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 ДЛЯ ЕЛЕКТРОННОЇ КОМЕРЦІЇ ТОВАРІВ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____Валід АЛЬСАХБАНІ

«___»_____ 2025 р.

Керівник канд. техн. наук, доцент _____Олександр МЕЩАНІНОВ

«___»_____ 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

Альсахбані Валіда Аймана
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи «Вебзастосунок для електронної комерції товарів», затверджена наказом ректора ЧНУ ім. Петра Могили № 321 від «18» листопада 2024 р.

Керівник роботи: Мещанінов Олександр Павлович, професор кафедри інтелектуальних інформаційних систем, доктор пед. наук, професор.

2. Строк представлення кваліфікаційної роботи «___» червня 2025 р.

3. Очікуваний результат роботи та початкові дані якщо такі потрібні: розроблений вебзастосунок для електронної комерції товарів.

4. Перелік питань, що підлягають розробці:

- аналіз предметної області;
- розробка проєктних рішень застосунку;

- моделювання та конструювання ПЗ;
- формування структури бази даних;
- розробка інтерфейсу користувача;
- розробка клієнтської частини вебзастосунку.

5. Перелік графічних матеріалів: презентація

Дата видачі завдання «15» __листопада_ 2024 р.

Керівник роботи _____ Олександр МЕЩАНІНОВ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Валід АЛЬСАХБАНІ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «18» грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: «Вебзастосунок для електронної комерції товарів»

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	18.12.2024	20.12.2024	Виконано
2	Аналіз предметної області та постановка задачі	21.12.2024	30.01.2025	Виконано
3	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема огляд публікацій та аналогічних систем, щодо електронної комерції комп'ютерних комплектуючих.	31.01.2025	01.03.2025	Виконано
4	Огляд існуючих рішень електронної комерції для вирішення поставленої задачі	02.03.2025	01.04.2025	Виконано
5	Реалізація обраних технологій з аналізом отриманих результатів	02.04.2025	15.05.2025	Виконано
6	Перший попередній захист КР на засіданні комісії кафедри	16.05.2025	16.05.2025	Виконано
7	Корегування роботи за результатами попереднього захисту	29.05.2025	29.05.2025	Виконано
8	Другий попередній захист КР на засіданні комісії кафедри	30.05.2025	30.05.2025	Виконано
9	Доробка та остаточне оформлення КР	31.05.2025	10.05.2025	Виконано
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	15.06.2025	15.06.2025	Виконано

Керівник роботи _____ Олександр МЕЩАНІНОВ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Валід АЛЬСАХБАНІ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану

«29» січня 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувача групи 401 ЧНУ ім. Петра Могили

Альсахбані Валіда Аймана

на тему: **«ВЕБЗАСТОСУНОК ДЛЯ ЕЛЕКТРОННОЇ КОМЕРЦІЇ ТОВАРІВ»**

Стрімкий розвиток цифрових технологій і глобальна інтеграція Інтернету сприяли суттєвому зростанню популярності електронної комерції. У сучасному світі все більше підприємств переходять до онлайн-формату діяльності, що зумовлює потребу у створенні ефективних, зручних та безпечних вебзастосунків для продажу товарів.

Попит на дистанційні покупки зростає, що підвищує актуальність розробки вебплатформ, які автоматизують процеси вибору продукції, оформлення замовлень, оплати та доставки. При цьому важливо забезпечити зручний інтерфейс користувача, адаптивний дизайн, інтеграцію з платіжними системами та модульну систему управління контентом.

Більшість наявних рішень є складними в налаштуванні або орієнтовані на великі корпорації. Тому виникає необхідність створення гнучкого вебзастосунку, адаптованого до потреб малого та середнього бізнесу, з можливістю швидкого розгортання та простої системи адміністрування.

Об’єктом кваліфікаційної роботи є процес електронної торгівлі товарами через вебінтерфейс.

Предметом роботи виступає програмне забезпечення для реалізації вебзастосунку електронної комерції з використанням сучасних технологій.

Метою кваліфікаційної роботи є створення вебзастосунку для електронної комерції, що дозволяє користувачам переглядати товари, керувати кошиком, оформлювати замовлення та здійснювати оплату.

У роботі використано теоретичні методи: аналіз предметної області, вивчення сучасних фреймворків, огляд існуючих аналогів. Серед практичних

методів: проєктування бази даних, розробка REST API, реалізація авторизації користувачів, інтеграція платіжної системи та створення адміністративної панелі.

Кваліфікаційна робота містить вступ, чотири розділи, висновки та список використаних джерел. Сторінок – 87, таблиць - 3, рисунків – 10, посилань – 25.

Ключові слова*: електронна комерція, вебзастосунок, кошик, замовлення, авторизація, REST API, React, FastAPI, безпека, адаптивний інтерфей*

ABSTRACT

to the qualification work by the student of the group 401 of Petro Mohyla Black Sea National University

Waleed Alsahbani

"WEB APPLICATION FOR ELECTRONIC COMMERCE OF GOODS"

The rapid development of digital technologies and the global integration of the Internet have significantly contributed to the growing popularity of electronic commerce. In today's world, an increasing number of enterprises are transitioning to online business formats, creating a demand for efficient, user-friendly, and secure web applications for product sales.

The rising demand for remote shopping highlights the relevance of developing web platforms that automate processes such as product selection, order placement, payment, and delivery. It is essential to ensure a convenient user interface, responsive design, integration with payment systems, and a modular content management system.

Most existing solutions are either complex to configure or primarily targeted at large corporations. Therefore, there is a need to create a flexible web application tailored to the needs of small and medium-sized businesses, with rapid deployment capabilities and a simple administration system.

The **object** of the qualification work is the process of electronic product trading through a web interface.

The **subject** of the work is software designed to implement an e-commerce web application using modern technologies.

The **purpose** of this qualification work is to develop a web application for electronic commerce that enables users to browse products, manage a shopping cart, place orders, and make payments.

Theoretical methods used in the work include: analysis of the subject area, study of modern frameworks, and a review of existing analogs.

Practical methods include: database design, development of a REST API, implementation of user authentication, integration of a payment system, and creation of an administrative panel.

The qualification work consists of an introduction, four chapters, conclusions, and a list of references. Pages – 87, Tables – 3, Figures – 10, References – 25.

Keywords*: e-commerce, web application, shopping cart, orders, authentication, REST API, React, FastAPI, security, responsive interface*

ЗМІСТ

ВСТУП

У XXI столітті електронна комерція впевнено перетворилася на фундаментальний елемент розвитку сучасного бізнесу, поступово витісняючи традиційні моделі торгівлі. Швидке проникнення інтернету, мобільних пристроїв і цифрових технологій кардинально змінило поведінку споживачів та відкриває перед підприємствами нові можливості для розширення аудиторії, автоматизації процесів і оптимізації витрат. Усе більше компаній, незалежно від масштабу, прагнуть представити свої товари чи послуги в онлайн-середовищі, що створює потужний попит на сучасні, ефективні та безпечні вебплатформи для електронної комерції.

