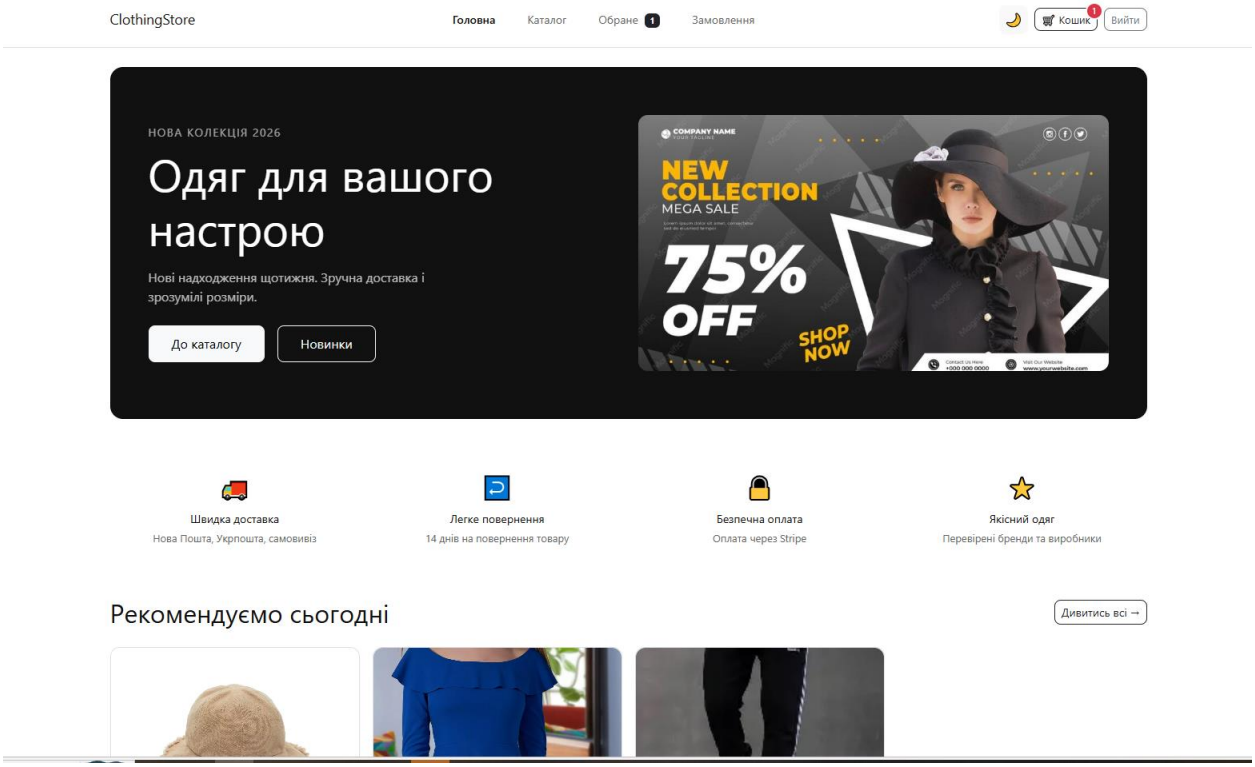

Рисунок 4.1 – Головна сторінка застосунку

Для реєстрації натиснути кнопку «Авторизація» у верхній правій частині, перейти за посиланням «Зареєструйся». У формі вказати ім'я (необов'язково), електронну пошту та пароль (мінімум 6 символів). Після успішної реєстрації на вказану адресу надходить вітальний лист (Рисунок 4.2).

