МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних
інформаційних систем
_____Юрій КОНДРАТЕНКО
«____»_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ВЕБЗАСТОСУНОК З ПРОДАЖУ ЕЛЕКТРОНІКИ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____Андрій ПЕРЕЛИГА
«___»_____ 2025 р.

Керівник д-р техн. наук, доцент _____ Ірина КАЛІНІНА
«___»_____ 2025 р.

Миколаїв – 2025

Чорноморський національний університет імені Петра Могили

(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп’ютерні науки
Освітня програма	Комп’ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем
_____ Юрій КОНДРАТЕНКО
«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Перелиги Андрія Олександровича
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Вебзастосунок з продажу електроніки.

Керівник роботи: Калініна Ірина Олександрівна, д-р техн. наук, доцент.
(прізвище, ім’я, по батькові, посада, науковий ступінь, вчене звання)

Затверджена наказом ЧНУ ім. Петра Могили від «18» листопада 2024 р. № 321.
2. Строк представлення кваліфікаційної роботи «___» _____ 2025 р.
3. Початкові дані до роботи: визначення вимог до функціоналу, інтерфейсу користувача та безпеки вебзастосунку, призначеного для продажу електроніки; аналіз очікувань і потреб цільової аудиторії онлайн-магазинів електроніки; вибір відповідного технологічного стека для розробки вебзастосунку.
Очікуваний результат: вебзастосунок електронної комерції для продажу електроніки, який орієнтований на зручність користувача, забезпечує надійний захист даних і транзакцій, а також передбачає можливість масштабування для подальшого розвитку проєкту.

4. Перелік питань, що підлягають розробці:

– дослідження поточного стану ринку електронної комерції;

– аналіз наявних технологій і методологій, що застосовуються для створення вебзастосунків;

– проєктування та реалізація вебзастосунку з продажу електроніки;

– тестування створеного вебзастосунку;

– оцінювання ефективності роботи вебзастосунку та визначення можливостей його подальшого оновлення та вдосконалення.

5. Перелік графічних матеріалів: презентація.

Керівник роботи _____ Ірина КАЛІНІНА

(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Андрій ПЕРЕЛИГА

(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «20» грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Вебзастосунок з продажу електроніки

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	20.12.2024	20.12.2024	
2	Аналіз предметної області та постановка задачі	21.12.2024	30.01.2025	
3	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема аналіз ринку, визначення основних потреб користувачів та розробка вимог	31.01.2025	01.03.2025	
4	Створення архітектури, проєктування та дизайн вебзастосунку	02.03.2025	01.04.2025	
5	Розробка клієнтської та серверної частин, конфігурація бази даних	02.04.2025	15.05.2025	
6	Перший попередній захист КР на засіданні комісії кафедри	16.05.2025	16.05.2025	
7	Корегування роботи за результатами попереднього захисту	17.05.2025	29.05.2025	
8	Другий попередній захист КР на засіданні комісії кафедри	30.05.2025	30.05.2025	
9	Доробка та остаточне оформлення КР	31.05.2025	10.06.2025	
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	12.06.2025	12.06.2025	

Керівник роботи _____ Ірина КАЛІНІНА

(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Андрій ПЕРЕЛИГА

(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«30» січня 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувача групи 402 ЧНУ ім. Петра Могили

Перелиги Андрія Олександровича

на тему: **«ВЕБЗАСТОСУНОК З ПРОДАЖУ ЕЛЕКТРОНІКИ»**

Актуальність даної роботи зумовлена стрімким розвитком електронної комерції, що особливо проявляється у сфері продажу електроніки та комп'ютерної техніки. Попит на таку продукцію невпинно зростає, а класичні роздрібні магазини дедалі активніше переходять у цифровий простір. Це дає змогу охоплювати ширшу аудиторію, підвищувати якість обслуговування та ефективніше організовувати бізнес-процеси.

Об'єктом роботи є процеси створення вебзастосунку для продажу електроніки із застосуванням сучасних вебтехнологій.

Предметом роботи є методи та технології для розробки програмних засобів вебзастосунків у сфері електронної комерції.

Метою роботи є оптимізація процесів реалізації електронної техніки шляхом розробки спеціалізованого вебзастосунку.

В результаті виконання роботи було проаналізовано особливості предметної області електронної комерції та сучасні підходи до розробки вебзастосунків, визначено основні компоненти системи й технологічний стек, а також на основі цього розроблено вебзастосунок.

Дана робота складається з чотирьох розділів. У першому розділі проведено аналіз предметної області та огляд аналогів на ринку електронної комерції. Другий розділ присвячений огляду технологій та програмних засобів, використаних у розробці вебзастосунку. У третьому розділі наведено опис процесу розробки вебзастосунку. В четвертому – аналіз отриманих результатів, тестування та інформаційні діаграми. Загальний обсяг роботи – 92 сторінки. Кваліфікаційна робота містить 1 додаток, 66 рисунків, 4 таблиці і 30 джерел посилання.

Ключові слова: *вебзастосунок, електронна комерція, продаж електроніки*.

ABSTRACT

to the qualification work by the student of the group 402 of Petro Mohyla Black Sea National University

Perelyha Andrii

«WEB APPLICATION FOR ELECTRONICS SALES»

The relevance of this work is due to the rapid development of e-commerce, which is particularly evident in the sale of electronics and computer equipment. Demand for such products is growing steadily, and traditional retail stores are increasingly moving into the digital space. This makes it possible to reach a wider audience, improve the quality of service, and organize business processes more efficiently.

The object of this work is the process of creating a web application for selling electronics using modern web technologies.

The subject of this work is methods and technologies for developing software for web applications in the field of e-commerce.

The purpose of this work is to optimize the processes of electronic equipment sales by developing a specialized web application.

As a result of the work, the features of the subject area of e-commerce and modern approaches to the development of web applications were analyzed, the main components of the system and the technology stack were determined, and a web application was developed based on this.

This work consists of four sections. The first section analyzes the subject area and reviews analogues in the e-commerce market. The second section is devoted to reviewing the technologies and software used in the development of the web application. The third section describes the process of developing the web application. The fourth section analyzes the results obtained, testing, and information diagrams.

The overall scope of the work is 92 pages. Thesis contains 1 application, 66 figures, 4 tables, and 30 references in it.

Keywords: *web application, e-commerce, electronics sales*.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API – Application Programming Interface

CSS – Cascading Style Sheets

DOM – Document Object Model

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

JSON – JavaScript Object Notation

UI – User Interface

URL – Uniform Resource Locator

VS – Visual Studio

ВСТУП

В сучасному світі спостерігається стрімкий розвиток електронної комерції, що особливо помітно у сфері продажу електроніки. Згідно з аналітичними даними, попит на електроніку постійно зростає, а традиційні магазини все частіше переходять в онлайн-формат, що дає змогу охоплювати ширшу аудиторію, підвищувати якість обслуговування та ефективніше організовувати бізнес-процеси. Однак, незважаючи на широку популярність сучасних вебзастосунків, орієнтованих на продаж електроніки, користувачі все ще стикаються з низкою проблем: незручний інтерфейс, недостатня оптимізація, низька швидкодія тощо.

Саме тому розробка сучасного вебзастосунку з продажу електроніки є актуальним завданням, яке дозволить автоматизувати процес вибору й покупки товарів, спростити взаємодію між продавцем і покупцем, а також забезпечити зручний та безпечний доступ до широкого асортименту техніки й аксесуарів. Під час виконання завдання було здійснено аналіз сучасних тенденцій електронної комерції, розглянуто архітектурні рішення та технології, що використовуються для створення інтернет-магазинів електроніки.

Метою роботи є оптимізація процесів реалізації електронної техніки шляхом розробки спеціалізованого вебзастосунку.

Необхідно виконати наступні завдання:

– дослідження поточного стану ринку електронної комерції;

– аналіз наявних технологій і методологій, що застосовуються для створення вебзастосунків;

– проєктування та реалізація вебзастосунку з продажу електроніки;

– тестування та оцінка ефективності роботи вебзастосунку з визначенням можливостей його подальшого оновлення та вдосконалення.

Об'єктом роботи є процеси створення вебзастосунку з продажу електроніки із застосуванням сучасних вебтехнологій.

Предметом роботи є методи та технології для розробки програмних засобів вебзастосунків у сфері електронної комерції.

1 АНАЛІЗ ЕЛЕКТРОННОЇ КОМЕРЦІЇ ТА ПОСТАНОВКА ЗАДАЧІ ДЛЯ РОЗРОБКИ ВЕБЗАСТОСУНКУ З ПРОДАЖУ ЕЛЕКТРОНІКИ

1.1 Опис сфери електронної комерції та програмного забезпечення для продажу електроніки

Вебзастосунок для продажу електроніки - це сучасний інструмент, що забезпечує користувачам зручний онлайн-доступ до різноманітних товарів, пов’язаних із комп’ютерною технікою та електронікою. Такі системи створюються для спрощення процесу вибору, замовлення та придбання пристроїв.

Електронна комерція у сфері комп’ютерної техніки охоплює не лише продаж фізичних товарів, а й надання супутніх послуг. Основні етапи взаємодії користувача з вебзастосунком включають пошук і порівняння товарів, оформлення замовлення, оплату, обробку покупки, організацію доставки та можливість повернення продукції.

Архітектура таких вебзастосунків зазвичай складається з двох основних частин:

– frontend - це інтерфейс, з яким безпосередньо працює користувач. Він відповідає за відображення інформації, зручну навігацію та взаємодію з функціоналом сайту;

– backend - це серверна частина, яка обробляє дані, керує логікою додатку, забезпечує зв’язок із базою даних і відповідає за безпеку та стабільність роботи системи.

Для розробки вебзастосунків у цій сфері використовують різні підходи:

– монолітна архітектура, в якій всі компоненти системи тісно пов’язані між собою [1];

– мікросервісний підхід, в якому розділено функціонал на незалежні сервіси [2];

– SPA (Single Page Application): додатки, що працюють без перезавантаження сторінок, забезпечуючи швидкий і плавний досвід користувача [3];

– JAMstack - архітектура, що базується на використанні клієнтського JavaScript, API та статичного контенту;

– MERN-стек: сучасний набір технологій на основі JavaScript (MongoDB, Express.js, React.js, Node.js), який дозволяє створювати масштабовані та функціональні вебзастосунки для онлайн-продажу електроніки.

Вибір конкретної технології та архітектури залежить від цілей проекту, вимог до продуктивності, безпеки, масштабованості та швидкості розробки. Важливо враховувати й особливості ринку електроніки, де конкуренція висока, а користувачі очікують швидкої роботи, інтуїтивного інтерфейсу та широкого асортименту продукції.

1.2 Огляд та аналіз наявних вебзастосунків з продажу електроніки

Сучасний ринок електронної комерції пропонує широкий вибір вебзастосунків, які спеціалізуються на продажу комп’ютерної техніки, комплектуючих та електроніки [4]. Вони відрізняються як за масштабом (від великих мультибрендових платформ до нішевих магазинів), так і за функціональністю, дизайном, рівнем сервісу та інноваційними можливостями.

За останні роки український сегмент демонструє динамічний розвиток, що пов’язано як із глобальними тенденціями цифровізації, так і з внутрішніми викликами, зокрема змінами у споживчих звичках та обмеженнями офлайн торгівлі. Значний поштовх до зростання ринку електронної комерції надала пандемія COVID-19, коли обсяги онлайн-продажів суттєво збільшилися. Попри складні економічні умови, онлайн-торгівля в Україні залишається перспективною галуззю з великим потенціалом для подальшого розвитку.

На рис. 1.1 наведено лідерів електронної комерції в Україні [5].

Рисунок 1.1 – Лідери електронної комерції в Україні

У якості прикладу вже існуючих вебзастосунків з продажу електроніки можна виділити наступні:

– Rozetka;
– Allo;
– Comfy.

Інтернет-магазин «Rozetka» - найбільший онлайн-ритейлер електроніки в Україні, заснований у 2005 році [6]. Спочатку компанія зосереджувалась саме на електроніці, згодом суттєво розширивши асортимент товарів. Платформа працює за моделлю маркетплейсу, що дозволяє залучати до співпраці численних постачальників та брендів. Rozetka активно інвестує в розвиток власної логістики, що забезпечує швидку доставку по всій країні. Окрему увагу приділено автоматизації процесів для продавців: інтеграція з CRM-системами дозволяє оперативно оновлювати асортимент, ціни та залишки, а також автоматизувати обробку замовлень. Платформа встановлює чіткі вимоги до якості контенту, фото та описів, що сприяє підвищенню довіри покупців і покращенню користувацького досвіду.

На рис. 1.2 наведено Інтернет-магазин «Rozetka».

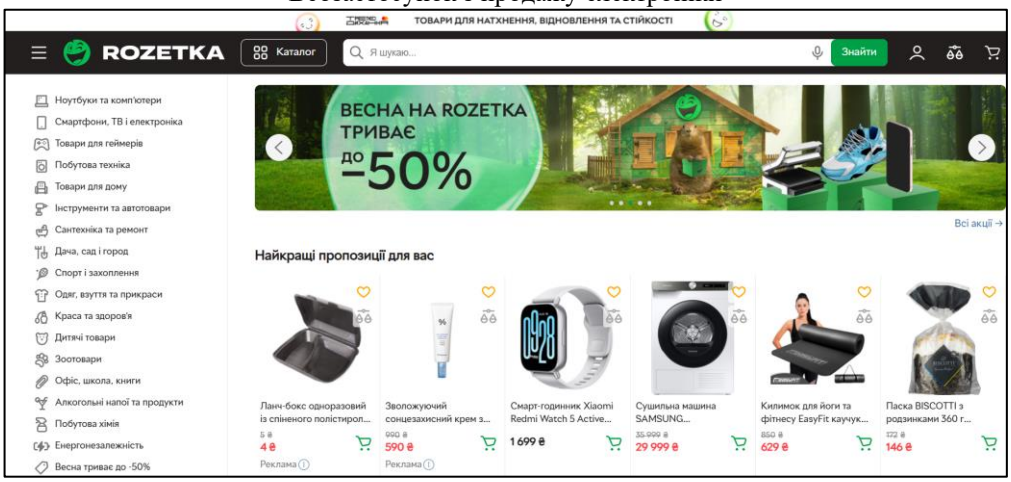

Рисунок 1.2 – Інтернет-магазин «Rozetka»

Інтернет-магазин «Allo» - один із найбільших українських маркетплейсів у сфері електроніки, що поєднує онлайн та класичний роздріб (див. рис. 1.3) [7]. Компанія активно розвиває власну мережу магазинів і пунктів самовивозу, а також впроваджує інноваційні сервіси для підвищення зручності покупців. Зокрема, у 2024–2025 роках Allo оновила мобільний застосунок, розширила функціонал каталогу, сторінок товарів і процесу оформлення замовлень. Платформа також впровадила нові формати роботи з клієнтами, такі як відеоконсультації в реальному часі та розширення можливостей самостійного отримання товарів у магазинах. Allo приділяє значну увагу соціальній відповідальності, підтримуючи благодійні ініціативи та впроваджуючи спеціальні категорії товарів для людей з особливими потребами. Компанія постійно адаптує бізнес-модель для підвищення ефективності та впровадження сучасних рішень у сфері електронної комерції.

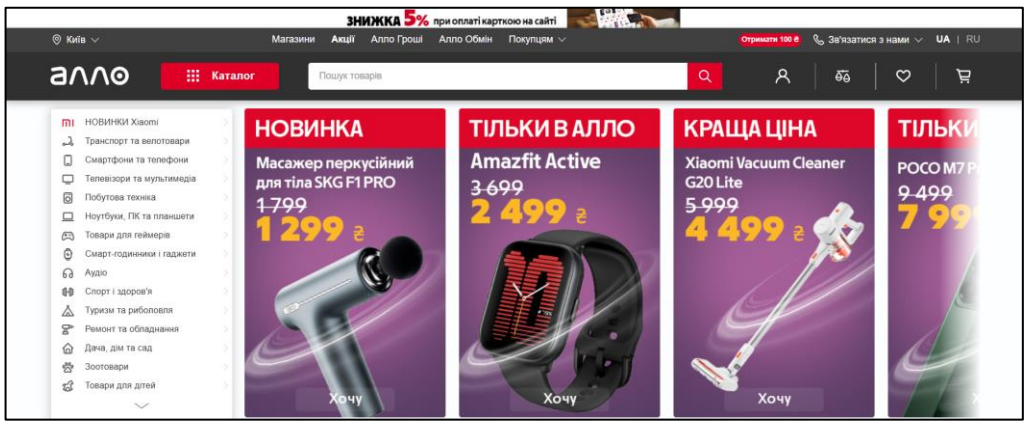

Рисунок 1.3 – Інтернет-магазин «Allo»

Інтернет-магазин «Comfy» - один із провідних українських онлайн-магазинів електроніки, що працює переважно на національному ринку (див. рис. 1.4) [8]. Платформа поєднує онлайн-продажі з розгалуженою мережею фізичних магазинів, що дозволяє клієнтам обирати між доставкою та самовивозом. Асортимент Comfy охоплює різні категорії електроніки, а також товари для дому та офісу. Компанія інвестує у власну логістичну інфраструктуру, що позитивно впливає на швидкість виконання замовлень. Окремо варто відзначити акцент на підвищенні якості сервісу: Comfy впроваджує програми післяпродажної підтримки, розвиває контакт-центр для оперативного вирішення питань клієнтів та забезпечує гнучкі умови повернення товарів. Платформа також постійно аналізує споживчу поведінку.

Рисунок 1.4 – Інтернет-магазин «Comfy»

Порівняння вищеописаних вебзастосунків буде подано у вигляді табл. 1.1.

Для структурного порівняння треба використати наступні критерії:

– фреймворки;
– платформа, на якій розроблена система;
– PWA;
– UI/UX;
– оптимізація;
– сторонні сервіси;
– мобільний додаток;
– рівень кастомізації.

Таблиця 1.1 – Порівняльна характеристика наведених вебзастосунків з продажу електроніки

Критерій	Rozetka	Allo	Comfy
Фреймворки/ Мови	ASP.NET Core, Angular, TypeScript. Мікросервісна архітектура.	Symfony, Laravel, Magento,JavaScript/ jQuery. Мікросервісна архітектура.	PHP, Magento, JavaScript. Частково монолітна структура.
CMS/ Платформа	Власна e-commerce платформа, гнучко масштабується.	Кастомні розширення Magento.	Частково кастомізована Magento.
PWA	Мобільна версія працює як PWA, кешування, офлайн-режим.	Частково – деякі PWA-принципи реалізовано, але не повноцінно.	Звичайна адаптивна мобільна версія.
UI/UX	Сучасний, інтуїтивний інтерфейс, розумний пошук, фільтри, рейтинг.	Класичний інтерфейс Інтернет-магазину, фокус на мобільних користувачах.	Простий та зручний інтерфейс, підтримка чат-ботів.
Оптимізація	CDN, кешування на фронті та бекенді, lazy-loading, PWA.	Кешування Magento, використання CDN.	Magento та JS-оптимізації. lazy-loading зображень.
Сторонні сервіси	Apple Pay, Google Pay, Nova Poshta API, CRM (Creatio),аналітика, єПідтримка.	Оплата частинами, CRM, служби доставки, email/SMS-розсилки.	OroCRM, чат-боти, Nova Poshta API, подарункові сертифікати, інтеграція з BigQuery.
Мобільний додаток	Повнофункціональний, синхронізований з веб.	Адаптація Android та iOS.	Обмежений функціонально в порівнянні з веб-версією.
Рівень кастомізації	Розроблено власне ядро та API для зовнішніх інтеграцій.	Використовується Magento, але є кастомні модулі.	Переважно стандартний Magento з розширеннями.

Rozetka демонструє високий рівень технологічної зрілості завдяки власній платформі, PWA-архітектурі та мікросервісному підходу. Це дозволяє їй швидко масштабуватися та адаптуватися до змін.

Allo поєднує гнучкість Magento з кастомним розвитком, що забезпечує стабільну роботу та розширюваність функціоналу.

Comfy, хоча й частково базується на готовому рішенні Magento, робить акцент на простоту використання та інтеграції з каналами комунікації, такими як чат-боти.

Отримані результати аналізу можуть бути використані для розробки власного вебзастосунку, який відповідатиме сучасним вимогам ринку, сприятиме підвищенню конкурентоспроможності та забезпечить позитивний користувацький досвід. Важливо враховувати не лише технічні рішення, але й загальну архітектуру системи, її масштабованість, продуктивність та орієнтацію на клієнта.

1.3 Опис архітектури для побудови вебзастосунку

Щоб реалізувати програмну частину, слід визначити архітектурний підхід, який використовуватиметься при створенні застосунку. Для проєктування такої системи доцільно обрати між монолітною та мікросервісною архітектурою.

На рис. 1.5 наведено графічне порівняння монолітної та мікросервісних архітектур.