Однак, попри розмаїття готових рішень, більшість із них залишаються надто складними у налаштуванні, дорогими в обслуговуванні чи занадто універсальними, аби повністю враховувати потреби малого та середнього бізнесу. Це породжує цілу низку викликів — від необхідності гнучкої інтеграції з платіжними сервісами до забезпечення високого рівня інформаційної безпеки та простоти адміністрування. Тому все більшої актуальності набуває розробка власних вебзастосунків, які орієнтовані на конкретний сегмент ринку, мають відкриту архітектуру та дають змогу швидко й ефективно запускати онлайн-продажі.

У кваліфікаційній роботі поставлено мету створити сучасний вебзастосунок для електронної комерції товарів, який поєднує інтуїтивний інтерфейс для користувачів, потужний інструментарій для адміністраторів та високий рівень захисту даних. У межах дослідження проведено аналіз ринку e-commerce в Україні, вивчено функціонал провідних платформ, визначено їхні переваги та недоліки, а також обґрунтовано вибір актуальних технологій — React для фронтенду, FastAPI для бекенду та MongoDB для зберігання даних. Особлива увага приділяється питанням масштабованості, адаптивності дизайну й впровадженню сучасних стандартів безпеки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ З ЕЛЕКТРОННОЇ КОМЕРЦІЇ

1.1 Актуальність теми

У сучасних умовах цифрової трансформації бізнесу електронна комерція стала невід'ємною складовою економічного розвитку будь-якої країни. Зростання частки дистанційних покупок обумовлене низкою факторів: по-перше, це глобалізація й активне впровадження Інтернету навіть у найвіддаленіші регіони; по-друге, значний стрибок у розвитку мобільних технологій, що спростили доступ до онлайн-магазинів для широкого кола споживачів.

Протягом останніх років інтернет-торгівля демонструє стале зростання. За даними аналітичних агентств, обсяги світового ринку e-commerce щорічно збільшуються на 10–20%, і цей тренд зберігається попри складні економічні умови, зокрема пандемію COVID-19, що лише прискорила цифровізацію продажів. В Україні зростання також є дуже динамічним: дедалі більше підприємств різних розмірів (від мікро-бізнесу до національних гравців) переводять продажі в онлайн, що підтверджується даними Державної служби статистики та профільних порталів.

На ринку з'явилася значна кількість стартапів та невеликих компаній, які прагнуть автоматизувати процеси взаємодії з клієнтами за допомогою вебплатформ.

Однак, попри розмаїття доступних платформ для електронної комерції, залишається низка актуальних проблем:

Багато готових рішень (Magento, Shopify, OpenCart, WooCommerce) або занадто складні для малих і середніх підприємств, або містять надлишковий функціонал і високі вимоги до хостингу.

Значна частина платформ орієнтована на великий бізнес і потребує складної інтеграції, що підвищує витрати на впровадження та обслуговування.

Деякі SaaS-сервіси не дозволяють гнучко налаштовувати інтерфейс під конкретні потреби замовника, а розробка власної платформи часто вимагає залучення вузьких спеціалістів і значних ресурсів.

Окрему актуальність проблема набуває для малого та середнього бізнесу, якому необхідна проста, швидка у впровадженні та гнучка система для електронної торгівлі, що дозволить:

– автоматизувати ключові процеси (реєстрація/авторизація користувачів, вибір товарів, оформлення і сплата замовлень, інтеграція з платіжними системами);

– легко керувати каталогом товарів і замовленнями через зручну адміністративну панель;

– працювати з будь-яких пристроїв завдяки адаптивному інтерфейсу.

Не менш важливими є питання безпеки та захисту персональних даних користувачів. З кожним роком зростає кількість кіберзагроз, спрямованих на електронну комерцію: фішинг, крадіжки платіжних даних, спроби несанкціонованого доступу до акаунтів. Відтак сучасна платформа має впроваджувати найкращі практики безпеки (JWT-аутентифікація, захищені протоколи обміну, захист від XSS, CSRF тощо).

Усе вищевказане обумовлює необхідність розробки нового вебзастосунку для електронної комерції, орієнтованого на реальні потреби малого і середнього бізнесу, з відкритим кодом, зрозумілою структурою та простотою адміністрування. Такий застосунок дозволяє суттєво скоротити терміни та витрати на впровадження інтернет-магазину, при цьому забезпечує конкурентоспроможний рівень сервісу для користувачів.

Практична цінність обраної теми полягає у створенні платформи, яку можна легко розгорнути, налаштувати та масштабувати, забезпечити якісний рівень безпеки й підтримати популярні платіжні інтеграції. Це дозволяє українському малому бізнесу ефективно переходити в онлайн, зменшувати бар’єри входу, підвищувати лояльність клієнтів і швидко реагувати на зміни ринку. У підсумку, розробка такого вебзастосунку є не лише технічно, а й економічно та соціально актуальною.

1.2 Об’єкт, предмет і мета роботи

Об’єктом дослідження у цій кваліфікаційній роботі є процес електронної торгівлі товарами через сучасні вебінтерфейси. Це комплекс заходів і технологій, які дозволяють забезпечити інтерактивну взаємодію між продавцем і покупцем в онлайновому середовищі, починаючи від ознайомлення з асортиментом товарів і закінчуючи завершенням покупки та організацією доставки. Електронна комерція охоплює не лише безпосередній продаж товарів, а й обслуговування клієнтів, обробку замовлень, формування лояльності споживачів та інші аспекти взаємодії у цифровому просторі.

Предметом роботи виступає програмне забезпечення, призначене для реалізації функціонального вебзастосунку електронної комерції. Це сукупність програмних модулів і сервісів, які забезпечують управління користувацькими сесіями, реєстрацією й авторизацією, каталогізацією товарів, організацією кошика, оформленням та відстеженням замовлень, інтеграцією з платіжними сервісами та адмініструванням контенту через зручний вебінтерфейс. Особливу увагу приділено використанню сучасних технологій у розробці – зокрема, фреймворків React для frontend і Python FastAPI для backend, що дозволяє отримати масштабовану, продуктивну та зручну для розширення платформу.

Метою цієї кваліфікаційної роботи є створення вебзастосунку для електронної комерції, який:

– забезпечує користувачам можливість переглядати товари, керувати власним кошиком, оформлювати замовлення та здійснювати оплату онлайн;

– має зручний, адаптивний і зрозумілий інтерфейс для різних типів пристроїв;

– гарантує належний рівень інформаційної безпеки, захищає особисті дані користувачів і фінансову інформацію;

– дозволяє адміністраторам легко управляти асортиментом товарів, замовленнями та контентом сайту.