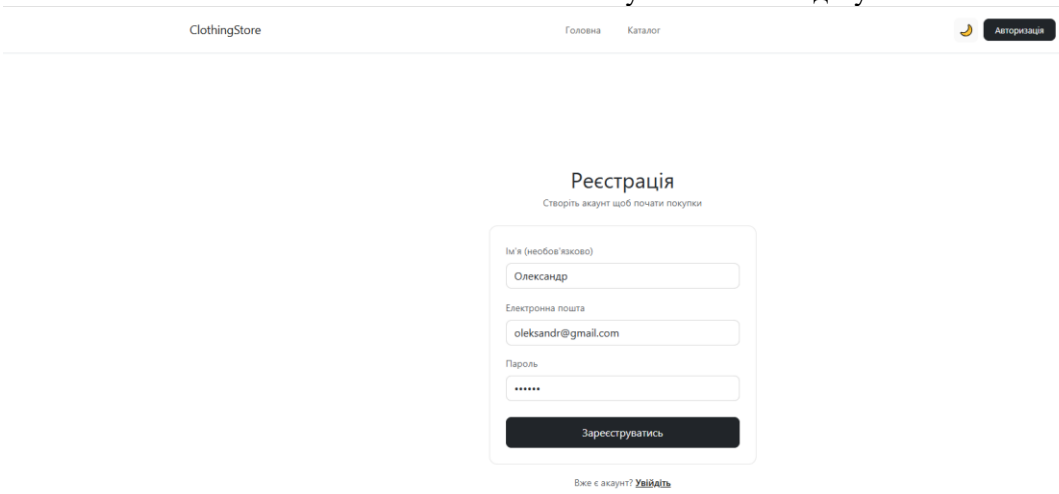

Рисунок 4.2 – Форма реєстрації

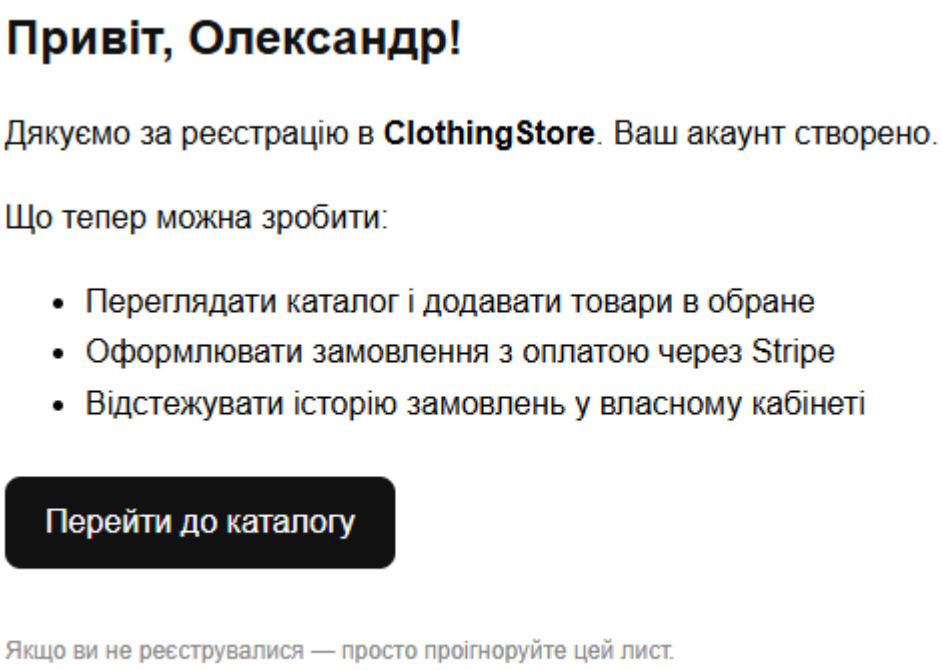

Рисунок 4.3 – Вітальний лист

Перехід у каталог здійснюється через пункт «Каталог» у головному меню. Користувачеві доступні:

- рядок пошуку за назвою товару;
- випадне меню сортування (6 варіантів);

– кнопка «Фільтри», що відкриває панель з фільтрами за категорією, брендом, ціновим діапазоном та наявністю на складі;

– візуальні «чіпи» активних фільтрів, які можна окремо вимкнути (Рисунок 4.4).

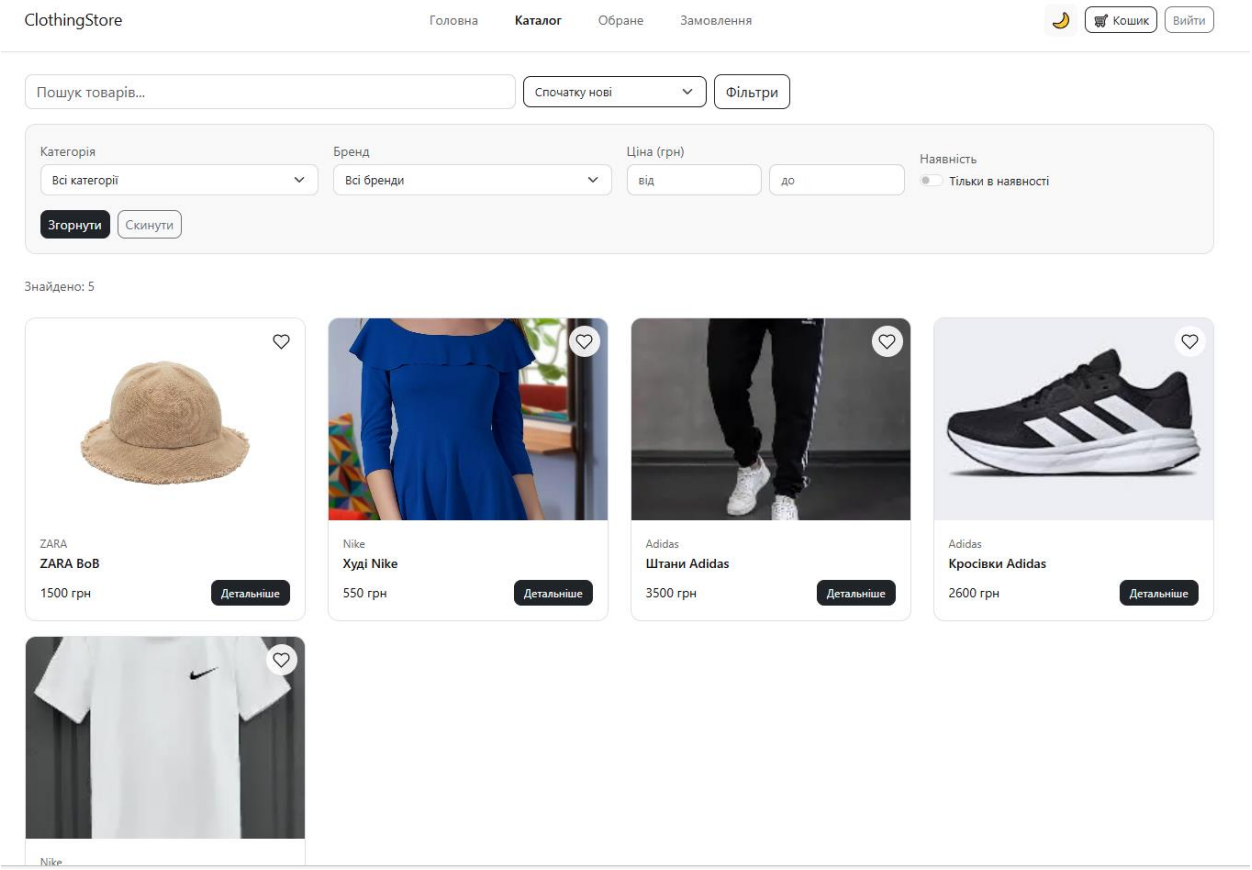

Рисунок 4.4 – Каталог з розгорнутою панеллю фільтрів

Клік на картку товару веде на сторінку деталей, де відображаються зображення, опис, перелік доступних варіантів (розмір/колір), середня оцінка та відгуки покупців. Кнопки «Додати в кошик» та «В обране» доступні авторизованим користувачам (Рисунок 4.5).