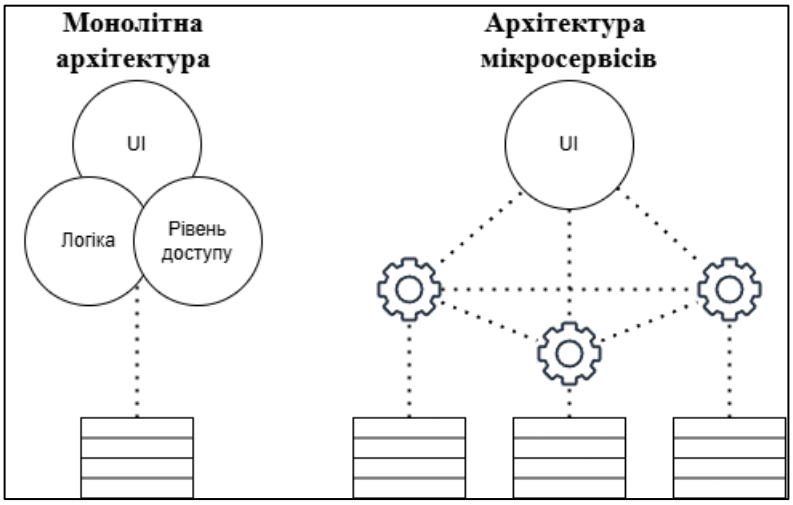

Рисунок 1.5 – Графічне порівняння монолітної та мікросервісних архітектур

Монолітна архітектура - це підхід до розробки програмного забезпечення, за якого застосунок створюється як єдине ціле. Прикладом такої архітектури може бути програма, яка складається з трьох основних компонентів: UI (користувацький інтерфейс) на клієнтській стороні (включає HTML та JavaScript, що працюють у браузері на пристрої користувача), бази даних та серверного застосунку. Серверна частина обробляє HTTP-запити від користувача, реалізує доменну логіку та генерує HTML-контент для відправки в браузер.

Монолітну архітектуру зазвичай використовують у системах з обмеженим функціоналом, де вся обробка запитів може виконуватися в одному процесі. Це дає змогу використовувати повний функціонал мов програмування для організації коду в структури, функції, класи, простори імен, інтерфейси тощо.

Перевагами монолітної архітектури є її простота в реалізації та тестуванні. Для розробки не потрібно глибоких спеціалізованих знань - достатньо базових навичок створення структури коду. Крім того, розробка таких систем часто потребує менших витрат на інфраструктуру.

Натомість, основні недоліки монолітної архітектури в порівнянні з мікросервісами полягають у її меншій гнучкості при масштабуванні. Оскільки система є єдиним цілим, будь-яка зміна однієї частини може вплинути на всю структуру. Також, монолітні системи важче інтегрувати з іншими системами.

Монолітна архітектура підходить для невеликих або середніх за розміром систем з чітко визначеним і рідко змінюваним функціоналом, де важливі простота розгортання та підтримки.

Мікросервісна архітектура - це підхід до розробки програмного забезпечення, при якому система складається з низки невеликих, самостійних сервісів, кожен з яких відповідає за виконання конкретної функції. Кожен з таких сервісів зазвичай називають мікросервісом. Ці мікросервіси разом формують єдину децентралізовану систему, яка може бути реалізована за допомогою різних мов програмування та технологій зберігання даних, залежно від вимог задачі. Взаємодія між сервісами відбувається через прості протоколи, такі як HTTP, MQTT чи gRPC.

Мікросервісна архітектура має високу гнучкість і надійність. Завдяки її структурі, коли один мікросервіс виходить з ладу, інші продовжують працювати, що дозволяє швидко виправити помилку без порушення роботи всієї системи. Крім того, кожен мікросервіс можна оновлювати та масштабувати незалежно, що полегшує роботу розробників і пришвидшує процес.

Однак основними труднощами є складність у реалізації та супроводі такої архітектури, оскільки вона вимагає досвіду роботи з мікросервісами та спеціальних фахівців, таких як DevOps. Також мікросервіси вимагають більш потужної інфраструктури для підтримки контейнеризації та моніторингу.

Зазвичай мікросервісна архітектура найбільш ефективна для великих систем з широким функціоналом, що потребують постійного супроводу досвідченими фахівцями з відповідними навичками.

Порівняння монолітної та мікросервісної архітектур зазначено в табл. 1.2.

Таблиця 1.2 – Порівняльна характеристика наведених архітектур для побудови вебзастосунку

Критерій	Мікросервісна архітектура	Монолітна архітектура
Структура	Система складається з незалежних мікросервісів.	Все програмне забезпечення об’єднане в один монолітний блок.
Масштабування	Легко масштабувати окремі мікросервіси за потребою.	Масштабування системи вимагає масштабування всієї програми.
Гнучкість	Висока гнучкість, кожен мікросервіс можна оновлювати незалежно.	Оновлення одного компонента вимагає перезавантаження всієї системи.
Залежність	Мікросервіси можуть використовувати різні мови програмування та технології.	Вся система використовує одну технологічну платформу.
Захист від помилок	Помилка в одному мікросервісі не впливає на всю систему.	Помилка в будь-якому компоненті може призвести до збоїв у всій системі.

Кінець таблиці 1.2

Критерій	Мікросервісна архітектура	Монолітна архітектура
Управління	Необхідність в додаткових інструментах для моніторингу та управління мікросервісами.	Простота в управлінні, оскільки все зібрано в одній системі.
Вимоги до інфраструктури	Потрібна потужна інфраструктура для контейнеризації та оркестрації.	Необхідність в меншій інфраструктурі, оскільки вся система об’єднана.
Швидкість розробки	Може бути повільнішою через складність взаємодії між мікросервісами.	Швидка розробка, оскільки всі компоненти об'єднані в одному проекті.
Тестування	Тестування кожного мікросервісу окремо.	Тестування моноліту потребує більше часу через тісну взаємодію компонентів.
Вартість підтримки	Може бути дорожчою через необхідність в DevOps та моніторингу.	Менші витрати на підтримку, оскільки система є єдиним блоком.
Використання в великих проектах	Підходить для великих, складних систем з великим функціоналом.	Підходить для менших проектів або систем з обмеженим функціоналом.
Підтримка старих версій	Легко підтримувати старі версії мікросервісів незалежно.	Підтримка старих версій складніша, оскільки система є монолітом.

Мікросервісна та монолітна архітектури мають свої переваги й недоліки та підходять для різних типів проектів. Мікросервіси - це про масштабованість, гнучкість і надійність, але за ціною складнішої реалізації та вищих вимог до інфраструктури. Моноліт - це простота, швидкість розробки та менші витрати на старті, однак із обмеженнями у масштабуванні та підтримці в міру зростання системи. Вибір між цими підходами залежить від розмірів проекту, ресурсів команди та технічних вимог до системи.

1.4 Ключові аспекти та постановка задачі

Розробка вебзастосунку для продажу електроніки охоплює низку важливих етапів, серед яких:

– проєктування архітектури системи: на початковому етапі необхідно визначити загальну структуру системи, її ключові компоненти, принципи взаємодії між ними та розподіл функціональних обов’язків;

– створення користувацького інтерфейсу: інтерфейс має бути зрозумілим, зручним і логічним, з легким доступом до всіх основних функцій;

– розробка бази даних: база даних повинна забезпечувати швидкий доступ до інформації, бути масштабованою й надійною. Вона має містити повну інформацію про товари, замовлення, клієнтів тощо;

– інтеграція платіжних сервісів: для реалізації онлайн-оплати необхідно під’єднати до системи одну або кілька платіжних платформ;

– забезпечення безпеки: система повинна бути захищена від загроз, таких як несанкціонований доступ чи витік даних. Важливо впровадити комплекс заходів з безпеки;

– контроль якості коду та тестування: щоб уникнути помилок і забезпечити стабільну роботу застосунку, слід розробити стратегію тестування та дотримуватись стандартів якості програмного коду;

– оптимізація продуктивності: вебзастосунок повинен працювати швидко та ефективно. Для цього варто оптимізувати серверні ресурси, кешування та інші методи підвищення швидкодії;

– сумісність і доступність: застосунок має коректно відображатися в різних браузерах, бути адаптованим для мобільних пристроїв і доступним для користувачів;

– розгортання та подальша підтримка: після створення необхідно налаштувати сервер, розгорнути програмне забезпечення й забезпечити його безперебійну роботу. Підтримка та регулярне оновлення системи є обов’язковими для стабільної роботи.

Ці етапи становлять основу процесу створення ефективного та надійного вебзастосунку для електронної комерції.

Висновки до першого розділу

У процесі аналізу предметної області та постановки задачі для створення вебзастосунку з продажу електроніки було виявлено, що дана галузь охоплює великий асортимент товарів і послуг, що зумовлює потребу в сучасній, гнучкій та ефективній онлайн-платформі для їх реалізації.

Аналіз наявних рішень і тематичних публікацій показав, що на ринку присутня значна кількість вебзастосунків, орієнтованих на продаж електроніки. Проте їх функціональність, зручність користування та рівень безпеки суттєво відрізняються, що свідчить про актуальність створення продуманого та якісного вебзастосунку.

Ключові етапи розробки такого вебзастосунку охоплюють проєктування архітектури системи, створення зручного інтерфейсу, розробку бази даних, інтеграцію платіжних сервісів, забезпечення належного рівня безпеки, проведення тестування, підтримку якості коду, оптимізацію продуктивності, забезпечення сумісності з різними пристроями й доступності для користувачів, а також впровадження системи та її подальший підтримку.

Усі зазначені аспекти повинні бути враховані під час формування технічного завдання, щоб гарантувати успішну реалізацію вебзастосунку та його ефективне функціонування.

2 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА СЕРВІСИ, ЩО ВИКОРИСТОВУЮТЬСЯ ПРИ СТВОРЕНІ ВЕБЗАСТОСУНКУ

2.1 Характеристика обраного середовища розробки

Для створення вебзастосунку було обрано інтегроване середовище розробки Visual Studio Code (див. рис. 2.1) [9]. Це сучасний, безкоштовний, відкритий редактор коду, створений корпорацією Microsoft, який здобув популярність серед розробників у всьому світі завдяки своїй гнучкості, продуктивності та багатому функціоналу.

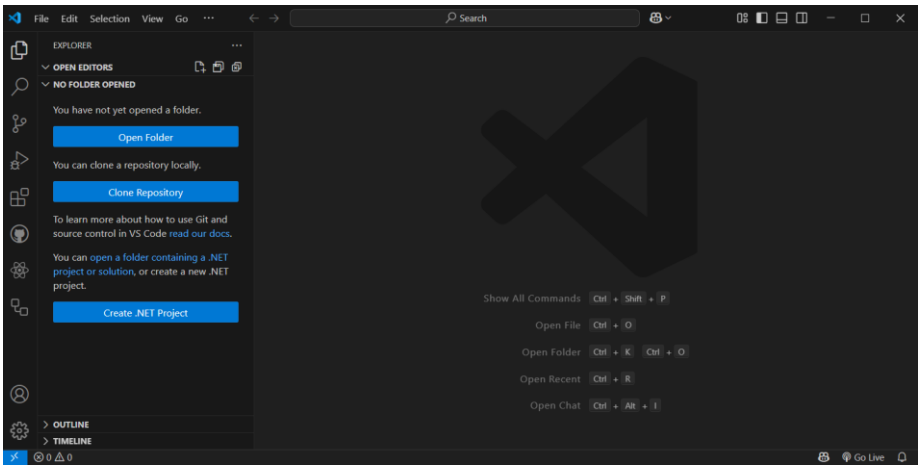

Рисунок 2.1 – Середовище розробки VS Code

Середовище доступне для встановлення та використання на основних операційних системах, що забезпечує універсальність та доступність, незалежно від операційної системи. Легкість встановлення та низькі системні вимоги дозволяють швидко розпочати роботу.

VS Code надає вбудовану підтримку для значної кількості мов програмування та технологій, включаючи ті, що є критично важливими для вебзастосунків, такі як HTML, CSS, JavaScript, TypeScript, JSON, а також базову підтримку для Python, PHP, C#, Java та багатьох інших. Ця підтримка включає підсвічування синтаксису, автоматичне доповнення коду, навігацію по коду, рефакторинг та інші функції, що підвищують швидкість та якість написання коду.

Екосистема розширень є одним із найпотужніших аспектів. Офіційний маркетплейс містить безліч розширень, які значно доповнюють функціональність середовища та адаптують його під будь-які потреби проєкту. Для вебзастосунків доступні розширення для підтримки специфічних фреймворків, бібліотек, баз даних, клієнтів для API, лінтерів, форматерів коду, інструментів для роботи з Docker, засобами для тестування та багато іншого. Це дозволяє створити високоспеціалізоване робоче середовище безпосередньо в редакторі.

Важливим інтегрованим інструментом є підтримка системи контролю версій Git. VS Code надає зручний графічний інтерфейс для виконання основних операцій Git: відстеження змін, створення комітів, управління гілками, виконання операцій push та pull, вирішення конфліктів злиття. Це спрощує командну роботу та управління історією проєкту, інтегруючи ці процеси безпосередньо в робочий процес розробника.

Крім того, середовище має потужний вбудований відлагоджувач, який підтримує відлагодження коду різних мов. Відлагоджувач дозволяє встановлювати точки зупинки, покроково виконувати код, переглядати значення змінних у реальному часі, аналізувати стек викликів, встановлювати умовні точки зупинки та точки зупинки на виключеннях. Це значно полегшує процес пошуку та виправлення помилок, скорочуючи час на відлагодження.

Для організації роботи над великими або мультипроєктними рішеннями VS Code підтримує концепцію робочих областей, що дозволяє групувати пов'язані папки та налаштування в одному вікні редактора. Це зручно при роботі над проєктами, що складаються з окремих фронтенд та бекенд частин.

Високий ступінь налаштовуваності VS Code досягається не тільки за рахунок розширень, але й через гнучку систему налаштувань, яка дозволяє адаптувати поведінку редактора, гарячі клавіші, візуальні теми, шрифти, правила форматування коду та інші параметри під індивідуальні переваги та специфіку проєкту. Налаштування можуть бути глобальними або специфічними для окремої робочої області.

Відкритий вихідний код та активна спільнота розробників сприяють постійному вдосконаленню VS Code, швидкому реагуванню на нові тенденції у веброзробці та виправленню помилок. Регулярні оновлення додають нові функції, покращення продуктивності та безпеки.

Враховуючи сукупність своїх переваг – легкість, швидкість роботи, крос-платформність, багатий функціонал, потужну екосистему розширень, інтегровані інструменти та високий ступінь налаштовуваності, є оптимальним вибором для розробки сучасних вебзастосунків. Він забезпечує всі необхідні інструменти для ефективного написання, відлагодження, тестування та розгортання коду, що робить процес розробки більш продуктивним, зручним та керованим.

2.2 Опис використання та особливостей технологічного стеку в розробці вебзастосунків

Для реалізації вебзастосунку було популярний набір технологій на основі JavaScript - MERN стек, призначений для швидкої та ефективної розробки повноцінних вебзастосунків. Абревіатура MERN розшифровується як:

– MongoDB – документо-орієнтована база даних NoSQL;
– Express.js – фреймворк для серверної частини, що працює на Node.js;
– React – бібліотека для побудови користувацьких інтерфейсів;
– Node.js – середовище виконання JavaScript на стороні сервера.

Використання цього стеку дозволяє розробляти як frontend, так і backend частини застосунку, використовуючи переважно одну мову програмування.

На рис. 2.2 наведено стек технологій MERN.

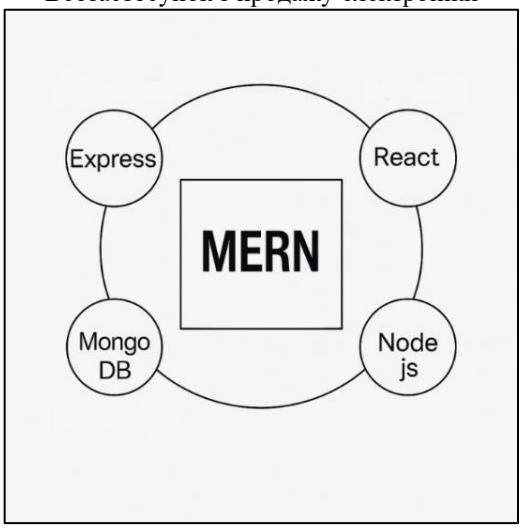

Рисунок 2.2 – Стек технологій MERN

MongoDB - це сучасна NoSQL база даних, яка зберігає інформацію у форматі гнучких документів JSON [10], що робить її зручною для зберігання складних та неоднорідних даних. Вона вирізняється високою продуктивністю, масштабованістю, а також простотою у налаштуванні та використанні. MongoDB особливо підходить для роботи з великими обсягами даних, що постійно змінюються, та дозволяє оперувати структурами даних без потреби дотримання жорстко визначеної схеми. Це робить її ідеальним вибором для сучасних вебзастосунків, систем аналітики та хмарних рішень.

На рис. 2.3 представлено базу даних MongoDB.

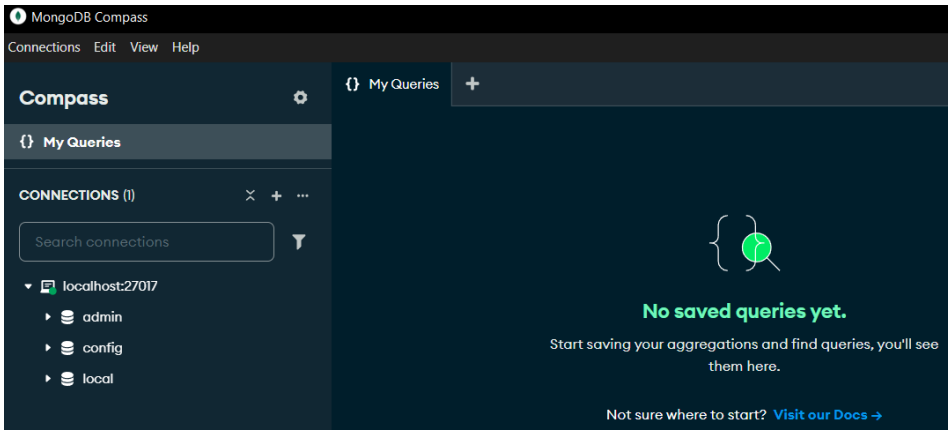

Рисунок 2.3 – База даних MongoDB

Express – це гнучкий, легкий і мінімалістичний серверний фреймворк, який широко використовується для створення вебзастосунків і RESTful API. Його основною перевагою є простота у використанні та швидкість розробки, що робить його популярним вибором серед розробників. Express надає зручний набір функцій для налаштування маршрутизації, обробки HTTP-запитів і відповідей, а також ефективної інтеграції проміжного програмного забезпечення (middleware), що дозволяє легко розширювати функціональність застосунку. Завдяки своїй гнучкій архітектурі, фреймворк підходить як для невеликих проектів, так і для масштабних систем.

На рис. 2.4 наведено фреймворк Express [11].

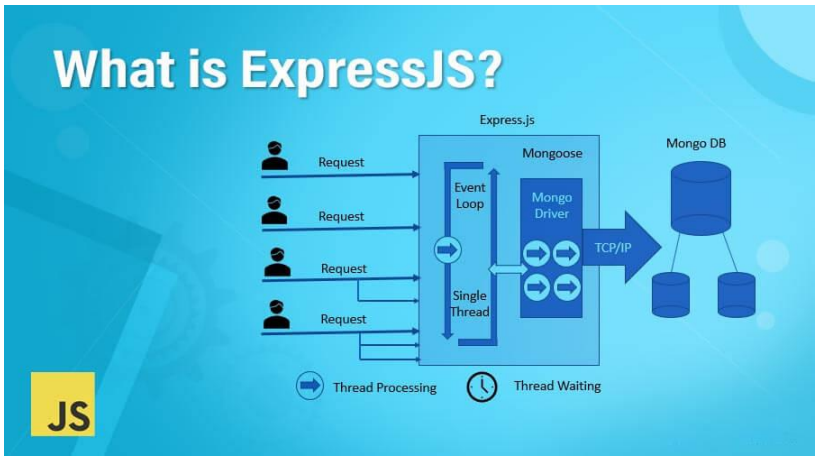

Рисунок 2.4 – Фреймворк Express

React - це популярний JavaScript-фреймворк React, розроблений компанією Facebook для створення UI [12]. Він базується на концепції компонентів, яка дозволяє розробникам будувати масштабні вебзастосунки, розділяючи інтерфейс на незалежні, зручні у використанні та багаторазові частини. Крім того, React використовує віртуальний DOM, що дозволяє ефективніше оновлювати сторінку й підвищувати продуктивність застосунку.

На рис. 2.5 наведено фреймворк React [13].