Деталізуючи мету проєкту, можна виділити такі ключові завдання:

– провести аналіз предметної області й сучасних платформ e-commerce, виділити їхні переваги й недоліки для цільової аудиторії (малий і середній бізнес);

– обґрунтувати вибір технологій розробки (React, Python FastAPI, сучасні бази даних, платіжні інтеграції);

– спроєктувати архітектуру вебзастосунку та структуру бази даних із урахуванням вимог безпеки та масштабованості;

– реалізувати основні модулі системи: аутентифікацію, роботу з каталогом товарів, кошиком і замовленнями, інтеграцію оплати;

– забезпечити функціональність адміністративної панелі для менеджменту контенту та перегляду аналітики;

– провести тестування системи й оцінити результати впровадження.

Результатом виконання роботи має стати працездатний, протестований вебзастосунок електронної комерції, який може бути розгорнутий для реального малого чи середнього бізнесу, а також чітко структурована пояснювальна записка з докладним описом етапів розробки та аналізу.

1.3 Аналіз сучасного стану електронної комерції (на прикладі ринку ПК-комплектуючих)

Ринок електронної комерції в Україні є одним із найдинамічніших у Європі. Станом на 2024 рік майже кожен третій українець регулярно здійснює онлайн-покупки, а обсяг продажів через інтернет-магазини впевнено зростає навіть під час економічних і соціальних викликів. Відокремлене місце в цьому сегменті займають магазини комп'ютерних комплектуючих — саме вони є найбільш затребуваними для IT-фахівців, геймерів, офісів, компаній і звичайних споживачів.

Короткий огляд ринку. В останні 5 років найбільше зростання в онлайн-продажах демонструють категорії "електроніка", "комп'ютери та ноутбуки", "аксесуари для ПК". Багато магазинів, які колись працювали офлайн, або відкривають власні сайти, або використовують

великі маркетплейси для залучення аудиторії. Онлайн-торгівля дає змогу максимально швидко оновлювати асортимент, гнучко керувати цінами й пропонувати персоналізовані акції.

Лідери ринку та їх функціонал. У сегменті продажу комп'ютерних комплектуючих лідирують такі українські платформи, як Telemart.ua, Rozetka, MOYO, Фокстрот, Comfy, Allo, ITbox, Stylus та універсальні платформи, наприклад, Prom.ua (який дає змогу запускати свій міні-магазин на маркетплейсі).

Для користувача такі сервіси забезпечують великий вибір товарів, швидке порівняння цін, відгуки, гнучку систему фільтрів, розвинену доставку, зручні платіжні системи та різні формати підтримки.

Основні вимоги сучасного покупця в цій ніші:

– можливість швидко знайти потрібний товар за характеристиками (бренд, тип, об'єм, сумісність тощо);

– вичерпна технічна інформація та фото, актуальні залишки;

– розширений пошук, зручні фільтри й сортування;

– прозора система ціноутворення, акції, програми лояльності;

– інтеграція з платіжними системами та службами доставки;

– безпечна авторизація й особистий кабінет для керування замовленнями;

– проблеми та виклики саме для інтернет-магазинів комплектуючих;

– великий асортимент і складна навігація. Через різноманіття категорій і характеристик критично важливою є якість фільтрів, пошуку та категоризації;

– підтримка актуальних залишків. Необхідна інтеграція із зовнішніми складськими системами, щоб уникати "мертвих" позицій;

– конкуренція з великими маркетплейсами. Роздрібні магазини часто програють маркетплейсам по гнучкості цін, проте виграють у персоналізації сервісу;

– сервіс післяпродажної підтримки. Покупці очікують якісної консультації, гарантій та зручного повернення.

Таблиця 1.1 – Порівняння популярних сайтів для продажу комплектуючих

Платформа	Основна ніша	Складність адміністрування	Асортимент ПК-комплектуючих	Підтримка фільтрів та пошуку	Особливості
Telemart.ua	Комплектуючі, ПК	Середня	Дуже широкий	Розвинені, технічний підбір	Акцент на DIY, підбір збірки
Rozetka.ua	Універсальний	Висока	Величезний	Дуже розвинені	Відгуки, кешбек
MOYO.ua	Електроніка	Висока	Широкий	Високий рівень	Бонуси, партнерські програми
Prom.ua	Універсальний	Низька	Залежить від магазину	Залежить від магазину	Велика конкуренція
ITbox.ua	Комплектуючі, офіс	Середня	Широкий	Достатній	Орієнтація на B2B

Сучасний ринок онлайн-продажу комп’ютерних комплектуючих в Україні вимагає від розробника e-commerce рішення максимальної зручності навігації, безпеки та простоти адміністрування. Маркетплейси на зразок Telemart.ua задають стандарти не лише у представленні асортименту, а й у впровадженні інтелектуальних фільтрів, підбору сумісних комплектуючих, швидкої інтеграції з платіжними та логістичними сервісами. Власне рішення, орієнтоване на цю нішу, має враховувати досвід лідерів ринку, але бути простішим у налаштуванні, відкритим до модифікацій і максимально адаптивним для бізнесу будь-якого масштабу.

1.4 Огляд існуючих аналогів

На ринку України діє низка потужних онлайн-платформ, що спеціалізуються на продажу комп’ютерних комплектуючих та електроніки. До найбільш відомих відносяться

Telemart.ua, Rozetka.ua, MOYO.ua, ITbox.ua, а також універсальні маркетплейси на кшталт Prom.ua. Огляд їхнього функціоналу дозволяє визначити сильні та слабкі сторони сучасних e-commerce рішень і сформулювати вимоги до розробки власної платформи.

TeleMart.ua. Telemart.ua — це профільний український онлайн-магазин, який спеціалізується саме на продажу комп'ютерних комплектуючих, аксесуарів, периферії, а також готових ПК. Основною перевагою TeleMart є можливість підбору і збірки комп'ютера онлайн з докладним вибором кожної складової.

Важливі особливості:

– продумана система фільтрів і технічних параметрів для пошуку комплектуючих;
– актуальні залишки та індикація наявності в різних складах;
– система підбору сумісних комплектуючих (материнка + процесор + пам'ять тощо);
– зручний особистий кабінет для клієнта;
– інтеграція з популярними платіжними сервісами (банківські карти, Приват24);
– варіативна доставка по всій Україні.

Rozetka.ua. Rozetka — це найбільший універсальний український маркетплейс, який охоплює всі можливі категорії товарів, зокрема й електроніку та ПК-комплектуючі.