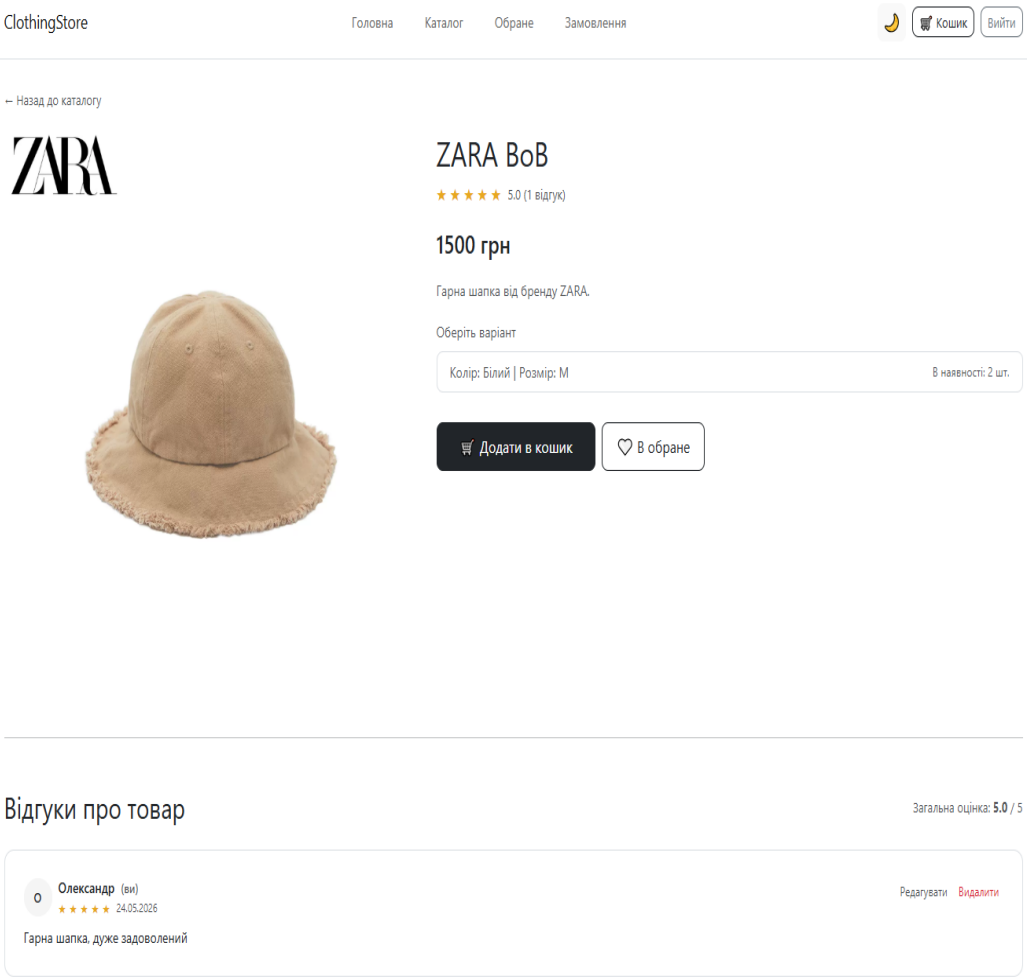

Рисунок 4.5 – Сторінка деталей товару зі секцією відгуків

У кошику для кожної позиції можна змінити кількість, обрати варіант або видалити запис. Кнопка «Оформити замовлення» переходить на сторінку чекауту, де користувач вводить ПІБ, телефон, спосіб доставки та адресу. Після підтвердження виконується переадресація на форму оплати Stripe Checkout. Після успішної оплати користувач повертається на сторінку «Дякуємо за покупку» (Рисунок 4.6–4.7).