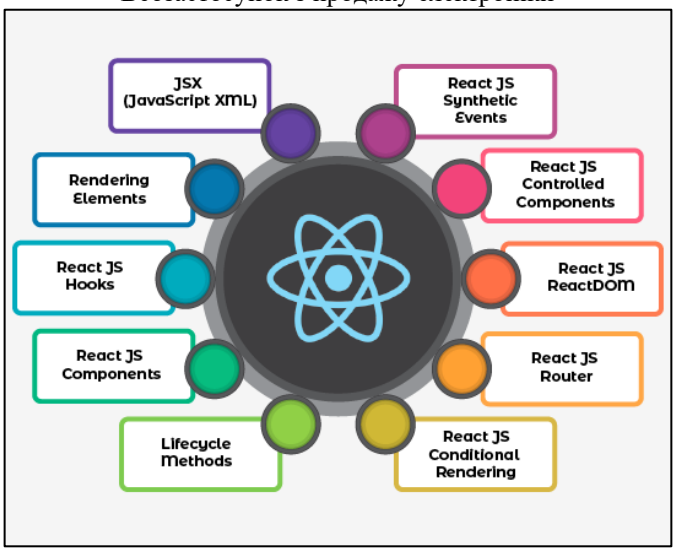

Рисунок 2.5 – Фреймворк React

Node.js - це кросплатформне середовище виконання JavaScript на сервері, яке працює на основі JavaScript-рушія від Google (див. рис. 2.6). Воно дозволяє використовувати JavaScript як на стороні клієнта, так і на стороні сервера, що значно спрощує процес розробки та подальшої підтримки проєктів. Node.js має розвинену екосистему модулів і пакетів, які розширюють функціональні можливості та полегшують створення додатків. Одним із найпоширеніших інструментів для керування цими пакетами є Node Package Manager (npm), який дозволяє зручно встановлювати, оновлювати та управляти залежностями в проєкті [14].

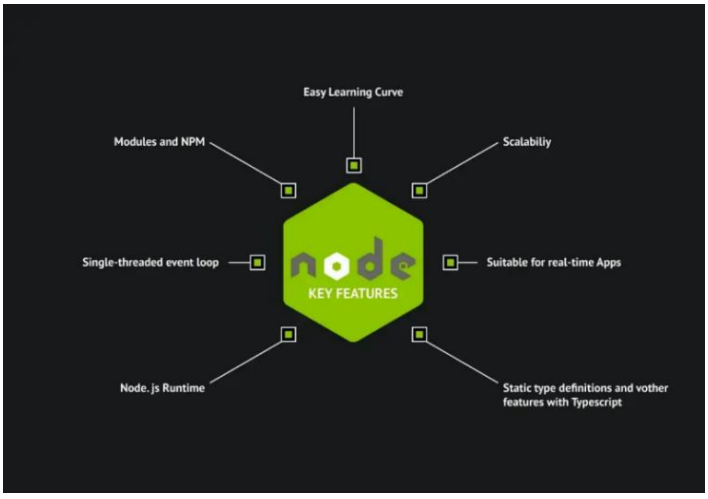

Рисунок 2.6 – Кросплатформне середовище Node.js

2.3 Характеристика використаних фреймворків

Сучасна розробка вебзастосунків потребує інструментів, що поєднують гнучкість, високу продуктивність і швидкодію. У цьому контексті фреймворки Next.js та React.js повністю відповідають цим критеріям, дозволяючи створювати надійні, ефективні та продуктивні вебрішення.

Next.js - це фреймворк, який розроблено спеціально для серверного рендерингу та побудови статичних вебсайтів з використанням React [16]. Він забезпечує розробників засобами для побудови швидких і SEO-оптимізованих вебзастосунків.

Цей фреймворк значно спрощує процес розробки, автоматизуючи такі аспекти, як серверний рендеринг, розподіл коду, статична оптимізація, підключення CSS. Крім того, Next.js має вбудовану маршрутизацію, яка базується на структурі файлової системи, що полегшує створення багатосторінкових вебзастосунків.

Next.js також сприяє пошуковій оптимізації вебзастосунків завдяки серверному рендерингу та можливості створення статичних сторінок. Це забезпечує повну доступність контенту для індексації пошуковими системами, що позитивно впливає на позиції сайту в результатах пошуку.

На рис. 2.7 наведено фреймворк Next.js [17].

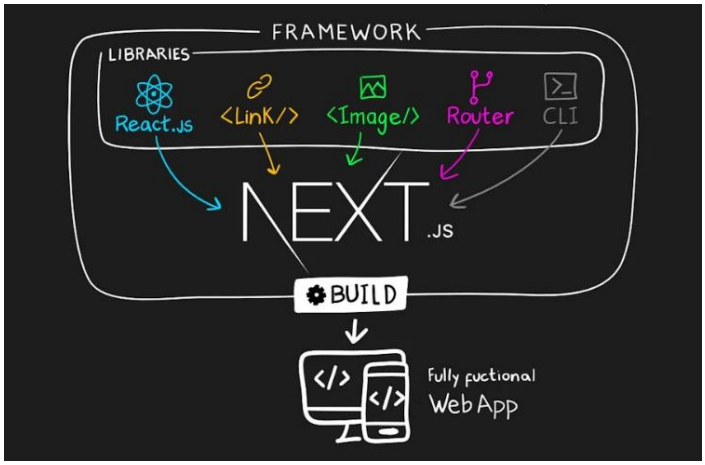

Рисунок 2.7 – Фреймворк Next.js

Ключова концепція React полягає в поділі інтерфейсу користувача на окремі, багаторазово використовувані компоненти. Кожен компонент у React може мати власний стан та життєвий цикл, що дає змогу створювати динамічні й функціонально насичені вебзастосунки.

Однією з переваг є використання віртуального DOM, який значно спрощує оновлення елементів на сторінці. Замість повного перезавантаження інтерфейсу при кожній зміні, React оновлює лише ті частини DOM, які зазнали змін. Це підвищує швидкодію застосунків і забезпечує плавну взаємодію з користувачем.

На рис. 2.8 наведено фреймворк React.

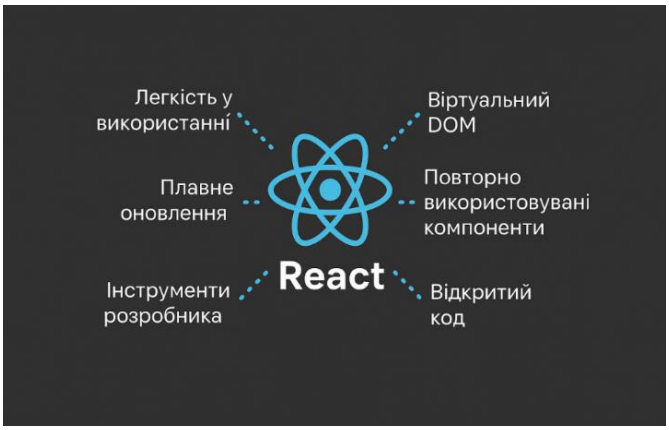

Рисунок 2.8 – Фреймворк React

2.4 Характеристика використаних бібліотек

Низка корисних бібліотек та інструментів, які можна застосовувати під час створення вебзастосунків. До переліку входять такі бібліотеки:

– бібліотека React Toastify призначена для відображення сповіщень (toast-повідомлень) у React-застосунках [18]. Дозволяє швидко інформувати користувача про успішні дії, помилки або інші події у зручному форматі, що не потребує оновлення сторінки. Підтримує автоматичне зникнення повідомлень, налаштування позиції та стилю;

– бібліотека Stripe: клієнтська та серверна бібліотека для інтеграції платіжної системи у вебзастосунки [19]. Підтримує обробку платежів, керування підписками, збереження платіжних даних та повернення коштів. Stripe широко

використовується завдяки високому рівню безпеки та зручній інтеграції з React та Node.js;

– бібліотека Chart.js: гнучка бібліотека з відкритим кодом для створення інтерактивних графіків та діаграм [20]. Підтримує різні типів діаграм, рендерить графіки в HTML5 canvas та забезпечує чудову продуктивність в сучасних браузерах. Бібліотека дозволяє гнучко налаштовувати стилі, анімації, легенди та підказки, роблячи її популярним вибором для візуалізації даних завдяки простоті використання та широким можливостям кастомізації;

– бібліотека JSON Web Token: ключовий інструмент у реалізації безпечної автентифікації та авторизації у вебзастосунках [21]. Забезпечує безпечну передачу інформації між клієнтом і сервером, дозволяє реалізувати безсесійну авторизацію;

– бібліотека bcryptjs використовується у вебзастосунках для хешування паролів перед збереженням у базі даних [22]. Реалізує алгоритм bcrypt, який забезпечує стійкість до атак перебору, та є стандартом безпечної обробки облікових даних користувач;

– бібліотека EmailJS дозволяє реалізувати надсилання електронних листів прямо з клієнтської частини вебзастосунку без необхідності налаштовувати сервер [23]. Підключається до популярних email-сервісів і використовується для реалізації функцій надсилання підтверджень замовлень, повідомлень про доставку, відновлення пароля;

– бібліотека Recharts - ще один інструмент для побудови графіків, спеціально розроблений для React [24]. Підходить для створення аналітичних дашбордів у вебзастосунках, зокрема статистики продажів, активності користувачів або популярності продуктів;

– бібліотека Mongoose: бібліотека для Node.js, яка спрощує роботу з MongoDB базою даних [25]. Дозволяє створювати структуровані схеми для документів, додавати валідацію даних та виконувати основні операції з базою даних через зручний інтерфейс. Mongoose робить роботу з MongoDB більш

організованою та безпечною, надаючи інструменти для контролю структури даних та їх перевірки;

На рис. 2.9 наведено принцип роботи Mongoose.

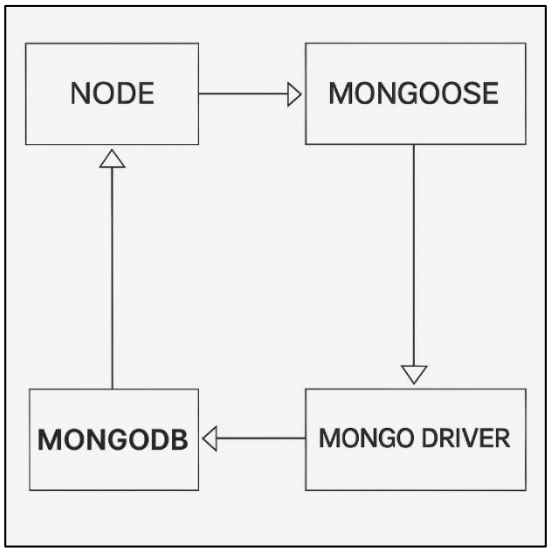

Рисунок 2.9 – Принцип роботи Mongoose

– бібліотека React Lazy Load Image Component призначена для оптимізації завантаження зображень за допомогою ледачого завантаження [26]. Це особливо важливо в магазинах із великою кількістю товарів і зображень, що дозволяє значно прискорити завантаження сторінки

– бібліотека Draggable: надає простий і гнучкий API, що дозволяє швидко додавати можливість перетягування, сортування та групування елементів без необхідності писати складний код [27]. Підтримує різні типи подій, налаштування обмежень руху та інтеграцію з іншими компонентами, що забезпечує високий рівень кастомізації та зручність використання;

– CSS-бібліотека Tailwind CSS: пропонує набір низькорівневих класів для створення вебдизайну [28]. Tailwind дає змогу будувати власні унікальні дизайни, використовуючи CSS-класи. Такий підхід забезпечує високу гнучкість і контроль, дозволяючи створювати налаштовувані стилі без потреби змінювати готові шаблони.

На рис. 2.10 наведено приклад використання бібліотеки Tailwind CSS.

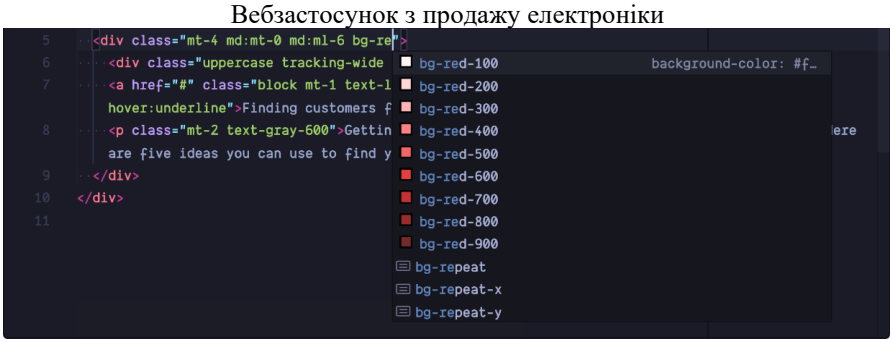

Рисунок 2.10 – Бібліотека Tailwind CSS

Порівняння вищеописаних бібліотек буде подано у вигляді табл. 2.1.

Для структурного порівняння треба використати наступні критерії:

– призначення;

– ключові можливості;

– сумісність.

Таблиця 2.1 – Порівняльна характеристика наведених бібліотек

Назва бібліотеки	Призначення	Ключові можливості	Сумісність
React Toastify	Відображення сповіщень	Toast-повідомлення, налаштування стилів	React
Stripe	Платіжна система	Обробка платежів, підписки, безпека	React, Node.js
Chart.js	Графіки та діаграми	8 типів діаграм, анімації, налаштування	JavaScript
JSON Web Token	Автентифікація	Безсесійна авторизація, безпека даних	Вебзастосунки
bcryptjs	Хешування паролів	Алгоритм bcrypt, захист від атак	Вебзастосунки
EmailJS	Надсилання email	Відправка без сервера, шаблони листів	Клієнтська частина
Recharts	Графіки для React	Аналітичні дашборди, статистика	React
Mongoose	Робота з MongoDB	CRUD, схеми, валідація, пошук	Node.js, MongoDB
React Lazy Load Image	Оптимізація зображень	Ледаче завантаження, прискорення	React

Кінець таблиці 2.1

Назва бібліотеки	Призначення	Ключові можливості	Сумісність
Draggable	Перетягування елементів	Drag & drop, сортування, групування	JavaScript
Tailwind CSS	CSS-фреймворк	Утилітарні класи, гнучкість дизайну	Вебзастосунки,CSS

Також однією з найпопулярніших та потужних бібліотек для здійснення HTTP-запитів у JavaScript є Axios [29]. Вона забезпечує зручний і лаконічний синтаксис, що значно полегшує взаємодію з зовнішніми API. Завдяки Axios можна легко надсилати запити типу GET, POST, PUT, DELETE та обробляти відповіді сервера. Бібліотека має вбудовану підтримку обробки помилок, конфігурації таймаутів, а також дозволяє гнучко налаштовувати заголовки запитів. Axios підтримує як браузерне середовище, так і Node.js, що робить її універсальним інструментом для створення клієнтсько-серверної логіки у вебзастосунках.

Крім того, Axios підтримує асинхронне програмування через проміси, що дозволяє ефективно керувати запитами та відповідями, уникаючи складних вкладених структур коду. Веброзробники можуть за допомогою цієї бібліотеки не лише надсилати дані на сервер, а й динамічно оновлювати контент сторінки без перезавантаження, забезпечуючи плавну та інтерактивну взаємодію з користувачем. Завдяки своїй гнучкості та простоті використання, Axios широко використовується у сучасних вебпроєктах.

Використання сучасних бібліотек у веброзробці є необхідністю, що дозволяє досягнути значно вищого рівня продуктивності та зручності в процесі розробки. Вони надають розробникам готові, оптимізовані рішення для типових завдань, від абстрагування складнощів роботи з DOM та керування станом на клієнті до ефективної обробки запитів та взаємодії з даними на сервері. Це дозволяє зосередитися на реалізації унікальної логіки проєкту, мінімізуючи витрати часу.

Крім того, бібліотеки візуального рівня відіграють ключову роль у створенні привабливого, інтерактивного та високоякісного інтерфейсу для кінцевого

користувача. Вони забезпечують швидке завантаження, плавну анімацію, адаптивність до різних пристроїв та покращений загальний користувацький досвід.

Поєднання візуальних та серверних бібліотек формує потужний, повноцінний стек інструментів. Така синергія дозволяє ефективно організувати обмін даними між клієнтом і сервером, підтримувати єдину архітектуру застосунку та спрощує інтеграцію різноманітних функціональних можливостей. Це забезпечує надійну основу для реалізації масштабованих, гнучких та високоефективних вебзастосунків, здатних витримувати зростаючі навантаження та легко адаптуватися до нових вимог.

Кожна з розглянутих бібліотек виконує свою специфічну, але важливу роль у загальній архітектурі застосунку. Їхнє виважене комбінування та ефективне використання дозволяє створити модульну структуру з чітким розділенням відповідальності, що спрощує підтримку, тестування та подальший розвиток проєкту. Використання сучасних бібліотек у веброзробці дозволяє досягнути високої продуктивності, зручності в розробці та привабливого інтерфейсу для кінцевого користувача. Поєднання візуальних та серверних бібліотек забезпечує повноцінний стек інструментів, що дає змогу реалізовувати масштабовані, гнучкі та ефективні вебзастосунки. Кожна з розглянутих бібліотек виконує конкретну роль, і разом вони формують надійну основу для створення сучасних вебзастосунків.

Висновки до другого розділу

Проведено детальний аналіз інструментів та технологій, що були використані під час розробки вебзастосунку. Спершу розглянуто обране середовище розробки, яке забезпечує високу гнучкість, продуктивність і підтримку великої кількості розширень, що полегшують роботу з JavaScript та пов’язаними технологіями.

Далі було представлено огляд технологічного стеку, який виступає основою створення повноцінного вебзастосунку. Кожен із компонентів стеку виконує важливу роль у забезпеченні функціональності та стабільності застосунку.

Також було описано використані фреймворки, їх ключові особливості та переваги у контексті створення сучасних вебзастосунків. Окрему увагу приділено додатковим бібліотекам, які надають розширену функціональність і спрощують процес розробки.

У підсумку, в цьому розділі було послідовно описано ключові технології та інструменти, обрані для реалізації вебзастосунку. Такий вибір є результатом зваженого підходу, що сприяє ефективній розробці та надійному впровадженню проєкту.

3 ВПРОВАДЖЕННЯ ОБРАНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ВЕБЗАСТОСУНКУ

3.1 Технічна реалізація адміністративної панелі

Адміністративна панель представляє собою ключовий елемент архітектури вебзастосунку, який служить універсальним інструментом для комплексного управління всіма функціональними модулями системи. У цьому розділі докладно розглядається процес створення адміністративної панелі.

3.1.1 Розробка серверної частини та інтеграція з базою даних

Серверна частина вебзастосунку (backend) обробляє запити клієнтів, взаємодіє з базами даних і повертає відповіді. Щоб упорядкувати код, він організований у спеціальні каталоги та файли. Серед найважливіших папок - components, libraries, models, routes, pages, а також конфігураційний файл package.json (див. рис. 3.1). Кожен із цих елементів виконує власну функцію в архітектурі проєкту.

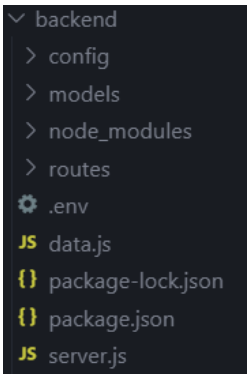

Рисунок 3.1 – Структура директорій

– директорія config містить конфігураційні файли проекту, включаючи налаштування з'єднання з базою даних MongoDB, секретні ключі для JWT автентифікації та інші параметри середовища;

– директорія models містить схеми та моделі MongoDB для проекту. Кожна модель визначає структуру даних (як User, Product, Order, Category) з відповідними полями, валідацією та методами для взаємодії з базою даних;

– директорія routes містить файли, відповідальні за налаштування кінцевих точок у проєкті. Кожна точка обробляє певний тип запиту та працює з відповідними даними. Файли організовані за функціональними напрямками, наприклад: productRoutes.js, userRoutes.js, orderRoutes.js, categoryRoutes.js;

– package.json призначений для керування залежностями проекту. У ньому вказано перелік необхідних для роботи застосунку пакетів із їхніми версіями, скрипти для запуску проекту та інші налаштування.

База даних відіграє фундаментальну роль у функціонуванні будь-якого вебзастосунку, що має справу з обробкою інформації. У межах цього проєкту було обрано MongoDB - сучасну NoSQL-платформу, орієнтовану на роботу з документами, яка дозволяє гнучко й масштабовано керувати даними. Для організації стабільного та безпечного з’єднання з MongoDB було реалізовано два основних модулі: server.js та db.js.

Файл db.js виконує роль посередника між застосунком і базою даних, ініціюючи та налаштовуючи з’єднання з MongoDB за допомогою Mongoose. Mongoose - це ODM (Object Data Modeling) бібліотека для MongoDB та Node.js, яка забезпечує більш простий спосіб роботи з MongoDB, додаючи структуру даним через схеми та моделі. У цьому файлі реалізовано функцію connectDB, яка відповідає за перевірку наявності середовищної змінної MONGO_URI, що слугує джерелом URL-адреси для підключення до бази даних MongoDB. Якщо з'єднання відбувається успішно, виводиться повідомлення про успішне підключення в консоль. Якщо виникає помилка, вона логується в консоль, і процес додатку завершується з кодом помилки. Також налаштовуються додаткові параметри підключення, такі як useNewUrlParser і useUnifiedTopology, щоб уникнути попереджень про застарілі опції.