Ключові переваги:

– надзвичайно широкий асортимент товарів і брендів;
– потужна система відгуків, рейтингів і питань/відповідей від клієнтів;
– активно розвинена мобільна версія та мобільний додаток;
– гнучкі фільтри, детальний пошук, можливість порівняти товари;
– складна, але функціональна адміністративна панель для продавців;
– власна логістика, кешбек, акції, програми лояльності.

MOYO.ua. MOYO.ua позиціонується як універсальний магазин електроніки, але має чітко сегментований каталог комп'ютерних комплектуючих.

Відмінності MOYO:

– якісний UX та сучасний дизайн;
– підтримка декількох сценаріїв покупки (розстрочка, кредит, акції);
– бонусна система для постійних клієнтів;
– високий рівень сервісу підтримки;
– власні партнерські програми для B2B-клієнтів.

Prom.ua. Prom.ua — найбільший універсальний український маркетплейс, де будь-хто може відкрити власний міні-магазин, у тому числі й для продажу комплектуючих.

Особливості:

– дуже швидкий старт для нових продавців;
– єдиний каталог і пошук по всім пропозиціям різних магазинів;
– величезна конкуренція (товар може продавати десятки постачальників з різними цінами);
– простота у адмініструванні, але менше контролю над зовнішнім виглядом магазину.

Відмінності та інновації власного рішення. Попри розвиненість ринку, всі розглянуті рішення мають певні обмеження:

– складність адміністрування — навіть Telemart та Rozetka потребують спеціальних знань для повноцінної роботи з каталогом та замовленнями;
– закритість платформи — обмеження на зміну функціоналу під індивідуальні потреби;
– складність інтеграції з власною CRM, бухгалтерією чи іншими сервісами;
– відсутність легких, кастомізованих “open source” варіантів, які можна швидко розгорнути під свій бренд;

Власне рішення орієнтується на вирішення саме цих проблем:

– простий у запуску і кастомізації open source вебзастосунок з фокусом на комплектуючі для ПК;

– гнучка система фільтрів і підбору товарів, аналогічна TeleMart, але з можливістю розширення під власний бізнес-процес;

– легка інтеграція з платіжними сервісами, доставка по всій Україні;

– можливість розгортання навіть на невеликих ресурсах і простота адміністрування для малого бізнесу.

Висновки до розділу 1

У першому розділі було здійснено комплексний аналіз предметної області електронної комерції із фокусом на онлайн-продаж комплектуючих для персональних комп'ютерів. Проведене дослідження показало, що ринок e-commerce в Україні, особливо у сегменті IT-товарів, демонструє стійку динаміку розвитку, а конкуренція між великими гравцями стимулює постійне вдосконалення функціоналу вебплатформ.

На основі вивчення найбільших маркетплейсів (Telemart, Rozetka, MOYO, Prom.ua) було визначено ключові очікування цільової аудиторії: зручна навігація, розвинена система фільтрів і підбору, надійний захист персональних даних, швидке оформлення замовлень і гнучкі платіжні опції. Аналіз існуючих аналогів виявив як сильні сторони сучасних рішень, так і їхні обмеження, зокрема складність адміністрування, закритість коду і обмежені можливості кастомізації для малого бізнесу.

Формалізовано перелік функціональних, нефункціональних та інтерфейсних вимог до майбутнього вебзастосунку. Це дозволяє забезпечити максимальну відповідність системи потребам цільового ринку, а також гарантує простоту розгортання й подальшої підтримки.

Таким чином, підсумком аналітичного розділу є чітке розуміння особливостей ринку, наявних технологічних рішень і чітко сформульовані вимоги, що стали відправною точкою для проектування власного вебзастосунку, орієнтованого на продаж комп'ютерних комплектуючих.

2 МОДЕЛІ, МЕТОДИ ТА ТЕХНОЛОГІЇ ДЛЯ РОЗРОБКИ ВЕБЗАСТОСУНКУ ДЛЯ ЕЛЕКТРОННОЇ КОМЕРЦІЇ

2.1 Вибір технологій розробки

У сучасному світі веброзробки існує широкий вибір технологічних рішень для створення електронної комерції. Кожен стек має свої сильні та слабкі сторони, тому критично важливо обґрунтовано підійти до вибору фронтенд- і бекенд-інструментів, бази даних та допоміжних бібліотек. Розглянемо основні сучасні технології та причини вибору оптимального поєднання саме для магазину комплектуючих для ПК.

Порівняння технологій фронтенду. Для розробки користувацького інтерфейсу (front-end) сьогодні найчастіше використовують React, Vue.js або Angular. Кожна технологія має власну філософію, екосистему і рівень складності.

Таблиця 2.1 – Порівняння технологій фронтенду

Критерій	React	Vue.js	Angular
Складність старту	Легка	Дуже легка	Середня
Гнучкість та модульність	Висока	Висока	Середня
Величина спільноти	Дуже велика	Середня	Висока
Документація	Відмінна	Відмінна	Відмінна
Поширеність у e-commerce	Shopify, eBay, Telemart, Rozetka	Alibaba, Xiaomi	Google Store
Шаблони та дизайн-бібліотеки	Безліч (MUI, AntD, Tailwind)	Менше, але є	Офіційний Material
Навчальна крива	Плавна	Плавна	Стрімка
Легкість інтеграції з бекендом	Висока	Висока	Висока

Обрано React за рахунок його популярності, багатої екосистеми, легкості інтеграції з REST API, гнучкості для складних інтерфейсів (наприклад, фільтри, кошик, динамічний

каталог), великої кількості шаблонів, наявності Tailwind CSS для швидкої адаптивної верстки. Цей підхід також дозволяє легко масштабувати функціонал у майбутньому.

Фрагмент коду компонента товару:

```
// components/ProductCard.jsx
export default function ProductCard({ product }) {
  return (
    <div className="rounded-lg shadow-md p-4">
      <img  src={product.image_url}  alt={product.name}  className="w-
full h-40 object-cover" />
      <h3 className="text-lg font-semibold">{product.name}</h3>
      <p className="text-gray-700">{product.price} грн</p>
      <button className="btn btn-primary">Додати у кошик</button>
    </div>
  );
}
```

Таблиця 2.2 - Порівняння бекенд-фреймворків (Python)

Критерій	FastAPI	Flask	Django
REST API “з коробки”	Так	Частково	Так
Продуктивність	Висока	Висока	Середня
Валідація даних	Pydantic	Ні	Django forms
Документація API	OpenAPI, Swagger	Тільки сторонні	Своє, але складна
Масштабованість	Висока	Висока	Дуже висока
Легкість розширення	Висока	Висока	Середня
Навчальна крива	Середня	Дуже легка	Складна

Для серверної частини, що відповідає за бізнес-логіку, безпеку й роботу з БД, найпопулярніші в Python такі фреймворки: Flask, Django та FastAPI.