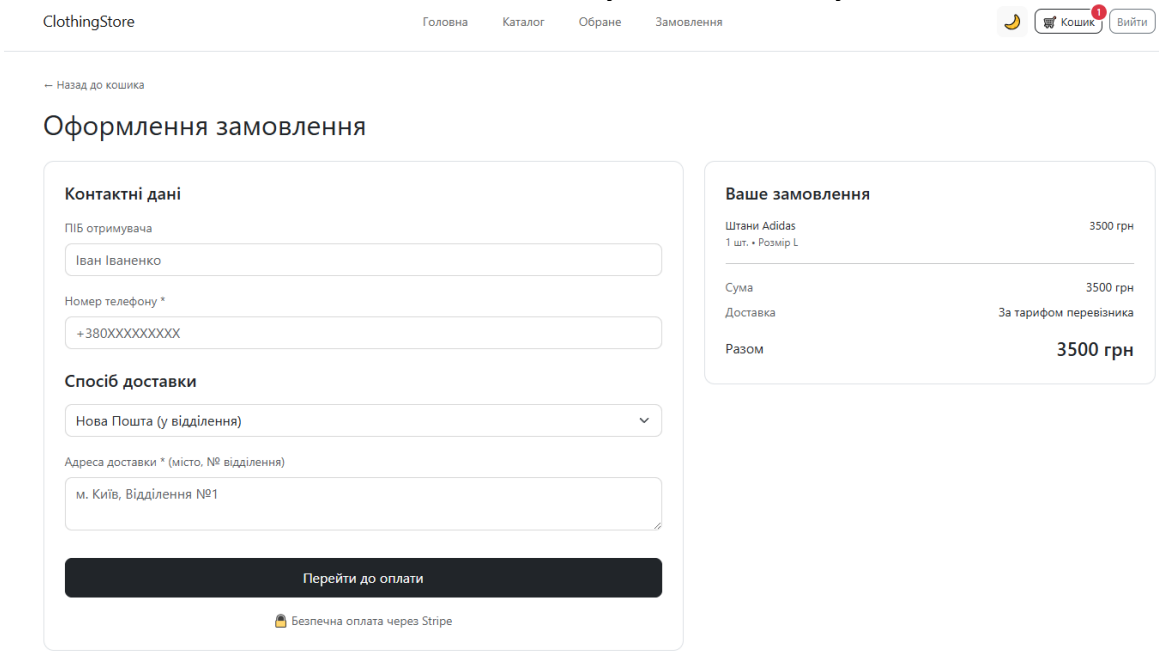

Рисунок 4.6 – Сторінка чекауту

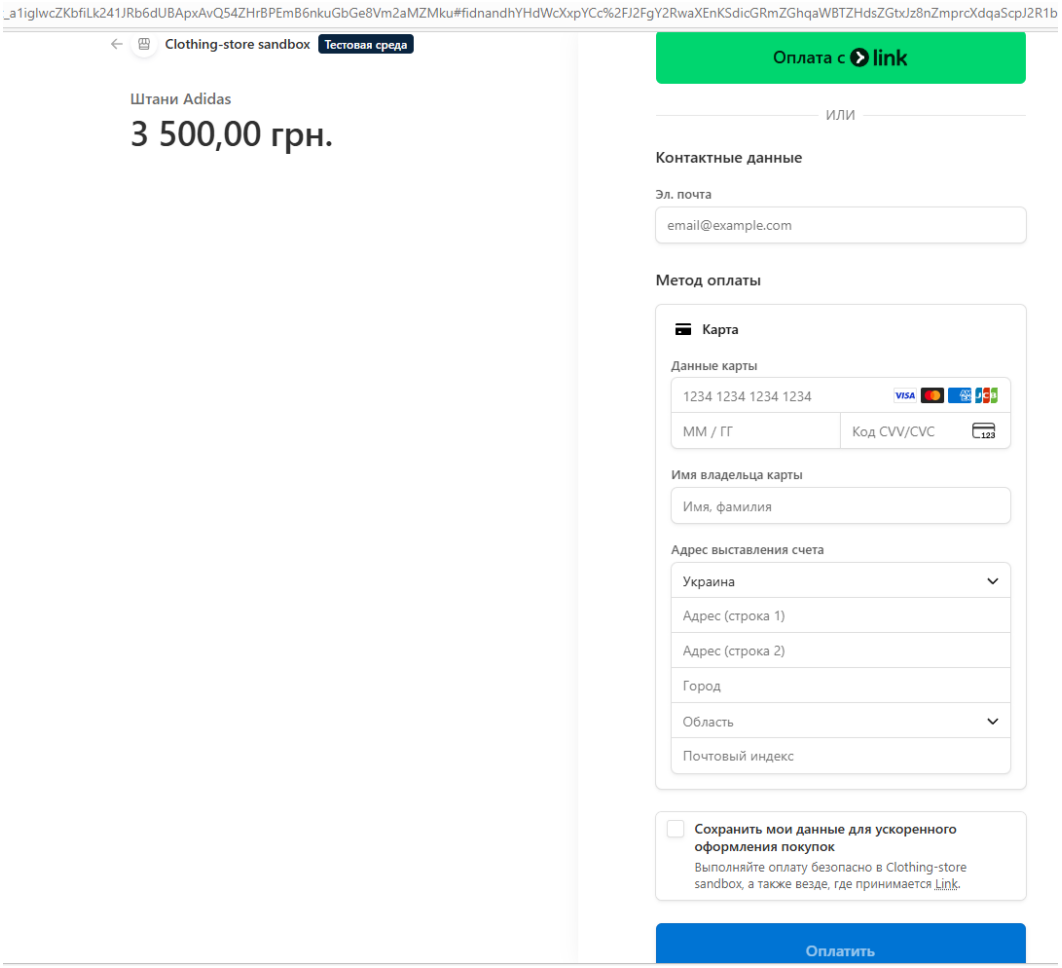

Рисунок 4.7 – Підтвердження оплати

Розділ «Замовлення» відображає історію замовлень із кольоровими індикаторами статусів. Розгортання картки замовлення показує склад і дані доставки. Для замовлень зі статусами PAID, SHIPPED, DELIVERED доступна кнопка «Запросити повернення», що відкриває модальне вікно для опису причини. Після подання запит відображається у тій же картці з поточним статусом (Рисунок 4.8).