Файл server.js є головним файлом, який ініціалізує та запускає Express сервер, який надає надійний набір функцій для веб та мобільних додатків. Файл імпортує необхідні залежності, такі як express, cors, path та dotenv. Dotenv використовується для завантаження змінних середовища з файлу .env. Потім створюється екземпляр

Express додатку, налаштовуються middleware для обробки JSON, URL-encoded даних та CORS. Викликається функція connectDB з модуля db.js для підключення до бази даних. Налаштовуються шляхи для статичних файлів і підключаються маршрути API, такі як userRoutes, productRoutes, orderRoutes та categoryRoutes. Створюється маршрут для завантаження зображень. Нарешті, сервер запускається на визначеному порту , і в консоль виводиться повідомлення про успішний запуск.

У сукупності ці два файли формують надійну основу для взаємодії з MongoDB, гарантуючи безперебійне підключення та створюючи умови для повноцінного виконання всіх CRUD-операцій у межах вебзастосунку.

3.1.2 Створення структур бази даних для організації даних

Схеми бази даних відіграють ключову роль у впорядкуванні, збереженні та ефективному доступі до інформації вебзастосунку. Визначається структура даних, включаючи типи сутностей, їхні атрибути та взаємозв'язки між ними. Представлено детальний опис реалізації п’яти основних моделей бази даних, що були використані у рамках цього проєкту: User, Product, Order, Category та OrderItem. Кожна з цих моделей виконує специфічну функцію в системі: забезпечує управління користувачами, класифікацію товарів, обробку замовлень, зберігання інформації про продукти.

Модель userModel.js містить схему користувача для збереження інформації про зареєстрованих користувачів. Схема включає поля: ім'я користувача, унікальна електронна пошта, зашифрований пароль, логічне поле, що вказує на адміністративні права користувача. Схема також містить методи для порівняння паролів при авторизації, використовуючи bcrypt для перевірки зашифрованих паролів.

На рис. 3.2 наведено лістинг коду схеми користувача.

```
const mongoose = require('mongoose');
const userSchema = new mongoose.Schema
({
name: {type: String,required: true},
email: {type: String,required: true,unique: true},
password: {type: String,required: true},
isAdmin: {type: Boolean,required: true,default: false}
});
```

Рисунок 3.2 – Лістинг коду схеми користувача

На рис. 3.3 наведено модель користувача у базі даних.

Рисунок 3.3 – Модель користувача у базі даних

Модель product.js визначає схему товару, яка зберігає всю інформацію про товари в магазині. Схема включає поля: назва товару, опис товару, ціна товару, шлях до зображення товару, ID категорії, кількість товару в наявності. Схема створює індекс для поля name для оптимізації пошуку товарів за назвою.

На рис. 3.4 наведено лістинг коду схеми товару.

```
const mongoose = require('mongoose');
const productSchema = new mongoose.Schema({
name: {type: String,required: true},description: {type: String,required: true},
price: {type: Number,required: true},
category: {type: mongoose.Schema.Types.ObjectId,ref: 'Category',required: true},
image: {type: String,required: true},
countInStock: {type: Number,required: true,default: 0}});
```

Рисунок 3.4 – Модель товару у базі даних

На рис. 3.5 наведено модель товару у базі даних.

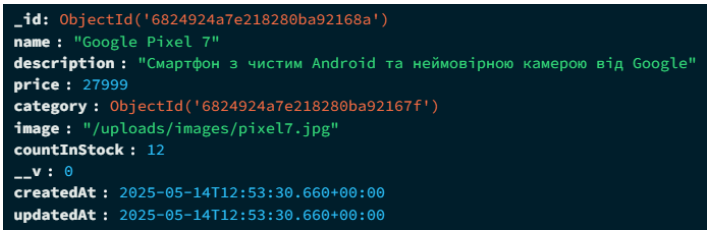

Рисунок 3.5 – Модель товару у базі даних

Модель order.js містить схему замовлення, яка зберігає інформацію про замовлення користувачів. Схема включає поля: ID користувача, масив об'єктів з інформацією про товари, загальна сума замовлення, статус оплати, дата оплати, статус доставки, дата доставки.

На рис. 3.6 наведено лістинг коду схеми замовлення.

```
const mongoose = require('mongoose');
const orderSchema = new mongoose.Schema({
user: {type: mongoose.Schema.Types.ObjectId,required: true,ref: 'User'},
orderItems: [{
name: { type: String, required: true },
quantity: { type: Number, required: true },
image: { type: String, required: true },
price: { type: Number, required: true },
product: {type: mongoose.Schema.Types.ObjectId,required: true,ref: 'Product'}}],
totalPrice: {type: Number,required: true,default: 0},
isPaid: {type: Boolean,required: true,default: false});
```

Рисунок 3.6 – Лістинг коду схеми замовлення

На рис. 3.7 наведено модель замовлення у базі даних.

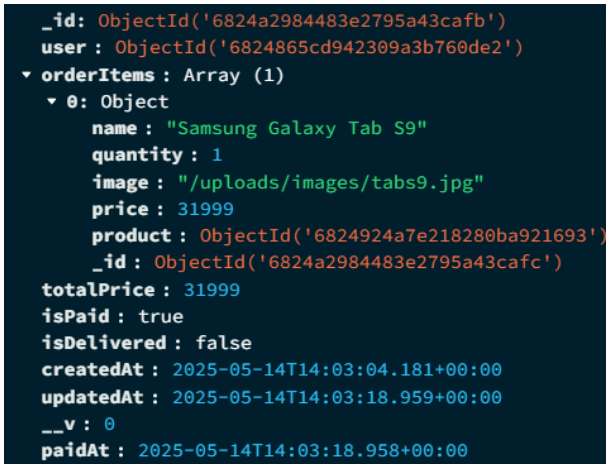

Рисунок 3.7 – Модель замовлення у базі даних

Модель category.js визначає схему категорії товарів. Схема включає поля: унікальна назва категорії, опис, шлях до зображення категорії). Ця схема дозволяє організувати товари в категорії для зручнішого пошуку та навігації по магазину.

На рис. 3.8 наведено лістинг коду схеми категорії.

```
const mongoose = require('mongoose');
const categorySchema = new mongoose.Schema({
name: {type: String,required: true,unique: true},
description: {type: String,required: false},
image: {type: String,required: false}}, {
timestamps: true});
module.exports = mongoose.model('Category', categorySchema);
```

Рисунок 3.8 – Лістинг коду схеми категорії

На рис. 3.9 наведено моделі категорії у базі даних.

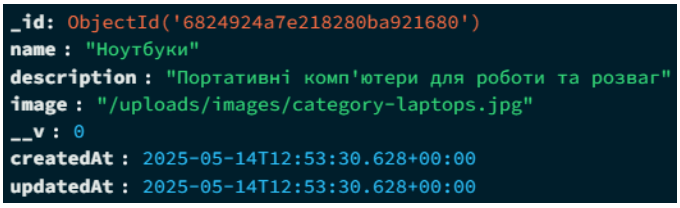

Рисунок 3.9 – Модель категорії у базі даних

Всі схеми перетворюються на моделі MongoDB з відповідними методами для операцій CRUD (створення, читання, оновлення, видалення) та експортуються для використання в інших частинах програми, зокрема в маршрутах API для відповідних типів даних.

3.1.3 Створення API-ендпоінтів застосунку

Точки доступу API відіграють ключову роль у налагодженні взаємодії між серверною логікою вебзастосунку та адміністративною панеллю [30]. Вони виступають посередниками між інтерфейсом користувача та базою даних, забезпечуючи можливість виконання основних дій із даними, таких як їхнє створення, модифікація, видалення та отримання. У цьому підрозділі буде проаналізовано API endpoints, які задіяні для реалізації базового функціоналу, зокрема:

– users дозволяє керувати користувачами системи. Надає функціонал реєстрації нових користувачів, авторизації, отримання та оновлення профілю користувача. Для адміністраторів доступні додаткові можливості: перегляд списку

всіх користувачів, видалення користувачів та призначення адміністративних прав. Автентифікація здійснюється за допомогою JWT токенів;

– categories забезпечує повний контроль над категоріями продуктів, дозволяє додавати категорії, змінювати їхні властивості, видаляти непотрібні, а також отримувати повний перелік доступних категорій для ефективного управління асортиментом. Категорії використовуються для організації товарів у логічні групи, що полегшує навігацію. Категорія може мати назву, опис та зображення;

– orders забезпечує функціонал для роботи з замовленнями. Дозволяє авторизованим користувачам створювати нові замовлення, переглядати список своїх замовлень та отримувати детальну інформацію про конкретне замовлення. Для адміністраторів доступні функції перегляду всіх замовлень, позначення замовлень як оплачених або доставлених, а також видалення замовлень. API також відстежує статуси оплати та доставки замовлень;

– products відповідає за управління товарами. Дозволяє отримувати список всіх товарів з підтримкою пагінації, шукати товари за ключовими словами, отримувати детальну інформацію про конкретний товар. Адміністратори можуть створювати нові товари, оновлювати існуючі та видаляти їх. Також API підтримує отримання товарів за категорією та надає рекомендації товарів на основі вже переглянутих товарів;

– upload забезпечує функціонал для завантаження файлів, переважно зображень для товарів та категорій. Використовує бібліотеку Multer для обробки файлових завантажень. Підтримує валідацію типів файлів (дозволені лише зображення у форматах JPEG, PNG, GIF), встановлює обмеження на розмір файлів та автоматично генерує унікальні імена файлів для уникнення конфліктів при зберіганні. Завантажені зображення зберігаються в директорії uploads/images і стають доступними через статичний URL.

Кожен API endpoint виконує чітко визначену роль та реалізує специфічні функції, необхідні для коректної роботи вебзастосунку. Вони розроблені з урахуванням принципів зручності, безпеки та ефективності, що дозволяє значно

спростити процес управління даними. Завдяки структурованості та узгодженості цих кінцевих точок, адміністратори можуть швидко взаємодіяти з системою - здійснювати пошук, редагування, видалення або створення нової інформації без необхідності глибокого технічного втручання.

На рис. 3.10 наведено лістинг коду прикладу кінцевої точки.

```
router.get('/category/:categoryId', async (req, res) => {
  try {
    const products = await Product.find({ category: req.params.categoryId })
      .populate('category', 'name');
    res.json(products);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});
```

Рисунок 3.10 – Лістинг коду прикладу кінцевої точки

На рис. 3.11 наведено перелік доступних у проєкті API-ендпоінтів.

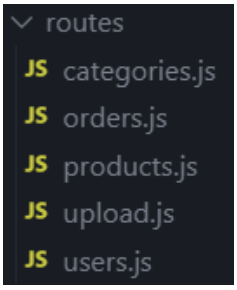

Рисунок 3.11 – API-ендпоінти проєкту

Найчастіше використовуються GET, POST, PUT, PATCH та DELETE методи у веб-розробці. Вони лежать в основі взаємодії між клієнтом і сервером та є невід'ємною частиною створення RESTful API. Кожен з цих методів виконує певну функцію:

– GET метод застосовується для отримання даних із сервера без зміни його стану. Наприклад, з його допомогою можна запитати інформацію про конкретний продукт або отримати список усіх доступних товарів. Це найпоширеніший метод для читання ресурсів;

– POST метод використовується для створення нових ресурсів на сервері. Наприклад, коли користувач надсилає форму для додавання нового продукту або оформлення замовлення, дані передаються за допомогою POST;

– PUT метод призначений для повного оновлення наявного ресурсу. Наприклад, якщо потрібно змінити всі дані про продукт, PUT повністю замінює існуючий запис новим;

– PATCH метод аналогічний PUT, але використовується для часткового оновлення ресурсу. Наприклад, якщо потрібно змінити лише назву продукту або ціну без впливу на інші поля;

– DELETE метод відповідає за видалення ресурсів із сервера. Його можна використовувати для видалення конкретного продукту, користувача або замовлення.

Ці методи разом формують повноцінний набір інструментів для реалізації CRUD-операцій, що є основою для створення гнучких та ефективних веб-сервісів. RESTful API, побудований на цих принципах, забезпечує зрозумілу та передбачувану структуру для обміну даними між клієнтом і сервером.

3.1.4 Реалізація авторизації через електронну пошту

Використання електронної пошти як засобу аутентифікації не лише спрощує процес входу для користувачів, а й додає додатковий рівень захисту їхніх особистих даних, оскільки система використовує надійні механізми верифікації через email. Зменшується ризик несанкціонованого доступу, оскільки підтвердження особи здійснюється через особисту електронну скриньку користувача. Такий підхід позитивно впливає на загальний користувацький досвід, роблячи процес автентифікації швидким і інтуїтивно зрозумілим, а також зміцнює довіру до вебзастосунку, оскільки користувачі бачать, що їхні дані захищені сучасними засобами

Для реалізації цього підходу доцільно використати сервіс EmailJS, який надає зручні інструменти для надсилання листів із підтверджувальними кодами або

посиланнями для входу. EmailJS дозволяє швидко інтегрувати функціонал надсилання електронних листів у вебзастосунок без необхідності налаштовувати власний сервер. Також підтримує інтеграцію з популярними email-провайдерами, такими як Gmail, Outlook, Yahoo або ж дає змогу використовувати власний SMTP-сервер, що забезпечує гнучкість та адаптацію під конкретні потреби проєкту.

Порівняння поштових сервісів подано у вигляді табл. 3.1.

Таблиця 3.1 – Порівняльна характеристика поштових сервісів

Назва сервісу	Переваги	Недоліки
Gmail	Потужний пошук, ефективне фільтрування спаму та висока надійність з гарантією	Обмеження до 2000 щоденних одержувачів та можливість потрапляння листів у спам
Outlook	Інтеграція з Office 365, можливість відкликати листи, просунуті інструменти управління поштою	Ліміт до 500 одержувачів в одному листі , слабке блокування спаму та складне меню налаштувань
Yahoo	Безкоштовність, щедрий обсяг сховища та відсутність реклами в інтерфейсі	Незважаючи на переваги, має обмежені бізнес-функції та інтеграції з іншими сервісами
Власний SMTP-сервер	Відсутність обмежень на обсяг розсилок та повний контроль над безпекою і приватністю	Потребує технічної підтримки, ризик високого відсотка відмов та потрапляння до чорних списків

Для більшості сучасних вебзастосунків на етапі розробки доцільно використовувати Gmail. як найпростіший і швидкий у налаштуванні варіант, який не потребує складних інтеграцій чи додаткових витрат.

Однак, коли йдеться про масштабовані комерційні рішення, Gmail може виявитися недостатньо гнучким або обмеженим за функціоналом і політиками

масових розсилок. У таких випадках доцільно перейти на власний SMTP-сервер або скористатися професійними сервісами надсилання листів, як SendGrid чи Mailgun. Власний SMTP-сервер дає повний контроль над даними та листуванням, дозволяє налаштовувати автентифікацію, управляти доставкою та уникати сторонніх обмежень, проте вимагає технічної підтримки та постійного адміністрування.

На рис. 3.12 наведено список доступних сервісів надсилання листів.

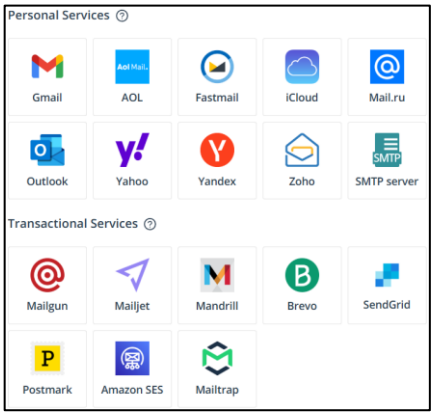

Рисунок 3.12 – Список доступних сервісів надсилання листів

Далі вводимо назву сервісу, а також заповнюємо всі необхідні поля у вкладці налаштування, включаючи параметри доступу, опис, специфікації та інші важливі дані, необхідні для коректної роботи сервісу.

На рис. 3.13 наведено конфігурацію самого сервісу.

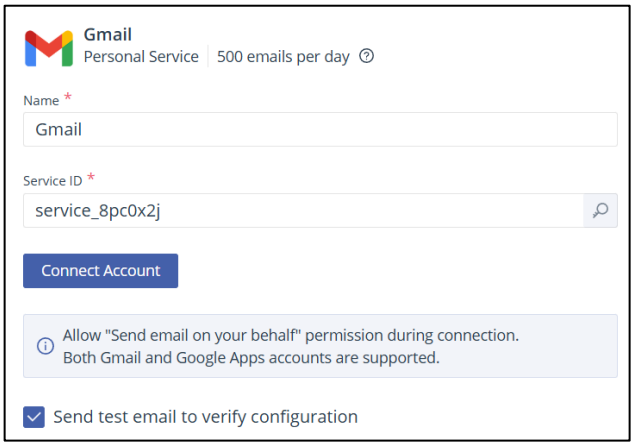

Рисунок 3.13 – Конфігурація сервісу

Після цього переходимо до створення шаблону листа, який отримуватимуть користувачі. У шаблоні зазначаємо заголовок, основний текст повідомлення, а також, за потреби, додаємо динамічні поля, посилання для персоналізації листа та підвищення його ефективності.

На рис. 3.14 наведено шаблон вітального листа.

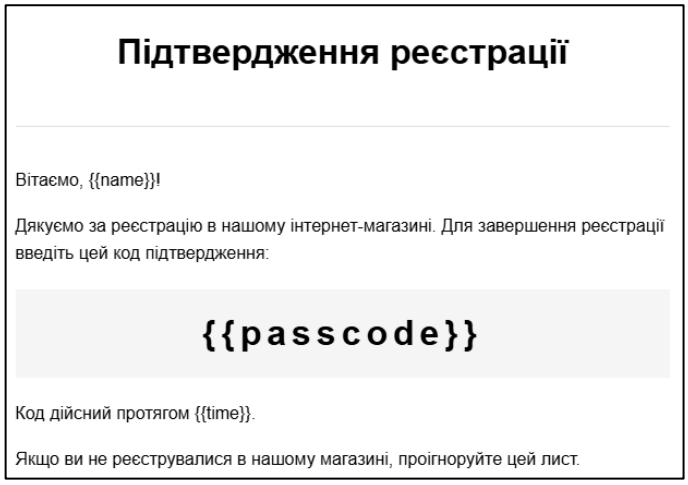

Рисунок 3.14 – Шаблон вітального листа

Після створення сервісу та шаблону листа з’явиться меню налаштувань, у якому будуть доступні всі необхідні дані для подальшої інтеграції, зокрема «Public Key» та «Private Key». Ці ключі слід зберегти, оскільки вони використовуються для автентифікації та безпечного обміну даними між сервісами.

На рис. 3.15 наведено API ключи.

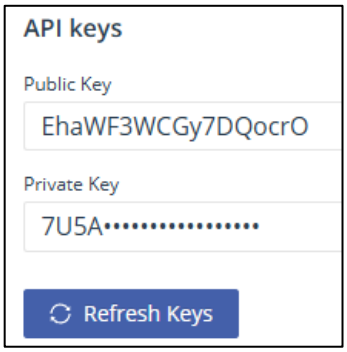

Рисунок 3.15 – API ключи

Далі редагуємо відповідний файл у проекті під назвою «.env», який зберігає усі секретні дані, ключі, тощо. Властивості цього файлу включають:

– emailjs_public_key: публічний ключ, який використовується для автентифікації клієнтських запитів до API EmailJS. Цей ключ не є секретним і може використовуватись у браузері;

– emailjs_private_key: приватний ключ, який використовується лише для серверних запитів;

– emailjs_service_id: унікальний ідентифікатор email-сервісу, який підключилено у EmailJS (наприклад, Gmail, Outlook або SMTP). EmailJS дозволяє мати декілька сервісів, кожен має свій Service ID;

– emailjs_template_id: ідентифікатор шаблону листа, який створюється у розділі Email Templates. Шаблон містить текст листа та змінні (наприклад, ім’я користувача, код підтвердження), які будуть підставлені під час надсилання.

Надання доступу до адміністративної панелі всім користувачам є недоцільним, адже це може спричинити несанкціоноване редагування, видалення чи додавання інформації на сайті. З цієї причини було впроваджено роль адміністратора.

Адміністратори отримують виключний доступ до адміністративної панелі, що дозволяє їм ефективно управляти даними вебзастосунку відповідно до своїх обов’язків і потреб. Це рішення сприяє захисту інформації та забезпечує, що лише уповноважені особи можуть впливати на функціонування та вміст сайту.

У разі, якщо користувач має статус адміністратора сайту, він отримає можливість увійти до системи адміністрування. В іншому випадку йому буде показано сторінку з повідомленням про відмову в доступі, а також автоматичне перенаправлення на головну сторінку сайту.

3.2 Технічна реалізація веб-магазину

Інтернет-магазин виступає як основна клієнтська частина вебзастосунку, що забезпечує користувачам зручний інтерфейс для перегляду, вибору та придбання

різноманітних товарів. У цьому підрозділі докладно описано процес розробки та впровадження Інтернет-магазину із застосуванням обраних сучасних технологій, інструментів та сервісів. Розглядаються ключові етапи створення, такі як проектування інтерфейсу, інтеграція з серверною частиною, налаштування бази даних, а також впровадження функціоналу для здійснення покупок і керування замовленнями.