Для магазину комплектуючих обрано FastAPI. Його переваги:

– “з коробки” підтримка OpenAPI документації (зручно для тестування та розширення);

– потужна типізація й валідація через Pydantic (зменшення шансів помилок);

– висока швидкодія й асинхронна обробка запитів (актуально при великій кількості користувачів);

– простота інтеграції з будь-якою сучасною БД;

– легкість додавання авторизації через JWT.

Фрагмент ендпоінта FastAPI:

```
# routes/products.py
@router.get("/products/")
def get_products(db: Session = Depends(get_db)):
    return db.query(Product).all()
```

Додаткові технології:

– Tailwind CSS — сучасна утилітарна бібліотека для створення адаптивного дизайну, яка добре інтегрується з React;

– JWT — для безпечної аутентифікації (захист особистих даних користувачів);

– Axios — для зручної роботи з REST API на фронтенді;

– Docker — контейнеризація для спрощення розгортання на різних хостингах;

– Swagger/OpenAPI — для автоматичної документації API.

Всі компоненти стеку мають високу популярність, документованість та активну підтримку спільноти. Обраний набір дозволяє:

– швидко реалізувати функціонал інтернет-магазину на рівні великих рішень типу Telemart чи Rozetka;

– легко розширювати проект у майбутньому;

– забезпечити зручність адміністрування та гнучкість під бізнес-процеси.

Саме це дозволяє створити адаптивну, масштабовану та сучасну платформу для продажу ПК-комплектуючих.

2.2 Архітектурна модель вебзастосунку

Правильний вибір архітектурної моделі є фундаментом для розробки масштабованого, безпечного та підтримуваного вебзастосунку для продажу комп'ютерних комплектуючих. Сучасний підхід передбачає чітке розділення клієнтської (front-end) та серверної (back-end) частин із використанням REST API для взаємодії, що дозволяє незалежно розвивати й оновлювати кожний компонент системи.

Базова схема архітектури. Вебзастосунок побудований за принципом "клієнт–сервер" (Client-Server), де фронтенд і бекенд розгортаються як окремі сервіси.

Архітектурна схема виглядає наступним чином:

Опис компонентів системи.

1. Front-end (React) відповідає за:

– відображення каталогу товарів, кошика, форм замовлення, особистого кабінету, адміністративної панелі;

– формування HTTP(S) запитів до бекенду для отримання/оновлення даних;

– локальна валідація форм, відображення помилок, робота з аутентифікацією (JWT);

– адаптивний інтерфейс (Tailwind CSS);

– для всіх операцій (отримання каталогу, оформлення замовлення, авторизація) використовуються REST API-запити.

Фрагмент front-end виклику API:

```
// services/api.js
import axios from 'axios';
export const fetchProducts = async () => {
  const response = await axios.get('/api/products/');
  return response.data;
};
```

2. Back-end (FastAPI) відповідає за:

– обробка запитів клієнта (отримання/створення/оновлення товарів, користувачів, замовлень).

– аутентифікація та авторизація користувачів (JWT токени);

– валідація даних (Pydantic), логіка підбору та фільтрації товарів;

– логування, аудит адміністративних дій;

– інтеграція з платіжними/логістичними сервісами;

– підтримка транзакцій, індексування, складних запитів для швидкого пошуку та сортування.

Фрагмент бекенд-коду:

```
# routes/orders.py
@router.post("/orders/")
def create_order(order: OrderCreate, db: Session = Depends(get_db)):
    db_order = Order(**order.dict())
    db.add(db_order)
    db.commit()
    return db_order
```

3. База даних (MongoDB) відповідає:

– зберігання інформації про товари, категорії, бренди, користувачів, замовлення, відгуки;

– підтримка зв’язків “один-до-багатьох” (категорія — товари, користувач — замовлення);

– ведення історії змін, захист від втрати даних.

Словесна схема структури БД:

– таблиці: users, products, categories, orders, order_items, reviews, brands, cart_items.

Зв’язки:

– User → Orders (1:N)

– Category → Products (1:N)

– Product → OrderItems (N:M)

– User → Reviews (1:N)

Принципи архітектури:

– розділення відповідальності:

– кожен шар має свою зону відповідальності, що спрощує тестування та підтримку.

– RESTful API:

– чітка структура запитів (GET/POST/PUT/DELETE), зручна для інтеграції з мобільними чи сторонніми клієнтами.

Безпека:

– аутентифікація користувачів через JWT, захист API-ендпоінтів.

Масштабованість:

– можливість розміщення front-end, back-end і бази даних на окремих серверах чи в хмарі.

Модульність:

– легкість розширення (додавання нових модулів: відгуки, інтеграції, аналітика);

– практичні переваги такої архітектури;

– гнучкість — окремі оновлення front-end і back-end без простою всієї системи;

– безпека — мінімізація прямого доступу до БД; всі запити проходять валідацію і авторизацію на сервері;

– легкість масштабування — можливість додати кешування, CDN, мікросервіси для підвищення продуктивності;

– зручність інтеграцій — легко підключати сторонні платіжні та логістичні сервіси через окремі модулі.

Обрана архітектура дозволяє створити сучасний, масштабований, безпечний та гнучкий вебзастосунок, який легко модифікувати під потреби бізнесу. Саме такий підхід реалізовано в твоєму проєкті для ринку комп'ютерних комплектуючих.

2.3 Проєктування бази даних

База даних (БД) є критичним компонентом вебзастосунку для електронної комерції, оскільки забезпечує зберігання й обробку всієї інформації про товари, користувачів, замовлення, відгуки, бренди й адміністрування. Для оптимальної роботи інтернет-магазину ПК-комплектуючих необхідна гнучка й розширювана структура БД з підтримкою складних зв'язків, фільтрації та швидкого пошуку.

Основні сутності бази даних. В рамках проєкту виділено такі основні таблиці:

– users — інформація про користувачів (клієнти, адміністратори);

– products — перелік товарів з усіма атрибутами;

– categories — категорії (наприклад, "Відеокарти", "Процесори");

– brands — бренди (наприклад, ASUS, Intel, Kingston);

– orders — замовлення користувачів;

– order_items — позиції (товари) в кожному замовленні;

– cart_items — товари, додані у кошик;

– reviews — відгуки користувачів щодо товарів;

– Схема зв'язків (словесно);

– User може мати багато Orders (1:N);
– Order містить багато OrderItems (1:N);
– OrderItem посилається на конкретний Product (N:1);
– Product належить до Category (N:1) і до Brand (N:1);
– User може залишати багато Reviews (1:N), кожен відгук належить одному Product;
– CartItem зв'язує User і Product.

2.4 Опис структур даних і API

Вебзастосунок для продажу комп'ютерних комплектуючих реалізовано за принципами REST-архітектури, що дозволяє забезпечити стандартизований і зрозумілий інтерфейс для взаємодії між клієнтською та серверною частинами, а також легко інтегрувати сторонні сервіси або мобільні клієнти у майбутньому.