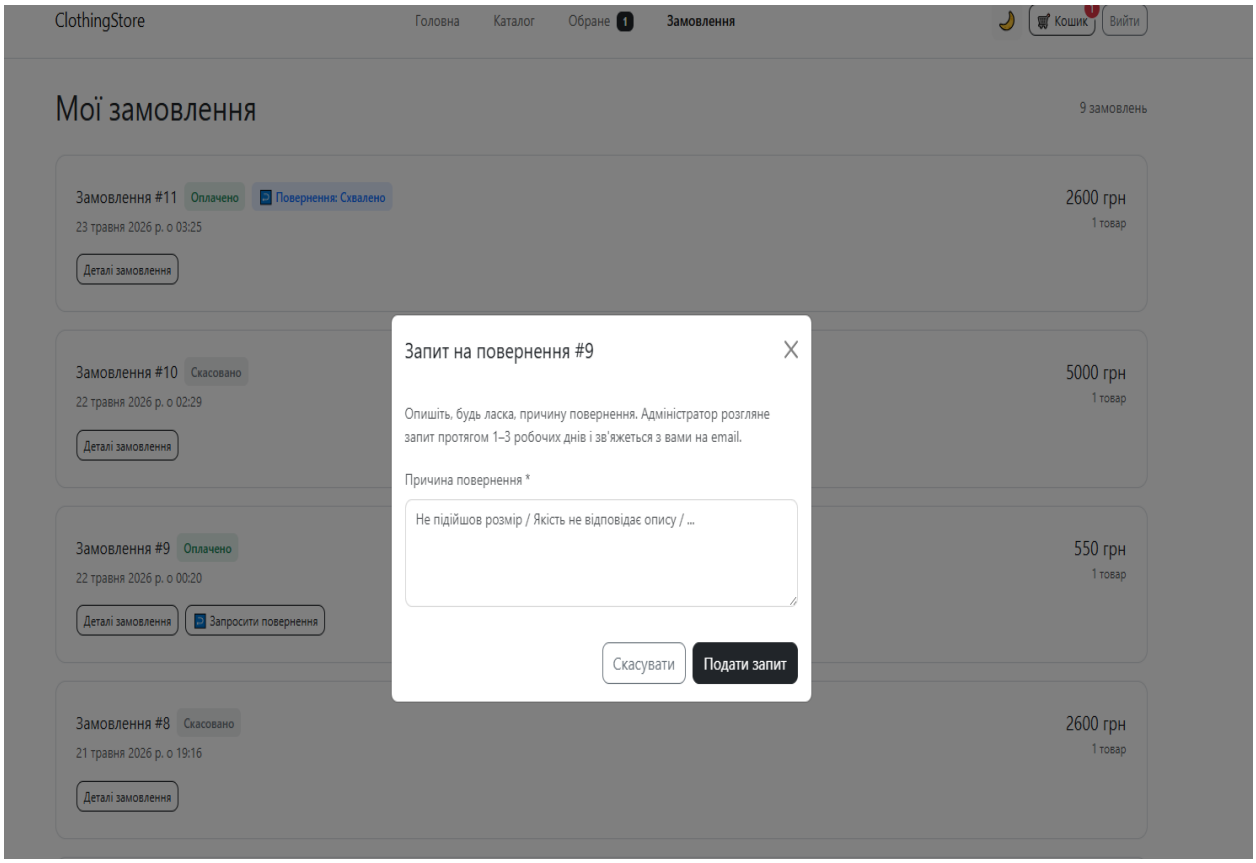

Рисунок 4.8 – Історія замовлень із запитом на повернення

На сторінці входу натиснути «Забули?». Ввести email – у разі існування акаунта на пошту надсилається посилання з токеном, дійсним 1 годину (Рисунок 4.9–4.10). Перехід за посиланням відкриває форму введення нового пароля.