3.2.1 Інтеграція проєкту, схем і API з базою даних

Оскільки інтернет-магазин і адміністративна панель працюють з однією й тією ж базою даних, підключення до неї здійснюється через спільні конфігураційні файли server.js і db.js, які були описані раніше. Це дозволяє підтримувати цілісність, узгодженість та єдиний підхід до взаємодії з базою даних в обох частинах проєкту.

У межах інтернет-магазину додатково реалізовано кілька специфічних моделей, призначених для обробки та зберігання даних, пов’язаних із функціональністю магазину.

Зокрема, модель Address відповідає за зберігання інформації, необхідної для доставки товарів користувачам. Вона містить такі поля, як: обліковий запис користувача, країна, місто, поштовий індекс та інші дані, що дозволяють точно ідентифікувати місце доставки. Завдяки цій моделі забезпечується ефективне управління адресами користувачів і коректна обробка замовлень.

На рис. 3.16 наведено модель адреси у базі даних.

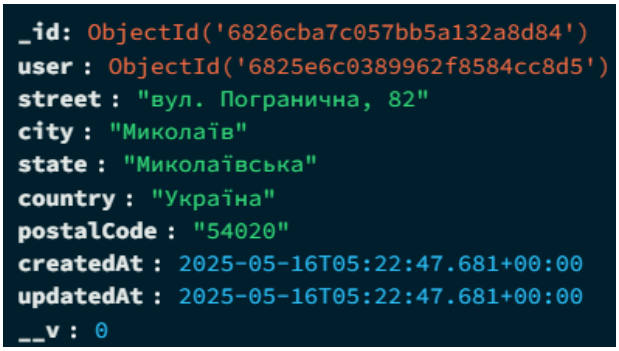

Рисунок 3.16 – Модель адреси у базі даних

Модель Review призначена для зберігання відгуків користувачів про товари, представлені в інтернет-магазині. Вона містить ключові поля, такі як ім’я користувача, текстовий опис відгуку, унікальний ідентифікатор відповідного товару, а також рейтинг, що зазвичай виражається у вигляді числової оцінки.

Ця модель відіграє важливу роль у формуванні довіри до товарів, оскільки дозволяє потенційним покупцям ознайомитися з досвідом інших користувачів. Крім того, зібрані відгуки можуть бути використані адміністрацією для покращення якості продукції, обслуговування та асортименту магазину.

На рис. 3.17 наведено модель відгуку у базі даних.

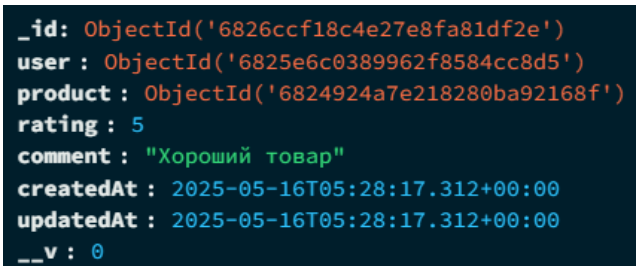

Рисунок 3.17 – Модель відгуку у базі даних

Усі ці моделі, як і ті, що були розглянуті раніше, побудовані на основі схем Mongoose, які визначають чітку структуру збережених даних.

Побудова кінцевих точок для інтернет-магазину відбувається за схожою логікою, як і для адміністративної частини, вони приймають запити від клієнтської сторони та виконують необхідні операції з базою даних для забезпечення коректної роботи застосунку.

На додачу до базових, було створено низку додаткових точок, що забезпечують виконання окремих завдань, характерних для роботи інтернет-магазину. Деякі з них:

– cart реалізує функціонал, пов’язаний з керуванням кошиком користувача: додавання товарів, їх оновлення чи видалення, а також перегляд актуального вмісту кошика;

– recommendations керує системою персоналізованих рекомендацій. Через ці кінцеві точки можна отримати загальні рекомендації, рекомендації за категорією товарів, а також на основі переглянутих раніше товарів;

– reviews відповідає за взаємодію з користувацькими відгуками, дозволяє залишати нові коментарі, видаляти наявні та отримувати перелік відгуків для відображення на сторінці товару;

– address надає можливість зберігати або редагувати адресу доставки, яка використовується під час оформлення замовлень;

– checkout відповідає за обробку процесу завершення покупки. За її допомогою користувач може підтвердити своє замовлення та перейти до оплати;

– search обробляє пошукові запити користувача. Забезпечує можливість загального пошуку по магазину, отримання пошукових підказок та доступних фільтрів для уточнення результатів.

Використання окремих кінцевих точок для кожної дії дозволяє розподілити функціональність і підтримувати чітку організацію застосунку, підвищувати масштабованість системи та спрощувати як розробку, так і подальшу підтримку. Такий підхід також полегшує інтеграцію з іншими сервісами, покращує читабельність API та сприяє повторному використанню функціональності.

3.2.2 Розробка клієнтської частини

Сучасний онлайн-магазин - це не лише потужний та функціональний інструмент для реалізації товарів, а й ключовий елемент побудови бренду. Він відіграє важливу роль у формуванні першого враження, залученні нових користувачів і підтримці лояльності постійних клієнтів. Саме тому при розробці фронтенду Інтернет-магазину особливу увагу було приділено створенню візуально привабливого, адаптивного та інтуїтивно зрозумілого інтерфейсу, який значно спрощує процес покупки для кінцевих користувачів.

Оскільки головна мета проєкту полягає у забезпеченні комфортного, швидкого та ефективного середовища для онлайн-шопінгу, важливими складовими

стали ретельно продумане розташування елементів, естетичне оформлення, а також загальна зручність взаємодії з сайтом. Усі ці аспекти були враховані під час проєктування та реалізації клієнтської частини застосунку.

Фронтенд-розробка включала побудову структури сайту, створення макетів та дизайну інтерфейсу користувача, а також реалізацію функціональності за допомогою сучасних технологій. Це охоплює обробку дій користувача, динамічне відображення даних із сервера, інтеграцію з API та забезпечення інтерактивної взаємодії на сторінці.

Для розробки клієнтської частини було використано фреймворк React.js, що забезпечив модульність, гнучкість і високу продуктивність. Порівняно з адміністративною панеллю, фронтенд Інтернет-магазину потребував набагато більшої уваги до UI/UX-дизайну, анімацій, візуального стилю та загальної взаємодії з користувачем. Це дозволило створити привабливий, сучасний інтерфейс, який відповідає очікуванням онлайн-покупців.

Окремо було зосереджено увагу на оптимізації продуктивності сайту. Було впроваджено ефективні підходи до рендерингу компонентів, зменшено обсяг завантажуваних ресурсів, оптимізовано зображення, а також реалізовано інші техніки підвищення швидкості завантаження.

Під час розробки фронтенду особливий акцент було зроблено на продуктивності та оптимізації. Зокрема, впроваджено ефективні методи рендерингу, зменшено обсяг завантажуваних ресурсів і застосовано низку технологій та практик, спрямованих на пришвидшення завантаження сторінок і покращення загальної ефективності роботи сайту.

На рис. 3.18 наведено вміст директорії components.

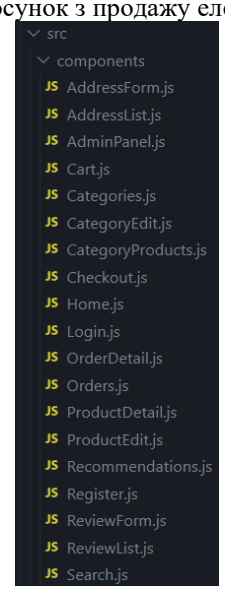

Рисунок 3.18 – Вміст директорії components

Stripe – це одне з провідних рішень у сфері онлайн-платежів, яке надає веброзробникам потужний і гнучкий інструмент для обробки та управління транзакціями. Stripe пропонує API, що дозволяє легко інтегрувати платіжні функції безпосередньо у вебзастосунки, підвищуючи рівень безпеки та спрощуючи обробку платежів.

Процес інтеграції Stripe в інтернет-магазин включає кілька основних етапів. Спершу необхідно зареєструвати обліковий запис на сайті Stripe і отримати відповідні API-ключі для взаємодії з сервісом. Потім ці ключі (див. рис. 3.19) застосовуються для використання Stripe як на стороні сервера, так і на стороні клієнта.

Рисунок 3.19 – Сторінка API ключів

На серверній стороні Stripe застосовується для створення платіжних інстанцій, які містять усю необхідну інформацію про майбутню транзакцію. Ці інстанції передаються на клієнтську сторону, де вже безпосередньо виконується процес оплати. На стороні клієнта Stripe відповідає за збір і валідацію платіжних даних користувача, зокрема даних банківської картки, та ініціацію платежу через зашифроване з'єднання для забезпечення безпеки (див. рис. 3.20). Такий розподіл функцій між клієнтом і сервером забезпечує захищену обробку даних і відповідає сучасним вимогам до безпеки платіжних систем.

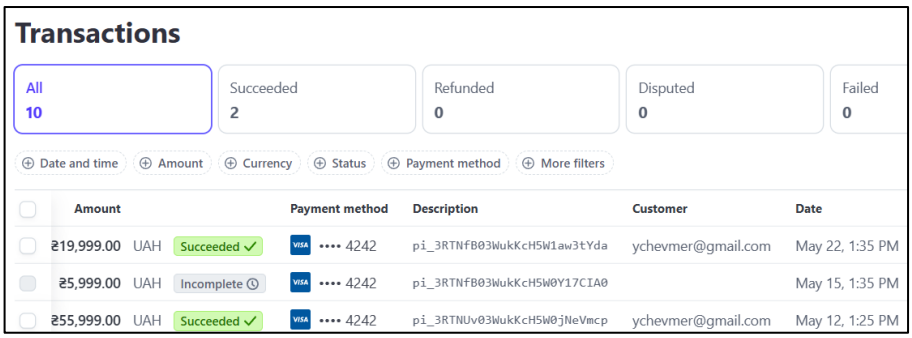

Рисунок 3.20 – Інформація про платежі

Після успішного здійснення платежу, перевірка результату відбувається на сервері за допомогою Stripe Webhook. Webhook автоматично надсилає повідомлення про статус транзакції. У разі підтвердження успішної оплати, система фіксує всі відповідні дані, зокрема, інформацію про покупця, товар/послугу та деталі платежу у базі даних. Після цього ці дані передаються до адміністративної панелі для подальшої обробки або перегляду адміністратором.

Важливим аспектом при використанні Stripe є те, що всі платіжні дані обробляються та зберігаються виключно на захищених серверах самої платформи Stripe. Це дозволяє дотримуватись високих стандартів безпеки та мінімізувати ризики витоку конфіденційної інформації.

Під час інтеграції Stripe до Інтернет-магазину розробник має змогу скористатися тестовим режимом, який дає змогу імітувати платіжні процеси без здійснення реальних фінансових операцій. Це особливо корисно на етапі розробки й перевірки функціоналу системи.

Можна створювати знижкові купони, які користувачі можуть застосовувати під час покупки для отримання вигідних пропозицій. Створення та управління такими купонами здійснюється у вкладці купонів (див. рис. 3.21). Система підтримує налаштування різних типів знижок. фіксованих сум, відсотків або безкоштовної доставки.

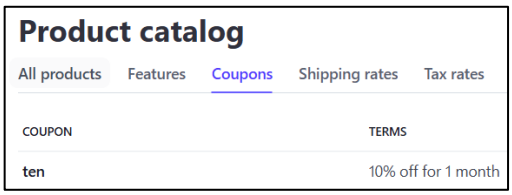

Рисунок 3.21 – Розділ купонів

Висновки до третього розділу

Описано процес створення технічної інфраструктури двох основних компонентів проєкту - онлайн-магазину та панелі адміністратора. Основний акцент було зроблено на серверній логіці, реалізовано набір API-ендпоінтів для взаємодії з базою даних і обробки запитів, що надходять від клієнтів. Важливим етапом стало підключення до MongoDB та побудова схем і моделей, що забезпечують структуровану та ефективну роботу з інформацією.

З боку клієнтського інтерфейсу реалізовано інтуїтивно зрозумілий і візуально привабливий дизайн. Для магазину окрему увагу приділено зручності навігації, адаптивності інтерфейсу та використанню анімаційних елементів для покращення загального досвіду користувача.

Окремо було опрацьовано взаємозв’язок між адміністративною панеллю та магазином - реалізовано механізми контролю, моніторингу та керування вмістом через панель адміністратора.

У підсумку реалізоване технічне рішення відображає імплементацію новітніх засобів розробки, орієнтованих на стабільність, масштабованість та комфорт користувачів.

4 ПРОГРАМНА ІМПЛЕМЕНТАЦІЯ ТА ІНСТРУКЦІЯ ДЛЯ КОРИСТУВАЧІВ

Цей розділ присвячений детальному висвітленню програмної реалізації адміністративної панелі інтернет-магазину, а також процесу розробки користувацького керівництва. У ньому надається огляд основних функціональних можливостей панелі адміністратора, зокрема інструментів управління товарами, замовленнями, користувачами. Крім того, представлено покрокові інструкції для кінцевих користувачів, що дозволяють ефективно орієнтуватися в системі та максимально використовувати її потенціал для адміністрування.

4.1 Керівництво для роботи з панеллю керування

Адміністративна панель є ключовим елементом для ефективного управління Інтернет-магазином. Вона забезпечує доступ до великого обсягу функцій, необхідних для організації та контролю основних бізнес-процесів. Зокрема, панель дозволяє керувати замовленнями, товарами, категоріями продукції, а також користувачами з правами адміністратора.

У цьому підрозділі представлено детальне керівництво з використання адміністративної панелі. Описано основні розділи інтерфейсу, порядок виконання типових операцій та надано рекомендації щодо ефективного використання доступних функціональних можливостей для забезпечення стабільної та зручної роботи системи керування.

4.1.1 Вхід до системи та стартова сторінка

Щоб розпочати використання адміністративної панелі, користувач повинен пройти процес авторизації. Для цього йому надається можливість увійти в систему за допомогою електронної пошти (див рис. 4.1).

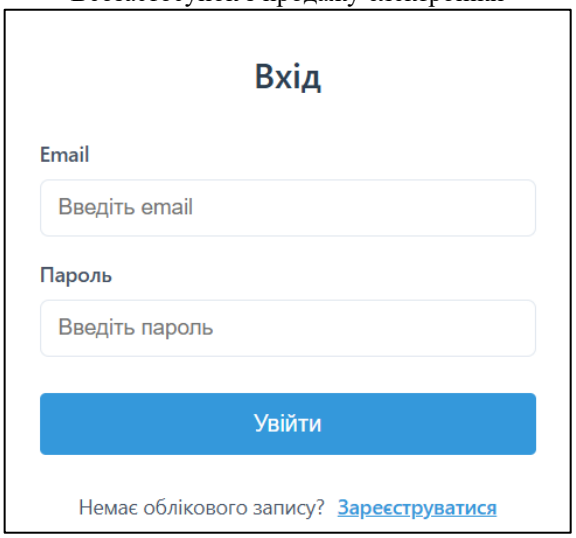

Рисунок 4.1 – Сторінка авторизації

Після заповнення форми користувач отримає електронного листа з унікальним кодом підтвердження для верифікації своєї електронної адреси (див. рис. 4.2). Цей код необхідно ввести у відповідне поле для завершення процесу.

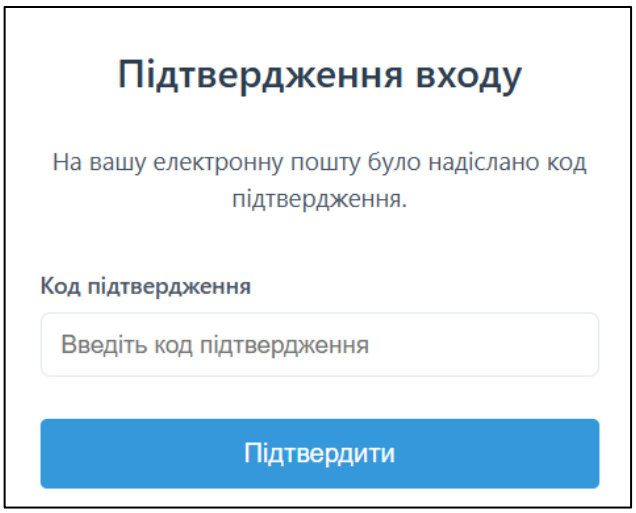

Рисунок 4.2 – Сторінка підтвердження входу

Якщо обліковий запис має права адміністратора, користувач буде перенаправлений на головну сторінку з доступом до адміністративної панелі. У протилежному випадку відобразиться повідомлення про заборону доступу через відсутність прав адміністратора, після чого користувача буде перенаправлено на головну сторінку вебзастосунку (див. рис. 4.3).

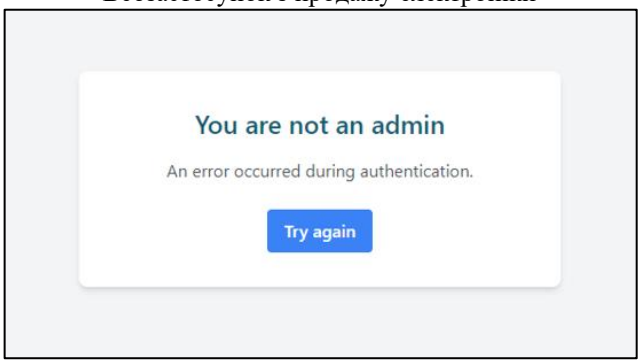

Рисунок 4.3 – Сторінка відмова доступу

Після успішної автентифікації та підтвердження особистості, користувач із правами адміністратора потрапляє на головну сторінку адміністративної панелі та секції аналітики. Цей розділ призначений для швидкого ознайомлення з основними показниками ефективності роботи магазину.

Аналітична інформація представлена у зручному для сприйняття вигляді за допомогою інтерактивних графіків і діаграм (див. рис. 4.4 та рис. 4.5).

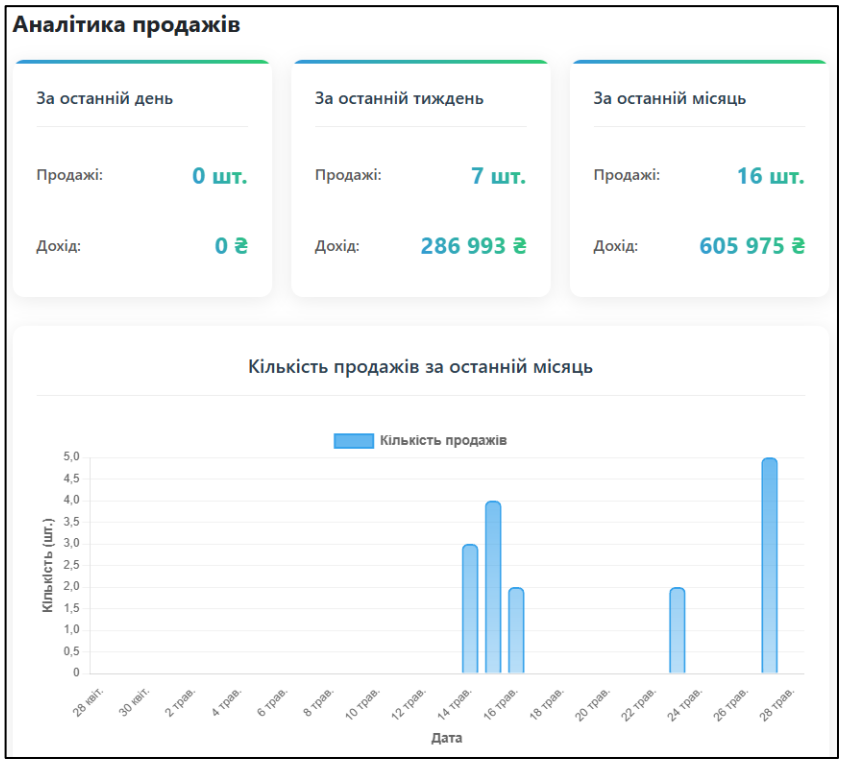

Рисунок 4.4 – Сторінка аналітики

Рисунок 4.5 – Сторінка аналітики

На цій сторінці представлено інформацію про кількість товарів, придбаних за останній день, тиждень та місяць, а також відповідні показники доходів за ці періоди. Крім того, як вже згадувалося раніше, тут розміщено графіки, які ілюструють щоденну кількість замовлень протягом місяця та динаміку доходів за аналогічний період.

4.1.2 Керування товарами та замовленнями

Ключовим елементом адміністративної панелі є управління замовленнями, який виконує центральну роль у процесі адміністрування онлайн-магазину (див. рис. 4.6). Адміністратор повний доступ до всіх замовлень, дозволяючи оперативно контролювати і керувати процесами обробки.

Детальне подання інформації, надає змогу адміністратору переглядати повний склад кожного замовлення, включаючи перелік товарів, дані про користувача (ПІБ, адресу доставки, електронну пошту та іншу контактну інформацію), статус оплати, а також час оформлення замовлення.