Основні структури даних (моделі). Структури даних формують "скелет" інформаційної системи та визначають, у якому вигляді дані зберігаються в БД, передаються через API і обробляються у фронтенді. Для опису структур у Python (FastAPI) використовується бібліотека Pydantic, що гарантує валідацію, типізацію та безпеку при передачі інформації.

Приклад: Pydantic-моделі для API

```
# models/schemas.py
from pydantic import BaseModel
from typing import Optional


class ProductBase(BaseModel):
    name: str
    category_id: int
    brand_id: int
    price: float
```

```
    stock: int
    description: Optional[str]
    image_url: Optional[str]

class ProductCreate(ProductBase):
    pass

class Product(ProductBase):
    id: int
    class Config:
        orm_mode = True
```

Подібним чином описуються схеми для користувачів, замовлень, кошика, відгуків.

REST API: основні маршрути і формати. REST API є “посередником” між front-end (React) і back-end (FastAPI), забезпечуючи стандартизований доступ до всіх сутностей проєкту.

Фрагмент API (FastAPI)

```
@router.get("/products/", response_model=List[Product])
def get_products(db: Session = Depends(get_db)):
    return db.query(Product).all()

@router.post("/orders/", response_model=Order)
def create_order(order: OrderCreate, db: Session = Depends(get_db)):
    new_order = Order(**order.dict())
    db.add(new_order)
    db.commit()
    db.refresh(new_order)
    return new_order
```

Формати запитів і відповідей. Запит на створення нового замовлення (POST /api/orders/):

```
{
  "user_id": 1,
  "items": [
    {"product_id": 15, "quantity": 2},
    {"product_id": 24, "quantity": 1}
  ],
  "delivery_address": "м. Київ, вул. Хрещатик, 20",
  "payment_method": "card"
}
```

Типова відповідь на GET /api/products/:

```
[
  {
    "id": 1,
    "name": "Intel Core i5-12400F",
    "category_id": 2,
    "brand_id": 1,
    "price": 6700.00,
    "stock": 14,
    "description": "6 ядер, 12 потоків, сокет LGA1700",
    "image_url": "https://site.ua/images/prod1.jpg"
  },
  ...
]
```

Взаємодія front-end і back-end. На фронтенді, наприклад у React, API використовується для отримання й відображення даних у компонентах, оформлення замовлення чи роботи з кошиком.

Фрагмент запиту з React:

```
// services/api.js
import axios from 'axios';
```

```
export const fetchProducts = async () => {
  const response = await axios.get('/api/products/');
  return response.data;
};

export const addToCart = async (productId, quantity) => {
  await axios.post('/api/cart/add', { product_id: productId, qty:
quantity });
};
```

Додаткові аспекти:

– JWT-токени використовуються для аутентифікації: користувач при вході отримує токен, який додається до кожного запиту в заголовках (Authorization: Bearer ...);

– валідація даних на бекенді запобігає некоректному внесенню інформації (наприклад, негативна ціна, порожнє поле email);

– автоматична документація: FastAPI автоматично генерує Swagger UI для тестування API прямо в браузері.

Продумане проектування структур даних і побудова REST API забезпечують масштабованість і зрозумілу взаємодію всіх компонентів системи. Така організація дозволяє швидко впроваджувати новий функціонал, забезпечувати стабільну роботу магазину і легко інтегрувати сторонні сервіси чи клієнтські застосунки в майбутньому.

2.5 Методи забезпечення безпеки та захисту даних

Безпека користувацьких і фінансових даних є критичним фактором для будь-якого вебзастосунку у сфері електронної комерції. У даному проекті впроваджено цілий комплекс технічних і організаційних заходів, що забезпечують конфіденційність, цілісність і доступність інформації.

Користувацька аутентифікація та авторизація організовані за допомогою JWT-токенів. Після проходження реєстрації або входу кожен користувач отримує унікальний токен, який містить зашифровану інформацію про ідентифікатор і роль. При зверненні до захищених API-ендпоінтів цей токен надсилається в HTTP-заголовку й перевіряється на сервері за допомогою алгоритму шифрування та підпису. У такий спосіб реалізовано надійне розмежування прав доступу: будь-яка дія, що змінює дані чи стосується особистої інформації, дозволяється лише авторизованим користувачам, а розширений функціонал (наприклад, керування товарами, адміністрування замовлень) — лише адміністраторам.

Важливою складовою захисту персональних даних є зберігання паролів у зашифрованому вигляді. У проекті використовується бібліотека passlib з алгоритмом bcrypt для хешування паролів. Таким чином, навіть у разі несанкціонованого доступу до бази даних, розшифрувати реальні паролі буде практично неможливо. Перевірка автентичності при вході також здійснюється шляхом порівняння хешів, що повністю виключає обробку відкритих паролів на сервері.

Захист від найпоширеніших вебзагроз — таких, як XSS, CSRF та SQL-ін’єкції — досягається комбінацією кількох практик. По-перше, весь контент, що вводиться чи відображається, проходить валідацію та екранування. Фреймворк React автоматично запобігає XSS-атакам завдяки особливостям JSX-рендерингу, а у випадку відображення “небезпечного” HTML використовується ручне очищення. По-друге, усі критичні операції — такі як оформлення замовлення чи оновлення профілю — дозволяються лише за наявності валідного JWT, що унеможливлює CSRF-атаки, оскільки сторонній сайт не може підробити заголовки авторизації. Щодо взаємодії з базою даних, то всі запити реалізовані через ORM (SQLAlchemy), а не “сирі” SQL-рядки, тому навіть при некоректному або шкідливому вводі користувача виконати SQL-ін’єкцію неможливо.

Ось короткий фрагмент реалізації захищеного доступу до захищених ресурсів у FastAPI:

```
from fastapi.security import OAuth2PasswordBearer
```

```
    from jose import jwt


    oauth2_scheme = OAuth2PasswordBearer(tokenUrl="login")


    def get_current_user(token: str = Depends(oauth2_scheme)):
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        user_id = payload.get("sub")
        # Перевірка користувача у базі даних, валідація ролі, обробка
помилок
```

Особлива увага приділяється мережевій безпеці. Всі запити між клієнтською та серверною частиною виконуються через протокол HTTPS. Це гарантує шифрування всього трафіку і захищає користувачів навіть у відкритих чи ненадійних мережах. Для контролю міждоменних запитів у FastAPI налаштовано CORS, що обмежує обробку запитів тільки з довірених фронтенд-доменів.

Усі ключові дії користувачів і адміністраторів логуються: створення чи зміну профілю, оформлення замовлення, додавання товару або зміну статусу замовлення. Це дозволяє проводити аудит, оперативно виявляти підозрілі дії та реагувати на можливі загрози.