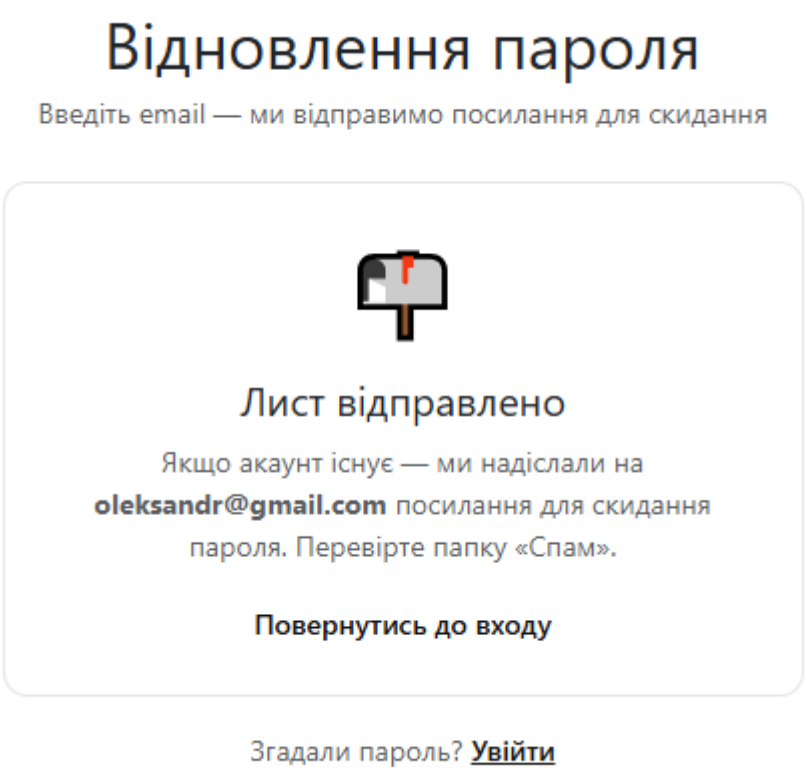

Рисунок 4.9 – Запит на скидання пароля

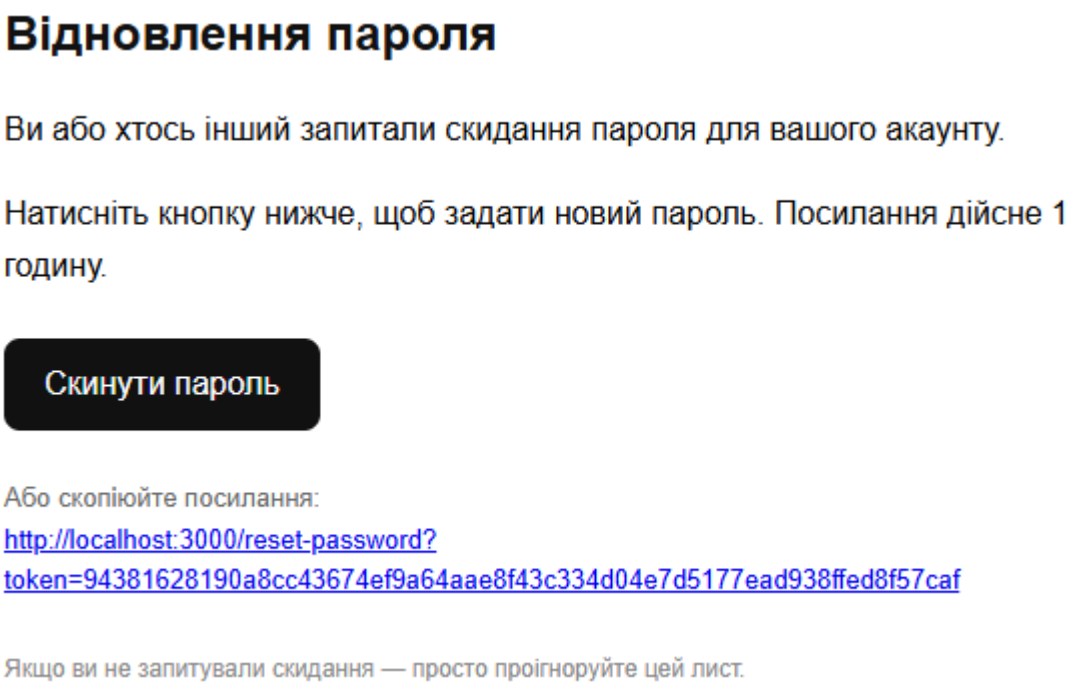

Рисунок 4.10 – Лист на пошту

Кнопка зі значком місяця (або сонця) у верхньому правому куті перемикає світлу/темну тему. Вибір зберігається у localStorage і застосовується при наступних візитах (Рисунок 4.11).

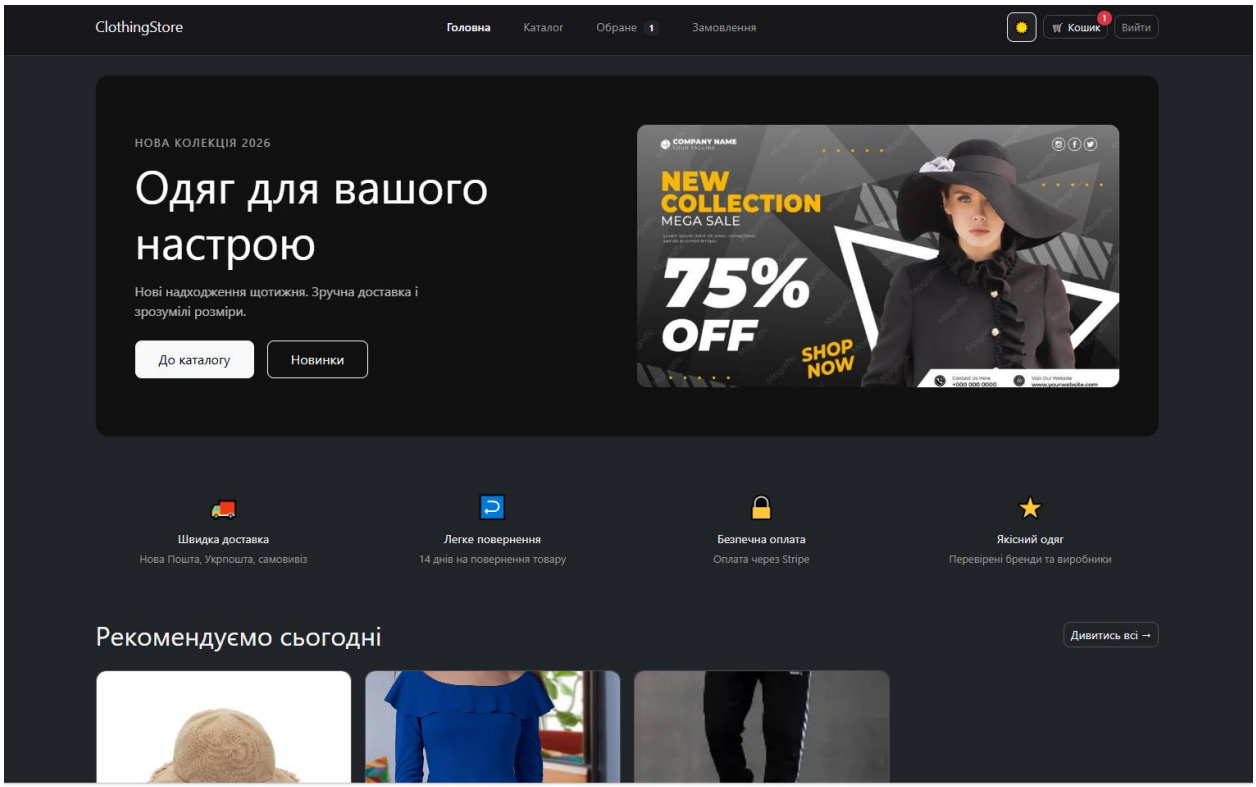

Рисунок 4.11 – Інтерфейс у темній темі

Користувачі з роллю ADMIN мають доступ до сторінки «Адмін панель» (Рисунок 4.12) з трьома вкладками:

– товари – створення категорій, брендів, товарів (з варіантами та зображенням), редагування і видалення товарів;

– відгуки – перегляд, редагування та видалення відгуків будь-якого користувача;

– повернення – перегляд усіх запитів, зміна статусу з коментарем (зміна статусу автоматично надсилає лист клієнту).