Ця функціональність є важливою для своєчасної обробки замовлень, планування логістичних процесів, аналізу клієнтської активності, формування звітності та своєчасного реагування на можливі проблеми чи затримки.

Рисунок 4.6 – Модуль керування замовленнями

Варто зауважити, що в аналітичних звітах щодо прибутку враховуються виключно ті замовлення, які мають статус «Оплачено», що забезпечує точність фінансових показників та запобігає викривленню статистичних даних.

Особливу увагу варто звернути на розділ керування товарами, який надає адміністратору повний контроль над асортиментом товарів, представлених у веб-магазині (див. рис. 4.7).

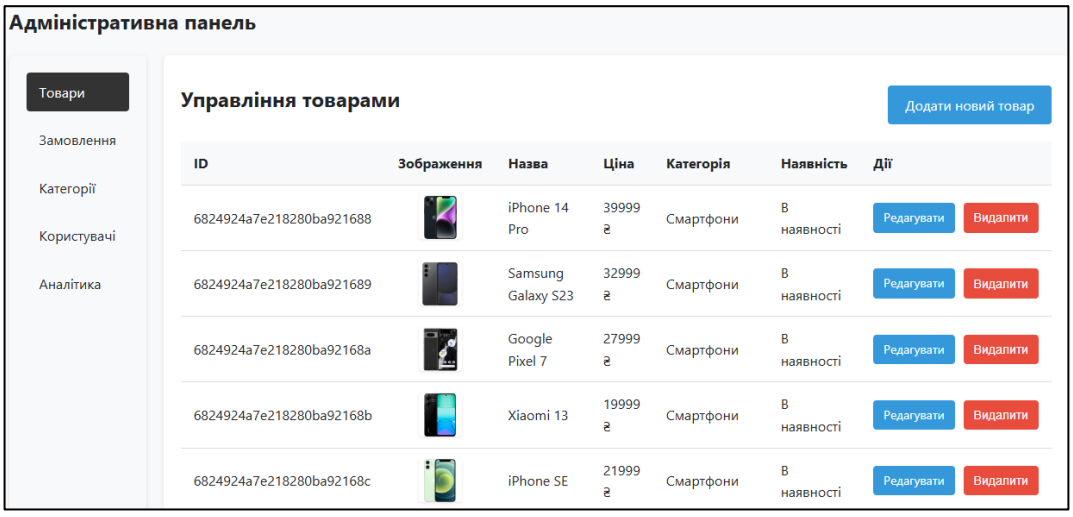

Рисунок 4.7 – Перелік товарів

Функція додавання нових товарів дозволяє адміністратору створювати нові позиції в каталозі, заповнюючи все необхідне, зокрема найменування, опис, вартість, фото та категорію товару (див. рис. 4.8).

Крім можливості додавати нові товари, адміністратор також має доступ до перегляду та редагування вже існуючих продуктів (див. рис. 4.9). Товари, що наразі представлені, відображаються в цьому розділі у вигляді зручного списку.

Адміністратор може швидко знайти потрібний продукт, переглянути його деталі, такі як назва, опис, ціна, кількість на складі, категорія та зображення і при необхідності внести зміни. Це дозволяє підтримувати актуальність, оперативно оновлювати дані про товари та ефективно керувати асортиментом магазину.

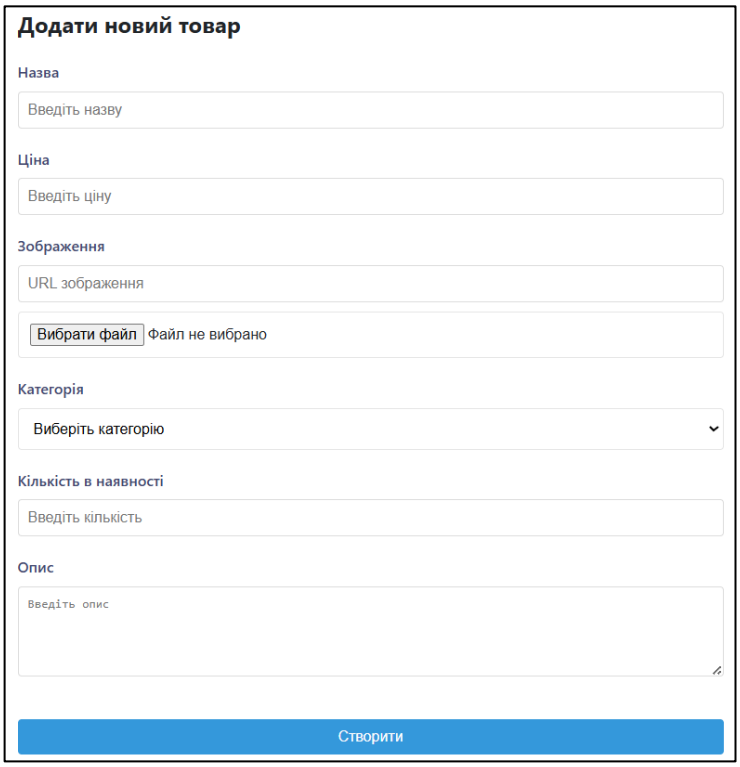

Рисунок 4.8 – Форма додавання нового товару

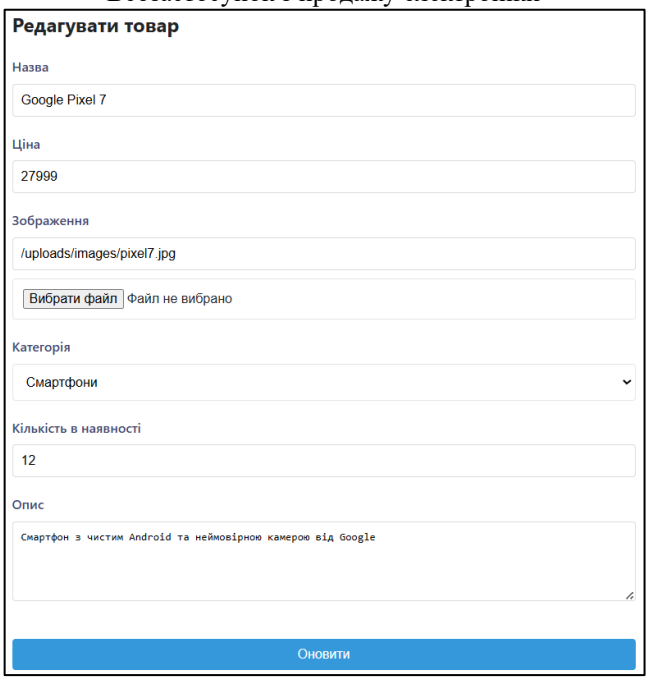

Рисунок 4.9 – Форма редагування товару

Також реалізовано важливу функцію видалення товарів. У випадках, коли певний товар більше не є актуальним, недоступний або знятий з продажу, адміністратор може скористатися функцією його повного видалення з системи.

Ця опція дозволяє підтримувати чистоту та актуальність товарного асортименту, уникати плутанини для користувачів і забезпечувати ефективне управління наповненням. Перед видаленням може реалізовано підтвердження дії, щоб уникнути випадкового видалення важливих позицій.

4.1.3 Керування категоріями та обліковими записами користувачів

Ефективне керування категоріями товарів відбувається на сторінці управління категоріями, які формують структуру контенту. Вона містить таблицю (див. рис. 4.10), де відображено перелік усіх наявних категорій із зазначенням їхніх назв, описів та пов’язаних зображень. Інтерфейс також включає форму для створення нової категорії з полями для введення назви, детального опису, завантаження зображення та кнопками для підтвердження збереження або скасування внесених змін.

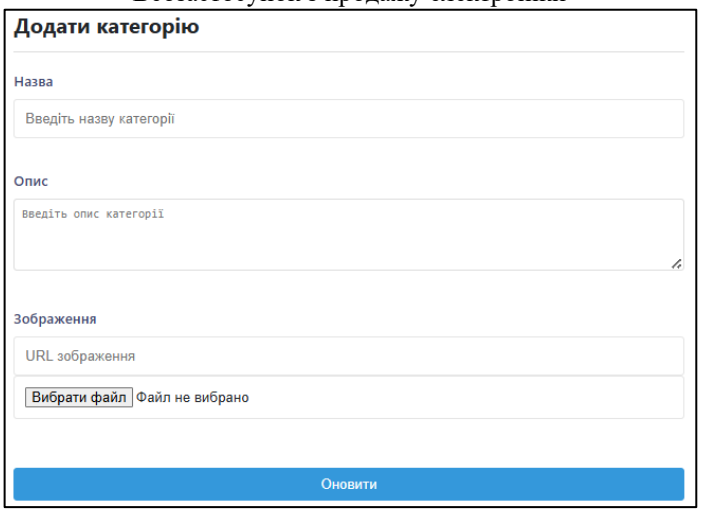

Рисунок 4.10 – Додавання нової категорії

Як і на попередній сторінці з товарами, адміністратор має змогу редагувати (див. рис. 4.11) та видаляти категорії. Крім того, реалізовано зручний інтерфейс для швидкого внесення змін, що підвищує ефективність управління структурою товарів на сайті.

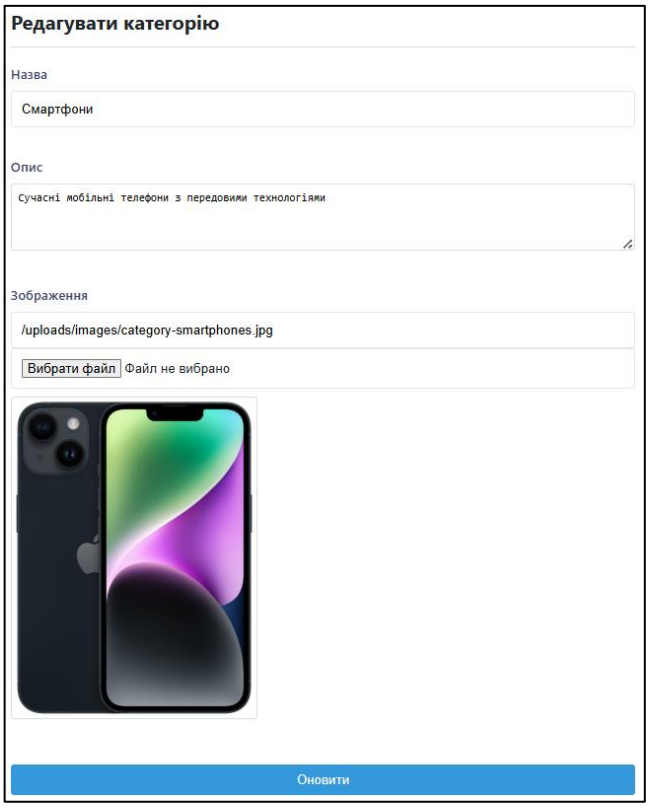

Рисунок 4.11 – Редагування існуючої категорії

Керування адміністраторами відіграє ключову роль у забезпеченні стабільної роботи та безпеки Інтернет-магазину. Відповідна секція адміністративної панелі дозволяє регулювати рівень доступу до різних функціональних частин системи (див. рис. 4.12).

У багатьох Інтернет-магазинах одночасно працює кілька адміністраторів, кожен з яких відповідає за окремий напрям: управління товарами, обробку замовлень, аналітику, технічну підтримку тощо. Функція керування адміністраторами дає змогу додавати нових учасників до адміністративної команди, а за потреби обмежувати або вилучати права доступу окремих користувачів.

Це необхідно для підтримання безпеки, підвищення ефективності та забезпечення прозорої організації роботи, адже кожному адміністратору можна надати доступ лише до тих функцій, які потрібні для виконання його професійних завдань.

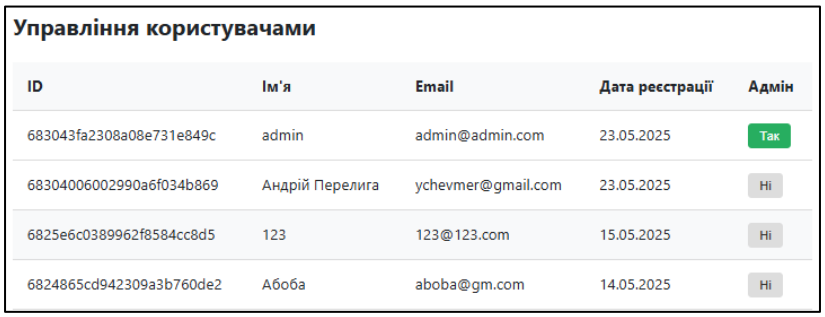

Рисунок 4.12 – Список користувачів

4.2 Керівництво користувача з користування функціоналом онлайн-магазину

Інтернет-магазин включає безліч функцій і можливостей, з якими варто ознайомитися як покупцям, так і адміністраторам. Від перегляду товарів до оформлення замовлення та здійснення оплати, кожен етап має значення для створення зручного та ефективного користувацького досвіду. У цьому розділі представлено докладний посібник, що допоможе найкращим чином користуватися Інтернет-магазином для досягнення максимальної вигоди й комфорту.

4.2.1 Головна сторінка та перегляд категорій

Головна сторінка Інтернет-магазину привертає увагу відвідувачів оригінальним і зручним дизайном. Верхня частина сторінки містить навігаційне меню, яке дозволяє користувачам легко переходити між різними розділами сайту. Під навігаційною панеллю розміщено вибрані категорії товарів, які адміністратор визначає як основні. Для кожної такої категорії подано опис, зображення та кнопку для перегляду додаткової інформації (див. рис. 4.13).

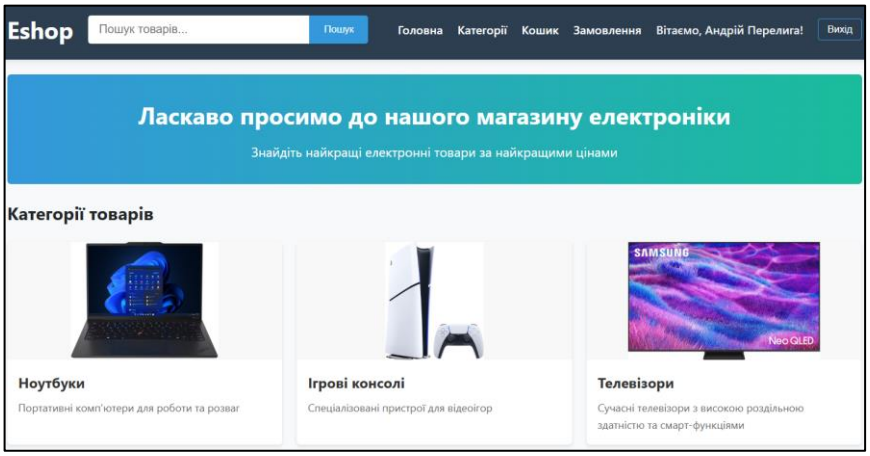

Рисунок 4.13 – Головна сторінка

Сторінка категорій має зручний та зрозумілий інтерфейс, що дає змогу користувачам легко переміщатися між різними групами товарів, об'єднаних спільними ознаками. Систематизація товарів може здійснюватися за різними критеріями, такими як призначення, спосіб використання, цінова категорія або інші властивості. Такий підхід забезпечує логічну організацію широкого асортименту продукції й спрощує покупцям процес пошуку потрібного товару.

На рис. 4.14 наведено категорії товарів.

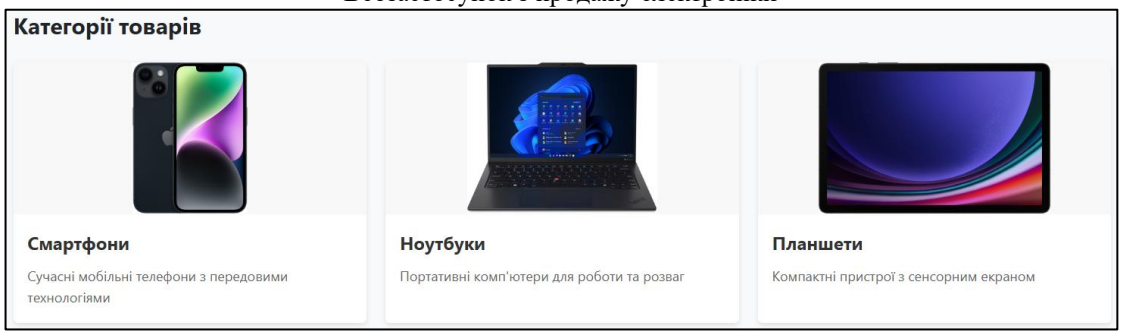

Рисунок 4.14 – Список категорій товарів

Користувач має змогу вибрати потрібну категорію, натиснувши на її зображення, після чого одразу переходить до списку товарів цієї категорії. На цій сторінці реалізована функція сортування (див. рис. 4.15), яка дозволяє впорядковувати товари за датою додавання (від нових до старих або навпаки), ціною (від нижчої до вищої або навпаки), а також за іншими параметрами (див. рис. 4.16). Це значно спрощує пошук і вибір потрібних товарів.

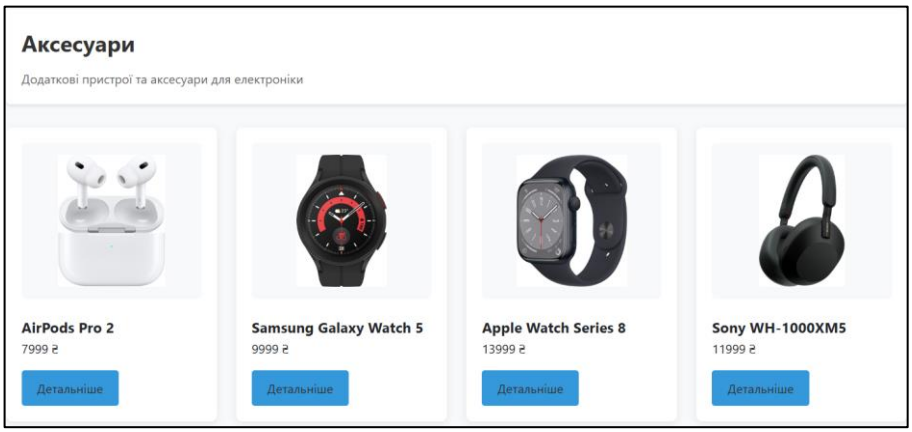

Рисунок 4.15 – Сторінка за категорією

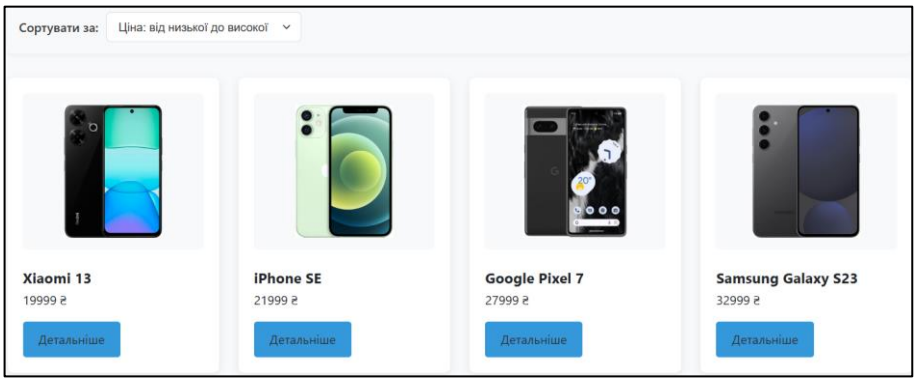

Рисунок 4.16 – Сортування товарів за ціною

4.2.2 Сторінка товару

Сторінка товару призначена для надання детальної інформації про конкретний товар (див. рис. 4.17). Вона містить зображення товару, його назву, опис, актуальну ціну, а також додаткову інформацію, таку як наявність, характеристики або відгуки покупців. Якщо користувач зацікавлений у придбанні товару, він може натиснути кнопку «Додати до кошика», після чого товар буде збережений у кошику для подальшого оформлення замовлення. Це дозволяє зручно керувати вибраними покупками й спрощує процес покупки.

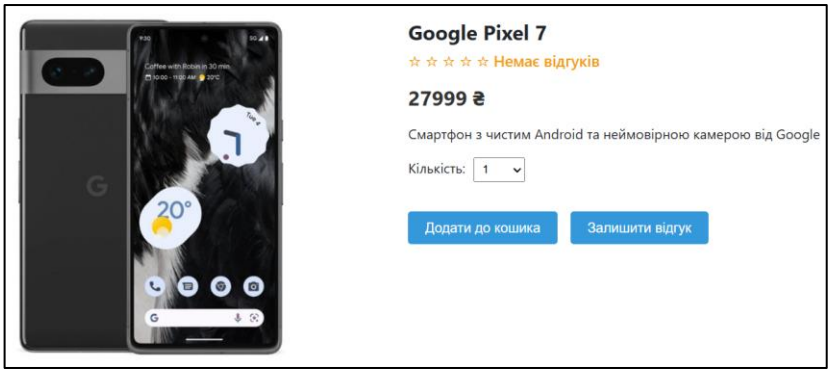

Рисунок 4.17 – Деталі товару

На цій сторінці також впроваджено алгоритм колаборативної фільтрації, який аналізує поведінку інших користувачів із подібними інтересами та формує персоналізовані рекомендації. Завдяки цьому покупцю пропонуються додаткові товари, які з великою ймовірністю можуть його зацікавити, що сприяє підвищенню рівня задоволеності клієнта та збільшенню обсягу продажів (див. рис. 4.18).