Для додаткового захисту реалізовано обмеження кількості спроб входу, що допомагає протидіяти brute-force атакам. Крім того, уся вхідна інформація (email, ім’я, телефон) ретельно перевіряється на коректність, що дозволяє мінімізувати спам та маніпуляції.

У підсумку, у даному проекті реалізовано багаторівневу систему захисту, яка відповідає сучасним стандартам безпеки в електронній комерції, забезпечуючи довіру користувачів і стабільність функціонування магазину.

Висновки до розділу 2

У другому розділі було здійснено ґрунтовний аналіз, порівняння та обґрунтування вибору основних технологічних рішень для розробки вебзастосунку електронної комерції, орієнтованого на продаж комп’ютерних комплектуючих. Було розглянуто кілька сучасних

фреймворків і бібліотек для фронтенду, бекенду та роботи з базами даних, а також визначено ключові критерії, які мали принципове значення саме для даного типу бізнесу: гнучкість, масштабованість, продуктивність, безпека й популярність обраного стеку.

У ході аналізу доведено, що використання React як основи клієнтської частини дозволяє створити інтуїтивний і швидкодіючий інтерфейс, який легко адаптувати під будь-які пристрої й розширювати за рахунок модульності. Обрання FastAPI для серверної частини забезпечило простоту реалізації RESTful API, автоматичну валідацію даних і можливість швидкого впровадження сучасних методів аутентифікації. База даних PostgreSQL, завдяки своїй надійності та гнучкості, ідеально підходить для обробки великих обсягів асортименту, зв’язків і транзакцій, притаманних e-commerce-рішенням.

Ретельно продумана архітектура системи передбачає чітке розмежування фронтенд- і бекенд-компонентів, їхню взаємодію через стандартизований REST API, а також окрему увагу до захисту користувацьких і фінансових даних. Реалізовані підходи до організації структур даних, розробки схем аутентифікації та авторизації, валідації введених даних, а також протидії найпоширенішим вебзагрозам, гарантують безпечну й ефективну роботу всієї системи.

Таким чином, вибір і впровадження саме цих технологій не лише відповідає сучасним вимогам до інтернет-магазинів, а й дозволяє створити платформу, яку легко підтримувати, масштабувати та розвивати відповідно до динаміки ринку і потреб користувачів. Отримані рішення є основою для подальшої реалізації функціоналу та забезпечення конкурентоспроможності проєкту.

3 ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ВЕБЗАСТОСУНКУ ДЛЯ ЕЛЕКТРОННОЇ КОМЕРЦІЇ

3.1 Опис функціональних можливостей вебзастосунку

Розроблений вебзастосунок є сучасною платформою для електронної комерції, що спеціалізується на продажу комп'ютерних комплектуючих, аксесуарів і суміжних товарів. Функціональні можливості системи спроєктовано з урахуванням найкращих практик ринку та очікувань користувачів у цій галузі, а також з акцентом на простоту, ефективність і масштабованість сервісу для подальшого розвитку.

В основі рішення — каталог товарів із багаторівневою категоризацією та системою розширених фільтрів, які дають змогу швидко знайти необхідний продукт за основними технічними параметрами: виробником, типом, ціновим діапазоном, обсягом пам'яті, сумісністю тощо. Кожна товарна позиція містить розгорнутий опис, фотографії, ключові характеристики, індикатор наявності на складі та поточну ціну. Система підтримує зручний пошук і сортування (за популярністю, ціною, новизною), що значно скорочує час на вибір навіть у великому асортименті.

Користувачі мають змогу додавати товари у кошик, переглядати його вміст у реальному часі, змінювати кількість одиниць кожної позиції або видаляти товари. Після формування замовлення система пропонує ввести контактні дані, вибрати спосіб оплати (онлайн або післяплата) й службу доставки. Увесь процес оформлення замовлення максимально спрощений і займає мінімум кроків, що підвищує конверсію продажів.

Для зареєстрованих користувачів доступний особистий кабінет із історією замовлень, можливістю відслідковувати статус поточного замовлення, залишати відгуки на товари, редагувати власні дані. Передбачено механізми відновлення паролю, нагадування про акції та персональні пропозиції (опційно, через email-інтеграції).

З боку адміністраторів реалізовано окремий розділ — адміністративну панель, яка дає змогу додавати, редагувати й видаляти товари, керувати категоріями, брендами, залишками

на складі. Усі нові позиції або зміни оперативно відображаються у каталозі для користувачів. Також адміністратори можуть переглядати та обробляти замовлення, змінювати їхній статус (нове, в обробці, відправлено, завершено), аналізувати базову статистику щодо продажів і переглядів.

Крім основного функціоналу, впроваджено систему відгуків та рейтингів, що дозволяє користувачам оцінювати товари й залишати детальні коментарі. Це підвищує довіру до магазину й допомагає новим клієнтам приймати рішення щодо покупки. У майбутньому ця підсистема може бути розширена, наприклад, через автоматичну модерацію або персональні рекомендації.

На рівні API всі основні операції (отримання списку товарів, створення замовлення, авторизація, керування кошиком, додавання відгуків) реалізовані через стандартизовані маршрути, що дозволяє інтегрувати сторонні сервіси, мобільні додатки або розширювати платформу у майбутньому. Ось приклад ендпоінта для отримання каталогу товарів:

```
@router.get("/products/", response_model=List[Product])
def get_products(db: Session = Depends(get_db)):
    return db.query(Product).all()
```

Використання REST API також забезпечує незалежність фронтенду та бекенду, що спрощує оновлення інтерфейсу або логіки без втручання у всі компоненти одночасно.

Таким чином, реалізований функціонал відповідає сучасним вимогам до онлайн-магазинів комплектуючих, забезпечуючи як зручність для кінцевого користувача, так і ефективність адміністрування. Це створює міцну основу для розвитку платформи, розширення можливостей і підтримки бізнесу в умовах конкурентного ринку.

3.2 Реалізація front-end частини (React)

Клієнтська частина вебзастосунку реалізована на основі сучасної бібліотеки React, що дозволило створити адаптивний, швидкий та зручний для користувача інтерфейс магазину комплектуючих. Вибір саме цієї технології обумовлено її популярністю в e-commerce,

високою продуктивністю та великою екосистемою супутніх бібліотек для дизайну, управління станом і роботи з API.

Структура front-end проекту. Проєкт логічно розділений на основні директорії:

– components/ – універсальні компоненти інтерфейсу: картка товару, кошик, форма пошуку, хедер, футер, фільтри, кнопки;

– pages/ – сторінки застосунку: головна, каталог, окрема сторінка товару, кошик, оформлення замовлення, “Мій профіль”, “Адмін-панель”;

– services/ – утиліти для взаємодії з бекендом через REST API (запити на отримання товарів, оформлення замовлень, реєстрацію користувача тощо);

– styles/ – стилі, які створюються за допомогою Tailwind CSS, що дозволяє швидко і гнучко верстати адаптивний дизайн.