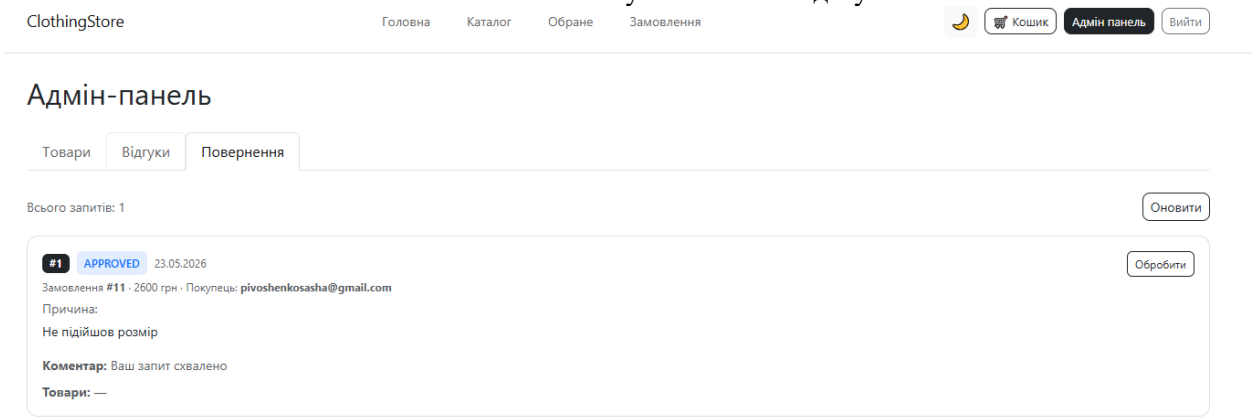

Рисунок 4.12 – Адміністративна панель (вкладка Повернення)

Наведені екранні форми демонструють, що розроблений інтерфейс послідовно супроводжує користувача на кожному кроці – від перегляжу каталогу неавторищованим відвідувачем до підтвердження повернення адміністратором. Усі три ролі (гість, зареєстрований покупець та адміністратор) працюють у межах єдиного візуального стилю, побудованого на компонентах React-Bootstrap, завдяки чому навігація лишається передбачуваною незалежно від рівня доступу. Адміністративна панель об’єднує керування товарами, замовленнями та поверненнями у трьох вкладках з однотипною логікою роботи, що скорочує час на опанування системи та мінімізує ймовірність помилкових дій оператора. Описані сценарії повністю охоплюють типовий життєвий цикл замовлення, підтверджуючи практичну придатність створеного програмного забезпечення до експлуатації.

Висновки до розділу 4

У розділі описано виконану роботу з кодування, тестування та документування програмного забезпечення «ClothingStore». Наведено специфікацію застосунку, перелік сутностей предметної області з типами даних, опис розроблених класів MobX-сховищ та контролерів серверної частини. Представлено лістинги ключових компонентів – контролера відгуків, поштового сервісу, фільтрації товарів та сховища теми. Розроблено 24 тестових сценарії, які підтвердили відповідність застосунку функціональним

вимогам; виявлені на етапі розробки помилки задокументовано та виправлено. Сформовано керівництво користувача з покроковим описом виконання основних операцій. Кількісні показники, отримані під час випробувань, підтверджують ефективність прийнятих інженерних рішень: середній час відповіді сервера на операції читання не перевищує 35 мс, а повний цикл оформлення замовлення разом зі зверненням до платіжного шлюзу Stripe укладається в 420 мс. Успішне проходження всіх 24 тестових сценаріїв та усунення виявлених дефектів свідчать про функціональну стабільність системи й готовність її основних модулів до дослідної експлуатації.

ВИСНОВКИ

У кваліфікаційній бакалаврській роботі розв’язано актуальне науково-практичне завдання – розроблено повнофункціональний вебзастосунок інтернет-магазину одягу “ClothingStore”, що автоматизує основні бізнес-процеси електронної комерції у сегменті Fashion e-commerce та забезпечує повний цикл взаємодії з клієнтом від ознайомлення з каталогом до отримання товару. У процесі виконання роботи досягнуто всіх завдань, поставлених у вступі. Зокрема:

1) Виконано порівняльний аналіз існуючих рішень. Досліджено платформи світових лідерів галузі – Nike, Adidas та ASOS. Встановлено, що галузевими стандартами є мікросервісна архітектура, фасетна фільтрація та глибока персоналізація. Водночас з’ясовано, що готові платформи є надлишковими за функціональність та коштовними у підтримці малого й середнього бізнесу, що обґрунтовує доцільність розробки власного гнучкого рішення.

2) Спроєктовано трирівневу клієнт-серверну архітектуру з чітким розмежуванням рівнів подання (React SPA), бізнес-логіки (Node.js? Express REST API) та збереження даних (Microsoft SQL Server). Розроблено комплект UML-моделей – діагарми прецедентів, класів, послідовності, станів, діяльності, компонентів та розгортання, які формалізували поведінку системи та структуру даних.

3) Розроблено реляційну структуру бази даних з тринадцяти сутностей з дотриманням третьої нормальної форми. Відокремлення сутностей Product та ProductVariant дозволило гнучко моделювати атрибути одягу (розмір, колір, кількість на складі), а зберігання історичної ціни у OrderItem забезпечило імутабельність фінансових записів незалежно від подальших змін у каталозі.