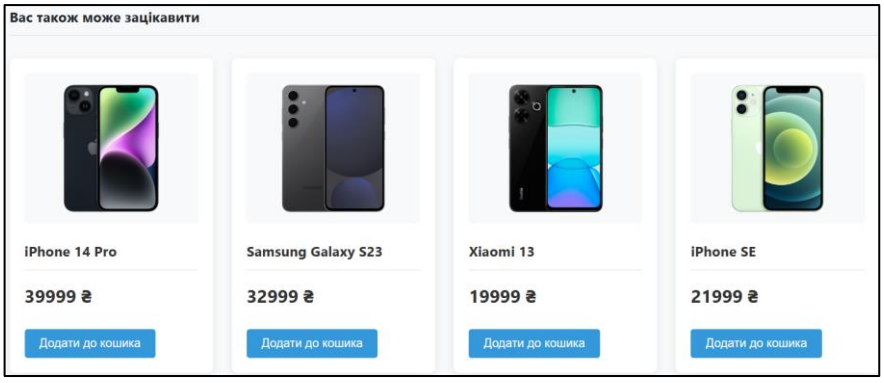

Рисунок 4.18 – Персоналізовані рекомендації

Головною відмінною особливістю даної сторінки є вбудовані відгуки покупців, які значно впливають на довіру до товару. У цьому розділі відображається середній рейтинг у вигляді зірок, а також текстові коментарі від покупців, які вже здійснили покупку та поділилися своїми враженнями. Крім того, кожен відвідувач сторінки має можливість залишити власний відгук, поставити оцінку та детально описати свій досвід використання товару. Така функціональність сприяє більш обґрунтованому прийняттю рішень іншими покупцями та підвищує рівень взаємодії з платформою (див. рис. 4.19).

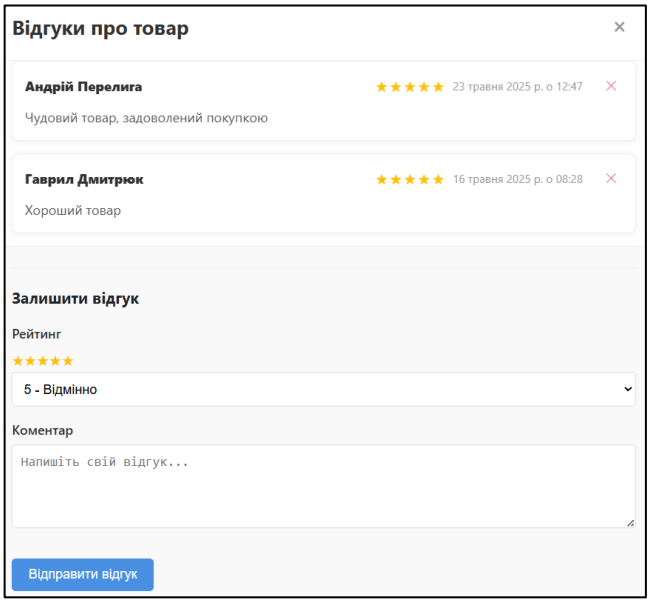

Рисунок 4.19 – Відгуки про товар

У випадку, якщо для товару ще не залишено жодного відгуку, користувачеві буде показано повідомлення про відсутність відгуків (див. рис. 4.20).

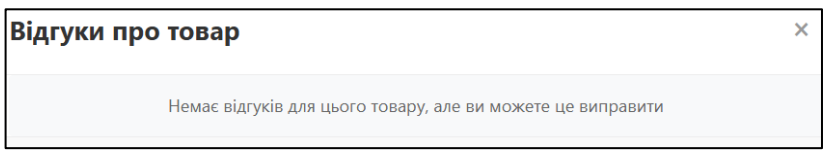

Рисунок 4.20 – Відсутні відгуки

4.2.3 Кошик товарів

На сторінці кошика користувач має змогу переглядати всі вибрані для покупки товари (див. рис. 4.21), а також редагувати їх. Зокрема, можна змінити кількість одиниць товару, видалити непотрібні позиції або повернутися до подальшого вибору продукції. Усі ці параметри доступні для редагування до моменту остаточного підтвердження замовлення.

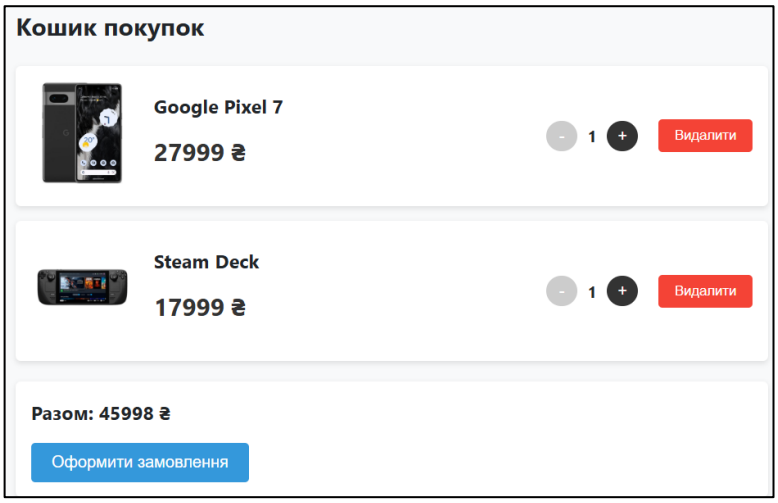

Рисунок 4.21 – Кошик з обраними товарами

Якщо кошик порожній, на сторінці відобразиться відповідне повідомлення для користувача (див. рис. 4.22).

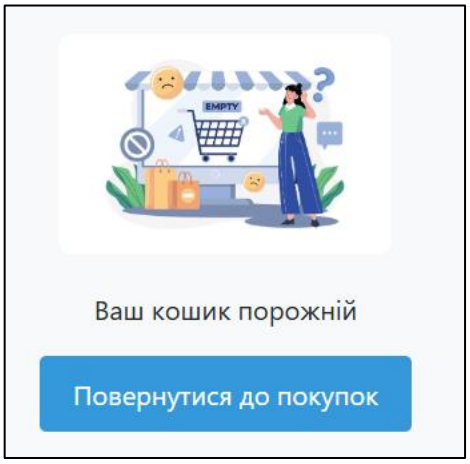

Рисунок 4.22 – Порожній кошик

Крім того, система аналізує категорії товарів, доданих до кошика, і пропонує випадковий продукт із відповідної категорії. Це може зацікавити користувача та спонукати його до додаткових покупок, що, у свою чергу, сприятиме збільшенню прибутку магазину (див. рис. 4.23).

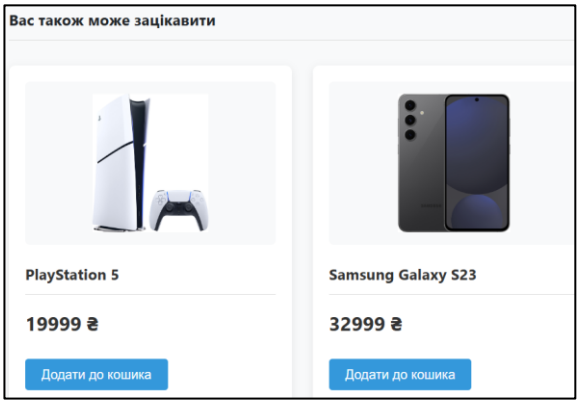

Рисунок 4.23 – Рекомендовані товари

Цю секцію розміщено саме в кошику, щоб користувач, переглядаючи вже обрані товари, мав змогу побачити додаткові корисні пропозиції та, за бажанням, додати їх до покупки.

4.2.4 Сторінка оформлення платежу та механізм пошуку

На сторінці оплати користувач має можливість підтвердити оформлення замовлення, а також переглянути його остаточну вартість з урахуванням податків і вартості доставки (рис. 4.24). Це дозволяє оцінити загальну суму до сплати перед завершенням покупки.

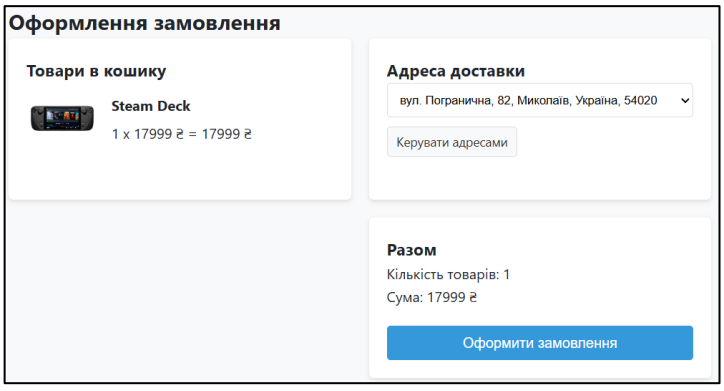

Рисунок 4.24 – Оформлення замовлення

Після цього користувач може перевірити всі деталі замовлення, включно з переліком товарів, цінами, адресою доставки(рис. 4.25).

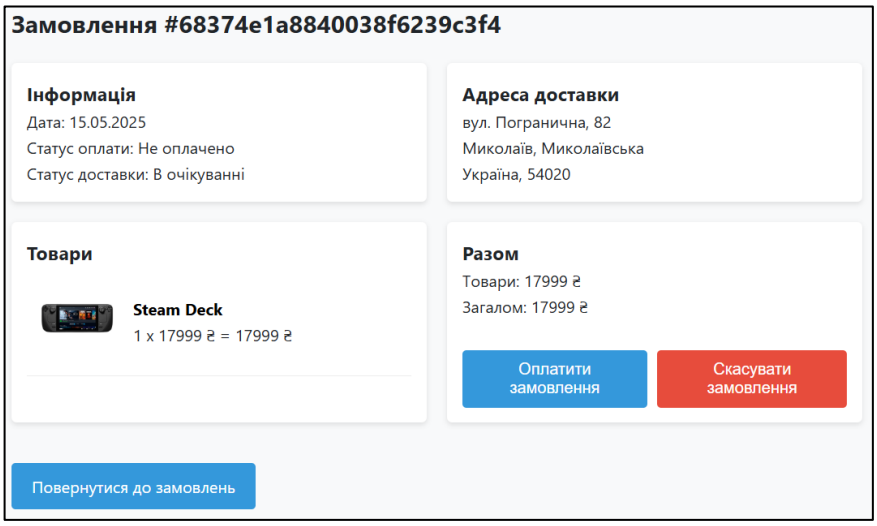

Рисунок 4.25 – Деталі замовлення

Після завершення перевірки введених даних користувач переходить до етапу оплати, де може скористатися карткою Visa або Mastercard, електронною платіжною системою чи альтернативними варіантами, що підтримуються сервісом (див. рис. 4.26).

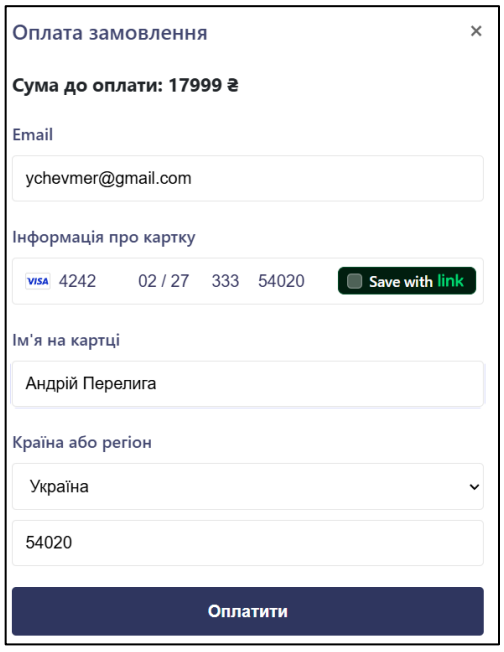

Рисунок 4.26 – Оплата замовлення

Пошукова функція є важливою складовою будь-якого сайту електронної комерції та надає змогу користувачам швидко та зручно орієнтуватися у магазині й знаходити необхідні товари.

Користувач вводить запит у поле пошуку, використовуючи ключові слова, що можуть відображати назву або властивості товару. На окремій ділянці сторінки з'являються відповідні результати з коротким описом, фото й ціною (див. рис. 4.27).

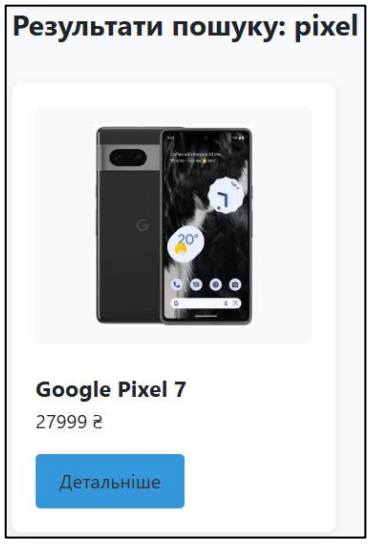

Рисунок 4.27 – Результати успішного пошуку товару

У разі відсутності результатів пошуку система повідомляє користувача про те, що товар не знайдено (див. рис. 4.28).

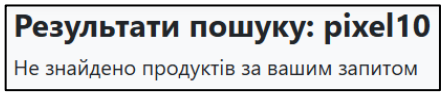

Рисунок 4.28 – Приклад відсутності результатів пошуку

4.2.5 Тестування розробленого вебзастосунку

Тестування вебзастосунку є важливим етапом оцінки його якості, зокрема таких аспектів, як ефективність, доступність, використання оптимальних методів розробки та відповідність пошукової оптимізації. Це дозволяє виявити проблеми ще до впровадження системи в експлуатацію, що знижує ризики виникнення

помилок у майбутньому. У ході тестування було проаналізовано ключові характеристики, результати представлені у вигляді діаграми (див. рис. 4.29).

Основні критерії оцінювання:

– ефективність: оцінює здатність вебзастосунку швидко та стабільно виконувати користувацькі запити;

– доступність: перевірка відповідності стандартам доступності, зокрема наявність альтернативного тексту, правильна семантика HTML-структур та підтримка клавіатурної навігації;

– оптимальні методи: включає використання сучасних підходів до розробки, таких як модульна структура коду, адаптивна верстка, асинхронна передача даних та використання CDN;

– оптимізація під пошукові системи: визначає рівень підготовленості сайту до SEO, зокрема наявність коректних мета-тегів, оптимізацію швидкості завантаження, структуру URL та мобільну адаптивність.

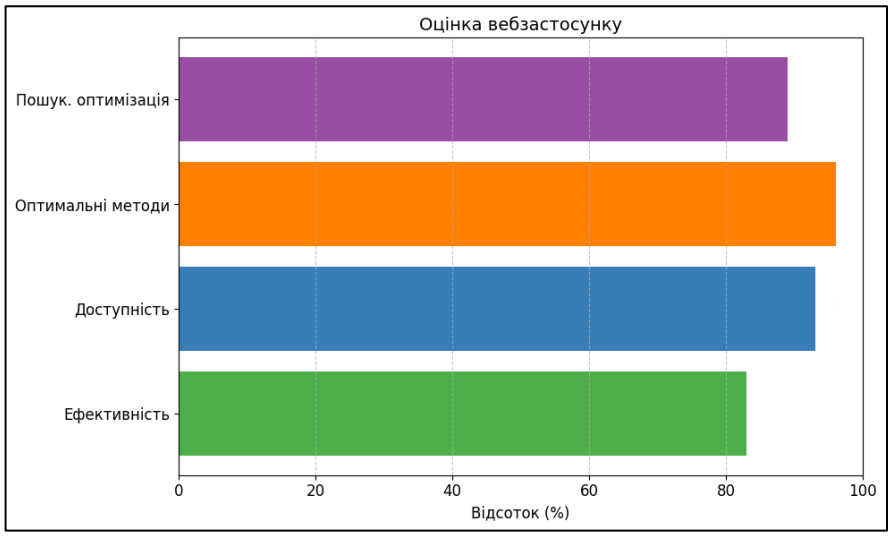

Рисунок 4.29 – Оцінка вебзастосунку

За оцінкою, застосунок демонструє високий рівень доступності та використання оптимальних методів. Водночас, пошукова оптимізація та ефективність потребують подальшого вдосконалення для успішного розвитку й масштабування проєкту.

Також створено діаграму, що наочно ілюструє, як різні чинники впливають на продуктивність вебзастосунку (див. рис. 4.30).

Рисунок 4.30 – Оцінка вебзастосунку

Оцінка факторів показує, що найбільший вплив на швидкодію вебзастосунку має завантаження ресурсів, яке становить більшість загального часу завантаження, зазвичай через великі зображення, неефективні шрифти та важкі JavaScript файли. Оптимізація коду посідає друге місце, оскільки надлишковий або непотрібний код може суттєво ускладнювати рендеринг і сповільнювати взаємодію з користувачем. Відповідь сервера визначає початкову затримку перед завантаженням, тоді як рендеринг сторінки залежить від складності DOM і навантаження на основний потік. Хоча кешування та зовнішні ресурси мають менший вплив, їхнє правильне використання дозволяє уникнути надмірних запитів і залежності від сторонніх серверів, що позитивно впливає на стабільність і загальну продуктивність вебзастосунку.

Висновки до четвертого розділу

Підсумовуючи, можна зробити висновок про важливість і користь створення такого вебзастосунку для продажу електроніки. Основні можливості сайту були докладно розглянуті, включаючи функціонал головної сторінки, сторінок

продуктів, окремих товарів та категорій, кошик, оплату та пошук. Кожен із цих розділів було ретельно проаналізовано, вказуючи, у яких випадках вони використовуються.

Завдяки ефективності системи користувачі мають змогу швидко і без зусиль знаходити необхідні позиції, контролювати їх наявність і ціни, зберігати вибрані позиції, а також зручно керувати покупками й замовленнями. Окрім того, простота інтерфейсу та зручність у використанні роблять цей застосунок зручним і приємним для користувачів і дають йому переваги перед багатьма іншими платформами електронної комерції.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи розроблено вебзастосунок, орієнтований на оптимізацію процесів онлайн-продажу електроніки. Проведено глибокий аналіз особливостей предметної області електронної комерції, розглянуто сучасні підходи до розробки вебзастосунків та проаналізовано існуючі аналоги в цій галузі.

У ході реалізації проєкту було застосовано низку інформаційних технологій, сервісів і бібліотек для вирішення поставлених завдань. Надано опис обраного середовища розробки, особливостей використання технологічного стеку, а також застосованих методів, фреймворків і сервісів.

У межах роботи реалізовано функціональні компоненти вебзастосунку, включаючи адміністративну панель та Інтернет-магазин. Детально описано технічну реалізацію як серверної, так і клієнтської частин, налаштування бази даних та інтеграцію системи електронної авторизації. Також підготовлено інструкцію користувача для обох частин застосунку, де детально розглянуто ключові функціональні аспекти системи.

Вебзастосунок пройшов тестування на відповідність функціональним вимогам, було проаналізовано перспективи розширення та підвищення ефективності системи. Поставлені цілі були успішно досягнуті, що свідчить про ефективність виконаної роботи.

Здобуті результати відкривають можливості для подальшого удосконалення проєкту, його масштабного впровадження, а також становлять підґрунтя для подальших досліджень. Накопичений практичний досвід при створенні вебзастосунку стане цінним ресурсом для розробки аналогічних рішень у майбутньому.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Монолітна архітектура ПЗ. QALight. URL: https://qalight.ua/baza-znaniy/shho-take-monolitna-arhitektura/ (дата звернення: 01.05.2025).

2. Мікросервісна архітектура: основні концепції та виклики. FoxmindEd. URL: https://foxminded.ua/mikroservisna-arkhitektura/ (дата звернення: 01.05.2025).

3. SPA (Single-page application) - MDN Web Docs Glossary. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA (дата звернення: 01.05.2025).

4. Український ринок e-commerce: як він змінився та що його очікує в майбутньому. Асоціація ритейлерів України. URL: https://rau.ua/novyni/ukrainskij-rinok-e-commerce/ (дата звернення: 01.05.2025).

5. Копишинський Ю. Ситуація з українським ринком електронної комерції. Forbes.ua. URL: https://surl.li/dpgzzp (дата звернення: 01.05.2025).

6. Інтернет-магазин Rozetka. URL: https://rozetka.com.ua/ (дата звернення: 01.05.2025).

7. Інтернет-магазин Allo. URL: https://allo.ua/ (дата звернення: 01.05.2025).

8. Інтернет-магазин Comfy. URL: https://comfy.ua/ (дата звернення: 01.05.2025).