4) Реалізовано увесь функціонал, передбачений завданням: перегляд каталогу з потужною фільтрацією за категорією, брендом, ціновим діапазоном

та наявністю на складі, шість варіантів сортування, пошук за назвою, повноцінне управління кошиком та обраним, оформлення замовлень з онлайн-оплатою, перегляд історії замовлень з можливістю скасування та ініціації запиту на повернення, залишення відгуків з оцінкою товарів.

5) Реалізовано REST API із 32 ендпоінтів, які забезпечують повний обмін даними між клієнтом і сервером. Доступ до захищених маршрутів регулюється проміжним middleware authMiddleware на основі JWT-токенів; адміністративні маршрути додатково обмежені middleware рольової перевірки checkRoleMiddleware('ADMIN'), що реалізує дворівневу модель доступу USER / ADMIN.

6) Реалізовано додатковий функціонал, що виходить за межі базових вимог:

- систему email-сповіщень (вітальний лист, підтвердження замовлення, скидання пароля, оновлення статусу повернення) на базі nodemailer;
- механізм відновлення пароля з токеном обмеженого терміну дії;
- повний життєвий цикл повернень товарів з модерацією адміністратором; перемикання світлої та темної теми зі збереженням вибору користувача у localStorage.

7) Інтегровано платіжний шлюз Stripe, що дозволило відповідати міжнародному стандарту PCI DSS без зберігання чутливих платіжних даних на сервері магазину. Обробка статусу платежу реалізована через асинхронний механізм webhook-подій.

8) Проведено комплексне тестування системи на трьох рівнях – модульне, інтеграційне тестування API через Postman та ручне функціональне тестування інтерфейсу у браузерах Chrome і Mozilla Firefox. Усі 24 розроблені тестові сценарії виконані успішно. На етапі розробки виявлено та усунуто 5 помилок.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Глибовець М. М. Архітектура програмних систем : підручник. Київ : НаУКМА, 2021. 260 с.

2. Пасічник В. В., Жежнич П. І., Кравець Р. Б. Глобальні інформаційні системи та технології (моделі, проєктування, розробка) : монографія. Львів : Видавництво Львівської політехніки, 2022. 350 с.

3. Терейковський І. А., Терейковська Л. О. Безпека веб-орієнтованих інформаційних систем : навчальний посібник. Київ : КПІ ім. Ігоря Сікорського, 2020. 184 с.

4. Adidas. Official website of Adidas AG. URL: https://www.adidas.com/ (дата звернення: 28.04.2026).

5. ASOS. Official website of ASOS Plc. URL: https://www.asos.com/ (дата звернення: 28.04.2026).

6. Bootstrap Documentation v5.3. The most popular HTML, CSS, and JS library in the world. URL: https://getbootstrap.com/docs/5.3/getting-started/introduction/ (дата звернення: 28.04.2026).

7. Express.js Documentation. Fast, unopinionated, minimalist web framework for Node.js. URL: https://expressjs.com/en/4x/api.html (дата звернення: 28.04.2026).

8. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : Doctoral Dissertation. Irvine : University of California, 2000. 162 p. URL: https://ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf (дата звернення: 28.04.2026).

9. ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. 2011. 34 p.

10. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519. Internet Engineering Task Force (IETF), 2015. 30 p. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 28.04.2026).

11. Microsoft SQL Server Documentation. SQL Server Technical Documentation. URL: https://learn.microsoft.com/en-us/sql/sql-server/ (дата звернення: 28.04.2026).

12. MobX Documentation. Simple, scalable state management. URL: https://mobx.js.org/README.html (дата звернення: 28.04.2026).

13. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol, CA : O'Reilly Media, 2021. 612 p.

14. Nielsen J. 10 Usability Heuristics for User Interface Design. Nielsen Norman Group. 2024. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (дата звернення: 28.04.2026).

15. Nike. Official website of Nike, Inc. URL: https://www.nike.com/ (дата звернення: 28.04.2026).

16. Node.js Documentation. Node.js v20.x runtime. URL: https://nodejs.org/dist/latest-v20.x/docs/api/ (дата звернення: 28.04.2026).

17. OWASP Foundation. OWASP Top 10:2021 – The Ten Most Critical Web Application Security Risks. 2021. URL: https://owasp.org/Top10/ (дата звернення: 28.04.2026).

18. PCI Security Standards Council. Payment Card Industry Data Security Standard (PCI DSS) Requirements and Testing Procedures. Version 4.0. 2022. 360 p. URL: https://www.pcisecuritystandards.org/document_library/ (дата звернення: 28.04.2026).

19. Provos N., Mazières D. A Future-Adaptable Password Scheme. Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference. Monterey, California, USA, 1999. P. 81–91.

20. React Documentation. Beta. React: A JavaScript library for building user interfaces. URL: https://react.dev/learn (дата звернення: 28.04.2026).

21. Saputro M. I., Rohman A. N. Implementation of the Apriori Algorithm for Clothing Store Product Recommendations based on Sales Transaction History. Sistemasi: Jurnal Sistem Informasi. 2026. Vol. 15, No. 1. P. 88–95. DOI: https://doi.org/10.32520/stmsi.v15i1.5648

22. Sequelize ORM Documentation. Sequelize: Feature-rich ORM for Modern Node.js. URL: https://sequelize.org/docs/v6/ (дата звернення: 28.04.2026).

23. Stripe API Reference. Online payment processing for internet businesses. URL: https://stripe.com/docs/api (дата звернення: 28.04.2026).

24. Ulfiana R. F., Satria M. N. D. Design and Implementation of a Responsive E-Commerce Web Application for Teras Thrifting Store. Sistemasi: Jurnal Sistem Informasi. 2025. Vol. 14, No. 4. P. 1640–1653. DOI: https://doi.org/10.32520/stmsi.v14i4.5242