9. IDE VS Code. URL: https://code.visualstudio.com/ (дата звернення: 01.05.2025).

10. Mehta, H. MongoDB for Beginners: Learn Basics of MongoDB in a Simple Way. BPB Publications, 2020. 180 с.

11. What is ExpressJS. EDUCBA. URL: https://www.educba.com/what-is-expressjs/ (дата звернення: 01.05.2025).

12. Griffiths, A. React Cookbook: Recipes for Mastering the React Framework. Packt Publishing, 2020. 350 с.

13. Core Principles of React JS and its Concepts. Pattem Digital. URL: https://pattemdigital.com/insight/core-principles-of-react-js-concepts/ (дата звернення: 01.05.2025).

14. Cantelon, M., & Harter, T. Node.js in Action. 1st ed. Greenwich, CT, USA: Manning Publications, 2013. 368 с.

15. Node.js Frameworks. Devabit. URL: https://devabit.com/blog/node-js-frameworks/ (дата звернення: 01.05.2025).

16. Houssein, I. Next.js: The React Framework for Production. 1st ed. Cham, Switzerland: Springer, 2021. 310 с.

17. Habitation C. Rendering Techniques for Next.js:. Medium. URL: https://surli.cc/imdccc (дата звернення: 01.05.2025).

18. React-Toastify: Setup, styling & real-world use cases. LogRocket Blog. URL: https://blog.logrocket.com/react-toastify-guide/ (дата звернення: 01.05.2025).

19. Stripe & JS: Payments Integration. RisingStack. URL: https://surli.cc/coejtc (дата звернення: 01.05.2025).

20. Chart.js. Chart.js | Open source HTML5 Charts for your website. URL: https://www.chartjs.org/ (дата звернення: 01.05.2025).

21. SuperTokens. What is JWT? Understand JSON Web Tokens. Open Source User Authentication. URL: https://supertokens.com/blog/what-is-jwt (date of access: 01.05.2025).

22. Singh C. How to Handle Passwords Safely. DigitalOcean. URL: https://surl.li/ankfnr/ (дата звернення: 01.05.2025).

23. Samarasinghe A. EmailJS: Send emails directly from your client-side code. Medium. URL: https://surl.li/zwtvqn (дата звернення: 01.05.2025).

24. How to use Recharts to visualize analytics data. PostHog. URL: https://posthog.com/tutorials/recharts (дата звернення: 01.05.2025).

25. Express Tutorial: Using a Database (with Mongoose) - Learn web development. MDN Web Docs. URL: https://surl.lu/akgeoa (дата звернення: 01.05.2025).

26. freeCodeCamp. How to Lazy Load Images in React. freeCodeCamp.org. URL: https://surl.li/jwusrb (дата звернення: 01.05.2025).

27. Draggable JS – JavaScript drag and drop library. Shopify. URL: https://shopify.github.io/draggable/ (дата звернення: 01.05.2025).

28. Ваше повне керівництво по Tailwind CSS. DreamHost Blog. URL: https://www.dreamhost.com/blog/uk/tailwind-css/ (дата звернення: 01.05.2025).

29. Getting Started | Axios Docs. Axios. URL: https://axios-http.com/docs/intro (дата звернення: 01.05.2025).

30. Найкращі практики проєктування API. DevZone. URL: https://surl.li/eblpox (дата звернення: 01.05.2025).

ДОДАТОК А

Лістинг програмного коду

Home.js

```
import React, { useState, useEffect } from 'react';
import { Link } from 'react-router-dom';
import axios from 'axios';

function Home() {
  const [categories, setCategories] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchCategories = async () => {
      try {
        const { data } = await axios.get('/api/categories');
        setCategories(data);
        setLoading(false);
      } catch (err) {
        setError('Помилка завантаження категорій');
        setLoading(false);
      }
    };
    fetchCategories();}, []);

  if (loading) return <div>Завантаження...</div>;
  if (error) return <div className="error">{error}</div>;

  return (
    <div className="home-page">
      <div className="banner">
        <h1>Вітаємо в нашому магазині електроніки</h1>
        <p>Знайдіть найкращі електронні товари за найкращими
цінами</p>
      </div>
      <h2>Категорії</h2>
      <div className="categories-grid">
        {categories.length === 0 ? (
          <p>Немає доступних категорій</p>) : (
          categories.map(category => (
            <div key={category._id} className="category-card">
              <Link to={`/category/${category._id}`}>
                <div className="category-image">
                  <img
                    src={category.image}
                    alt={category.name}
                    width="300"
                    height="160"
                    style={{ aspectRatio: 'auto' }}
```

```
                    fetchpriority={categories.indexOf(category) < 2 ?
"high" : "auto"}/>
                </div>
                <div className="category-info">
                  <h3>{category.name}</h3>
                  <p>{category.description}</p>
                </div>
              </Link>
            </div>
          ))
        )}
      </div>
    </div>
  );
}
export default Home;

Login.js
import React, { useState, useEffect } from 'react';
import { Link, useNavigate, useLocation } from 'react-router-dom';
import axios from 'axios';
function Login({ setUserInfo }) {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const      [isVerificationStep,      setIsVerificationStep]      =
useState(false);
  const [userId, setUserId] = useState('');
  const [otp, setOtp] = useState('');
  const navigate = useNavigate();
  const location = useLocation();
  const                redirect                =                new
URLSearchParams(location.search).get('redirect') || '/';
  useEffect(() => {
    const userInfoFromStorage = localStorage.getItem('userInfo');
    if (userInfoFromStorage) {
      navigate(redirect);}
  }, [navigate, redirect]);
  const submitHandler = async (e) => {
    e.preventDefault();
    try {
      const { data } = await axios.post('/api/auth/login', { email,
password });
      setUserId(data.userId);
      setIsVerificationStep(true);
      setError('');
    } catch (error) {
      setError(
        error.response && error.response.data.message
          ? error.response.data.message
          : 'Невдала спроба входу'
      );
```

```
    }
  };
  const verifyOtpHandler = async (e) => {
    e.preventDefault();
    try {
      const { data } = await axios.post('/api/auth/verify-otp', {
        userId,
        otp
      });
      if (data && data.user && data.token) {
        const userToSave = {
          _id: data.user._id,
          name: data.user.name,
          email: data.user.email,
          isAdmin: data.user.isAdmin,
          token: data.token
        };
        localStorage.setItem('userInfo', JSON.stringify(userToSave));
        setUserInfo(userToSave);
        navigate(redirect);
      } else {
        throw new Error('Отримано некоректні дані користувача від
сервера');}} catch (error) {
      setError(
        error.response && error.response.data.message
          ? error.response.data.message
          : error.message || 'Помилка підтвердження коду'
      );
    }
  };
  return (
    <div className="login-container">
      <h2>{isVerificationStep ? 'Підтвердження входу' : 'Вхід'}</h2>
      {error && <div className="error">{error}</div>}
      {!isVerificationStep ? (
      <form onSubmit={submitHandler} className="form">
        <div className="form-group">
          <label htmlFor="email">Email</label>
          <input
            type="email"
            id="email"
            placeholder="Введіть email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            required/>
        </div>
        <div className="form-group">
          <label htmlFor="password">Пароль</label>
          <input
            type="password"
            id="password"
            placeholder="Введіть пароль"
```

```
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            required/>
        </div>
        <button type="submit" className="btn btn-block">
          Увійти
        </button>
      </form>) : (
        <form onSubmit={verifyOtpHandler} className="form">
          <p>На вашу електронну пошту було надіслано код
підтвердження.</p>
          <div className="form-group">
            <label htmlFor="otp">Код підтвердження</label>
            <input
              type="text"
              id="otp"
              placeholder="Введіть код підтвердження"
              value={otp}
              onChange={(e) => setOtp(e.target.value)}
              required/>
          </div>
          <button type="submit" className="btn btn-block">
            Підтвердити
          </button>
        </form>)}
      {!isVerificationStep && (
      <div className="register-link">
        Немає облікового запису?{' '}
        <Link to={redirect !== '/' ? `/register?redirect=${redirect}`
: '/register'}>Зареєструватися
        </Link>
      </div>
      )}
    </div>
  );
}
export default Login;

Categories.js
import React, { useState, useEffect } from 'react';
import { Link } from 'react-router-dom';
import axios from 'axios';
function Categories() {
  const [categories, setCategories] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  useEffect(() => {
    async function fetchCategories() {
      try {
        const { data } = await axios.get('/api/categories');
        setCategories(data);
        setLoading(false);
```

```
      } catch (err) {
        setError('Помилка завантаження категорій');
        setLoading(false);}}
    fetchCategories();}, []);
  if (loading) return <div>Завантаження...</div>;
  if (error) return <div className="error">{error}</div>;
  return (
    <div className="categories-page">
      <h2>Категорії товарів</h2>
      <div className="categories-grid">
        {categories.length === 0 ? (
          <p>Немає доступних категорій</p>) : (
          categories.map(category => (
            <div key={category._id} className="category-card">
              <Link to={`/category/${category._id}`}>
                <div className="category-image">
                  <img src={category.image} alt={category.name} />
                </div>
                <div className="category-info">
                  <h3>{category.name}</h3>
                  <p>{category.description}</p>
                </div>
              </Link>
            </div>
          ))
        )}
      </div>
    </div>
  );
}
export default Categories;
```

ProductDetail.js

```
import React, { useState, useEffect } from 'react';
import { Link, useParams, useNavigate } from 'react-router-dom';
import { useDispatch } from 'react-redux';
import axios from 'axios';
import { addToCart } from '../store/cartSlice';
import ReviewModal from './ReviewModal';
function ProductDetail() {
  const { id } = useParams();
  const navigate = useNavigate();
  const dispatch = useDispatch();
  const [product, setProduct] = useState(null);
  const [relatedProducts, setRelatedProducts] = useState([]);
  const [qty, setQty] = useState(1);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [reviews, setReviews] = useState([]);
  const [reviewsLoading, setReviewsLoading] = useState(true);
  const [showReviewModal, setShowReviewModal] = useState(false);
  const [averageRating, setAverageRating] = useState(0);
```

```
  const [reviewCount, setReviewCount] = useState(0);
  useEffect(() => {
    const fetchProduct = async () => {
      try {
        const { data } = await axios.get(`/api/products/${id}`);
        setProduct(data);
        setLoading(false);
        const { data: relatedData } = await
axios.get(`/api/products/category/${data.category._id}`);
        setRelatedProducts(relatedData.filter(p => p._id !==
id).slice(0, 4));
      } catch (err) {
        setError('Помилка завантаження товару');
        setLoading(false);}};
    const fetchReviews = async () => {
      try {
        const { data } = await
axios.get(`/api/reviews/product/${id}`);
        setReviews(data);
        setReviewsLoading(false);

        if (data.length > 0) {
          const totalRating = data.reduce((sum, review) => sum +
review.rating, 0);
          setAverageRating(totalRating / data.length);
          setReviewCount(data.length);
        }
      } catch (err) {
        setReviewsLoading(false);}};
    fetchProduct();
    fetchReviews();}, [id]);
  const addToCartHandler = () => {
    if (!product) return;
    dispatch(addToCart({
      product: product._id,
      name: product.name,
      image: product.image,
      price: product.price,
      countInStock: product.countInStock,
      qty}));
    navigate('/cart');};
  const handleReviewAdded = (newReview) => {
    const updatedReviews = [newReview, ...reviews];
    setReviews(updatedReviews);
    const totalRating = updatedReviews.reduce((sum, review) => sum +
review.rating, 0);
    setAverageRating(totalRating / updatedReviews.length);
    setReviewCount(updatedReviews.length);};
  const handleReviewDeleted = (reviewId) => {
    const updatedReviews = reviews.filter(review => review._id !==
reviewId);
    setReviews(updatedReviews);
```

```
    if (updatedReviews.length > 0) {
      const totalRating = updatedReviews.reduce((sum, review) => sum
+ review.rating, 0);
      setAverageRating(totalRating / updatedReviews.length);
      setReviewCount(updatedReviews.length);
    } else {
      setAverageRating(0);
      setReviewCount(0);}};

  const renderStars = (rating) => {
    const stars = [];
    for (let i = 1; i <= 5; i++) {
      if (i <= rating) {
        stars.push(<span key={i} className="star-filled">★</span>);
      } else {
        stars.push(<span key={i} className="star-empty">☆</span>);
      }
    }
    return stars;};
  if (loading) return <div>Завантаження...</div>;
  if (error) return <div>{error}</div>;
  if (!product) return <div>Товар не знайдено</div>;

  return (
    <div className="product-page">
      <div className="product-detail">
        <div className="product-image">
          <img         src={product.image}         alt={product.name}
fetchpriority="high"  width="400"  height="400"  style={{  aspectRatio:
'auto' }} />
        </div>
        <div className="product-info">
          <h2>{product.name}</h2>

          <div     className="product-rating"     onClick={()      =>
setShowReviewModal(true)}>
            <div className="rating-stars">
              {renderStars(Math.round(averageRating))}
            </div>
            <span className="review-count">
              {reviewCount > 0 ? `${reviewCount} відгуків` : 'Немає
відгуків'}
            </span>
          </div>

          <p className="price">{product.price} ₴</p>
          <p>{product.description}</p>

          <div className="qty-select">
            <label>Кількість:</label>
```

```
            <select value={qty} onChange={(e) => setQty(Number(e.target.value))}>
              {[...Array(product.countInStock).keys()].map(x => (
                <option key={x + 1} value={x + 1}>
                  {x + 1}
                </option>))}
            </select>
          </div>

          <div className="product-buttons">
          <button
            onClick={addToCartHandler}
            className="btn"
            disabled={product.countInStock === 0}>
            {product.countInStock > 0 ? 'Додати до кошика' : 'Немає в наявності'}
          </button>
            <button
              className="btn"
              onClick={() => setShowReviewModal(true)}>
              Залишити відгук
            </button>
          </div>
        </div>
      </div>
      <ReviewModal
        isOpen={showReviewModal}
        onClose={() => setShowReviewModal(false)}
        productId={id}
        initialReviews={reviews}/>
      {relatedProducts.length > 0 && (
        <div className="recommendations">
          <h3>Вас також може зацікавити</h3>
          <div className="products">
            {relatedProducts.map(product => (
              <div key={product._id} className="product">
                <Link to={`/product/${product._id}`}>
                  <img src={product.image} alt={product.name} loading="lazy" width="200" height="200" />
                  <h3>{product.name}</h3>
                  <p className="price">{product.price} ₴</p>
                </Link>
                <div className="product-actions">
                  <button
                    className="btn"
                    onClick={() => {
                      dispatch(addToCart({
                        product: product._id,
                        name: product.name,
                        image: product.image,
                        price: product.price,
                        countInStock: product.countInStock,
```

```
                        qty: 1}));
                      }}>Додати до кошика
                    </button>
                  </div>
                </div>
              ))}
            </div>
          </div>
        )}
      </div>
    );
}

export default ProductDetail;
```

Cart.js

```
import React, { useState } from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { Link, useNavigate } from 'react-router-dom';
import { removeFromCart, incrementQty, decrementQty, clearCart } from
'../store/cartSlice';
import axios from 'axios';

function Cart() {
  const cart = useSelector(state => state.cart);
  const { cartItems, totalPrice } = cart;
  const dispatch = useDispatch();
  const navigate = useNavigate();
  const [showLoginMessage, setShowLoginMessage] = useState(false);
  const [orderPlaced, setOrderPlaced] = useState(false);
  const [orderId, setOrderId] = useState(null);
  const userInfo = localStorage.getItem('userInfo')
    ? JSON.parse(localStorage.getItem('userInfo'))
    : null;

  const handleRemoveFromCart = (id) => {
    dispatch(removeFromCart(id));
  };

  const handleIncrementQty = (id) => {
    dispatch(incrementQty(id));
  };

  const handleDecrementQty = (id) => {
    dispatch(decrementQty(id));
  };

  const handleCheckout = () => {
    if (!userInfo) {
      setShowLoginMessage(true);
      return;
    }
```

```
    navigate('/checkout');};

  const navigateToOrder = () => {
    if (orderId) {
      navigate(`/order/${orderId}`);
    } else {
      navigate('/orders');}};

  if (orderPlaced) {
    return (
      <div className="cart-page">
        <div className="order-success">
          <h2>Замовлення успішно оформлено!</h2>
          <p>Дякуємо за ваше замовлення. Ви можете переглянути деталі
замовлення або продовжити покупки.</p>
          <div className="order-success-actions">
            <button onClick={navigateToOrder} className="btn">
              Переглянути замовлення
            </button>
            <Link to="/" className="btn">
              Продовжити покупки
            </Link>
          </div>
        </div>
      </div>
    );
  }

  return (
    <div className="cart-page">
      <h2>Кошик покупок</h2>
      {cartItems.length === 0 ? (
        <div className="empty-cart-container">
          <div className="empty-cart">
            <img src="/uploads/images/empty.jpg" alt="Порожній кошик"
className="empty-image" />
          <p>Ваш кошик порожній</p>
          <Link to="/" className="btn">
            Повернутися до покупок
          </Link>
          </div>
        </div>) : (
        <div>
          {showLoginMessage && !userInfo && (
            <div className="login-message auth-modal">
              <p>Для оформлення замовлення необхідно увійти в
обліковий запис</p>
              <div className="login-actions">
                <Link to="/login?redirect=/cart" className="login-
btn"> Увійти
                </Link>
```

```
                <Link                    to="/register?redirect=/cart"
className="register-btn">
                  Зареєструватися
                </Link>
                <button
                  onClick={() => setShowLoginMessage(false)}
                  className="close-btn">Закрити
                </button>
              </div>
            </div>)}
          <div className="cart-items">
            {cartItems.map(item => (
              <div key={item.product} className="cart-item">
                <div className="cart-item-image">
                  <img src={item.image} alt={item.name} />
                </div>
                <div className="cart-item-details">
                  <Link to={`/product/${item.product}`}>
                    <h3>{item.name}</h3>
                  </Link>
                  <p className="price">{item.price} ₴</p>
                </div>
                <div className="cart-item-actions">
                  <button
                    onClick={() => handleDecrementQty(item.product)}
                    className="btn-qty"
                    disabled={item.qty === 1}>-
                  </button>
                  <span className="qty">{item.qty}</span>
                  <button
                    onClick={() => handleIncrementQty(item.product)}
                    className="btn-qty">+
                  </button>
                </div>
                <button
                  onClick={() => handleRemoveFromCart(item.product)}
                  className="btn-remove">Видалити
                </button>
              </div>
            ))}
          </div>
          <div className="cart-summary">
            <h3>Разом: {totalPrice} ₴</h3>
            <button
              onClick={handleCheckout}
              className="btn"
              disabled={cartItems.length === 0}>
              {userInfo ? 'Оформити замовлення' : 'Увійти та оформити
замовлення'}
            </button>
          </div>
        </div>
```

```
      )}
    </div>);}
export default Cart;

Orders.js
import React, { useState, useEffect } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import axios from 'axios';

function Orders() {
  const [orders, setOrders] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const navigate = useNavigate();

  const userInfo = localStorage.getItem('userInfo')
    ? JSON.parse(localStorage.getItem('userInfo'))
    : null;

  useEffect(() => {
    if (!userInfo) {
      navigate('/login');
      return;
    }
    const fetchOrders = async () => {
      try {
        const config = {
          headers: {
            'Content-Type': 'application/json',
          },
        };

        const { data } = await axios.post('/api/orders/myorders', {
userInfo }, config);
        setOrders(data);
        setLoading(false);
      } catch (err) {
        setError('Помилка завантаження замовлень');
        setLoading(false);
      }
    };
    fetchOrders();
  }, [userInfo, navigate]);

  if (loading) return <div>Завантаження...</div>;
  if (error) return <div>{error}</div>;

  return (
    <div className="orders-page">
      <h2>Мої замовлення</h2>
      {orders.length === 0 ? (
        <div className="empty-orders-container">
```

```
          <div className="empty-orders">
            <img   src="/uploads/images/empty.jpg"   alt="Замовлення
відсутні" className="empty-image" />
          <p>У вас немає замовлень</p>
          <Link to="/" className="btn">
            Повернутися до покупок
          </Link>
          </div>
        </div>
      ) : (
        <div className="orders-list">
          {orders.map(order => (
            <div key={order._id} className="order-item">
              <div className="order-header">
                <h3>Замовлення #{order._id}</h3>
                <p>Дата:                                          {new
Date(order.createdAt).toLocaleDateString()}</p>
                <p>Сума: {order.totalPrice} ₴</p>
                <p>
                  Статус: {order.isPaid ? 'Оплачено' : 'Не оплачено'}|
                  {order.isDelivered ? 'Доставлено' : 'В очікуванні'}
                </p>
              </div>
              <div className="order-items">
                <h4>Товари:</h4>
                {order.orderItems.map((item, index) => (
                  <div key={index} className="order-product">
                    <img src={item.image} alt={item.name} />
                    <div className="order-product-details">
                      <h5>{item.name}</h5>
                      <p>{item.quantity}   x   {item.price}   ₴   =
{item.quantity * item.price} ₴</p>
                    </div>
                  </div>
                ))}
              </div>
              <Link to={`/order/${order._id}`} className="btn">
                Деталі замовлення
              </Link>
            </div>
          ))}
        </div>
      )}
    </div>
  );
}
export default Orders;
```