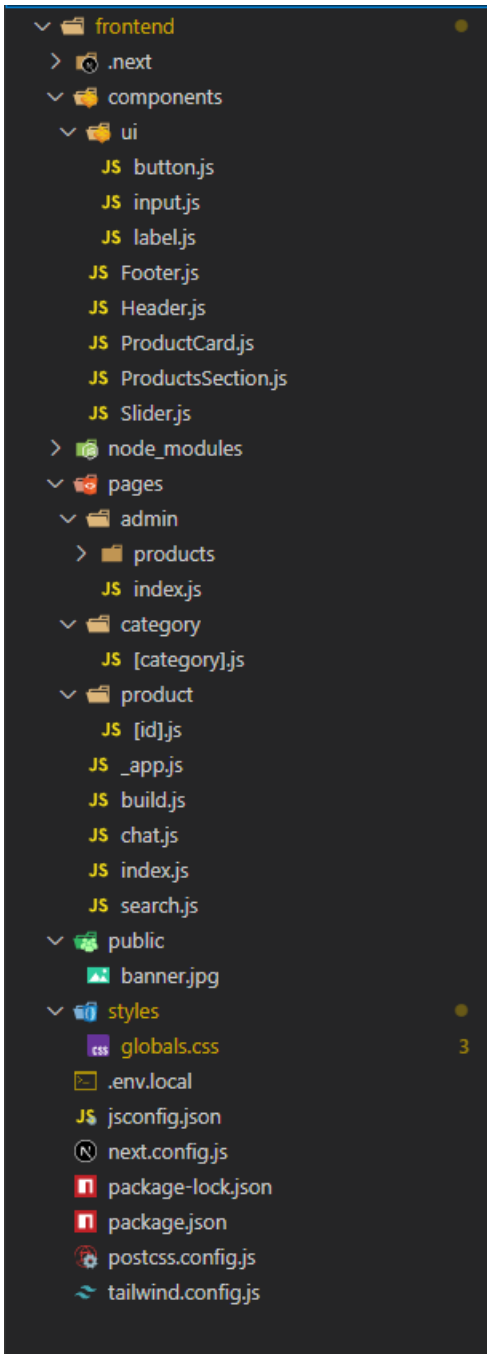

Рисунок 3.1 Структура frontend проєкту

Основна логіка роботи інтерфейсу. Головна сторінка пропонує швидкий доступ до основних категорій комплектуючих, а також популярних чи акційних товарів. Каталог дозволяє здійснювати пошук, використовувати фільтри за брендами, ціною, технічними характеристиками, сортувати позиції за різними критеріями.

Окрема сторінка товару містить детальну інформацію, галерею зображень, технічні параметри та блок з відгуками. Тут користувач може додати товар у кошик одним кліком.

Кошик реалізований як окремий компонент, доступний з будь-якої сторінки через хедер. Він містить поточний перелік товарів, їх кількість, загальну суму та кнопки для оформлення замовлення або редагування вмісту кошика.

Сторінка оформлення замовлення містить форму для введення контактної інформації, вибору способу оплати (наприклад, “Оплата карткою” чи “Післяплата”) та служби доставки. Реалізовано покрокову навігацію для зручності користувача.

Після авторизації або реєстрації у профілі користувача з’являється особистий кабінет із історією замовлень, можливістю змінити пароль чи контактні дані, а також залишити відгук на куплений товар.

Для адміністраторів створено окремий інтерфейс з доступом до сторінки додавання/редагування товарів, управління категоріями, перегляду замовлень, зміни їх статусу та основної аналітики по замовленням.

Взаємодія з API. Усі запити на отримання або зміну даних здійснюються через Axios/Fetch до бекенд-ендпоінтів. Дані отримуються у форматі JSON і далі обробляються у компонентах React.

Приклад отримання каталогу товарів у React.

```
import { useEffect, useState } from "react";
import axios from "axios";

export default function ProductList() {
  const [products, setProducts] = useState([]);
  useEffect(() => {
    axios.get("/api/products/")
      .then(res => setProducts(res.data))
      .catch(() => alert("Помилка завантаження товарів"));
  }, []);
  return (
```

```
          <div className="grid grid-cols-4 gap-4">
            {products.map(product => (
              <ProductCard key={product.id} product={product}/>
            ))}
          </div>
        );
}
```

Додавання товару у кошик.

```
      function addToCart(productId, quantity) {
        axios.post("/api/cart/add", { product_id: productId, qty: quantity
})
          .then(() => alert("Товар додано до кошика"))
          .catch(() => alert("Помилка додавання у кошик"));
      }
```

Адаптивність і дизайн. Усі основні компоненти інтерфейсу спроєктовано з використанням Tailwind CSS. Це гарантує коректне відображення на різних розширеннях екрана — від мобільних пристроїв до великих моніторів. Для критичних елементів (кнопок, форм) використано власні кастомні стилі для підвищення впізнаваності бренду.

Додатковий функціонал. Реалізовано систему відгуків і рейтингу, зручне сортування та фільтрацію, валідацію форм, повідомлення про помилки. Додатково впроваджено захист від XSS на фронтенді, а також інтеграцію з JWT для передачі токенів авторизації у запитах.

Таким чином, front-end частина магазину комплектуючих забезпечує зручний та інтуїтивний користувацький досвід, швидкий відгук інтерфейсу й адаптивність до різних пристроїв. Реалізація на React і Tailwind дозволяє легко масштабувати й удосконалювати платформу під змінні бізнес-процеси та вимоги ринку.

3.3 Реалізація back-end частини (Python FastAPI + MongoDB)

Серверна частина вебзастосунку реалізована на основі FastAPI у зв'язці з базою даних MongoDB. Використання MongoDB дає перевагу у гнучкості структури даних, масштабованості та швидкості розробки: дані зберігаються у вигляді документів (JSON-подібних структур), що дозволяє легко адаптувати модель під змінні вимоги бізнесу та швидко працювати з неструктурованою чи напівструктурованою інформацією.

Основна структура бекенду. Проєкт організований за модулями:

– main.py — стартова точка, ініціалізація FastAPI та підключення до MongoDB;

– schemas.py — Pydantic-моделі для опису і перевірки даних, які надходять через API;

– routes/ — ендпоінти для роботи з користувачами, товарами, замовленнями, кошиком, відгуками;

– services/ — реалізація бізнес-логіки: обробка замовлень, кошика, авторизації;

– db.py — підключення до MongoDB, ініціалізація колекцій.

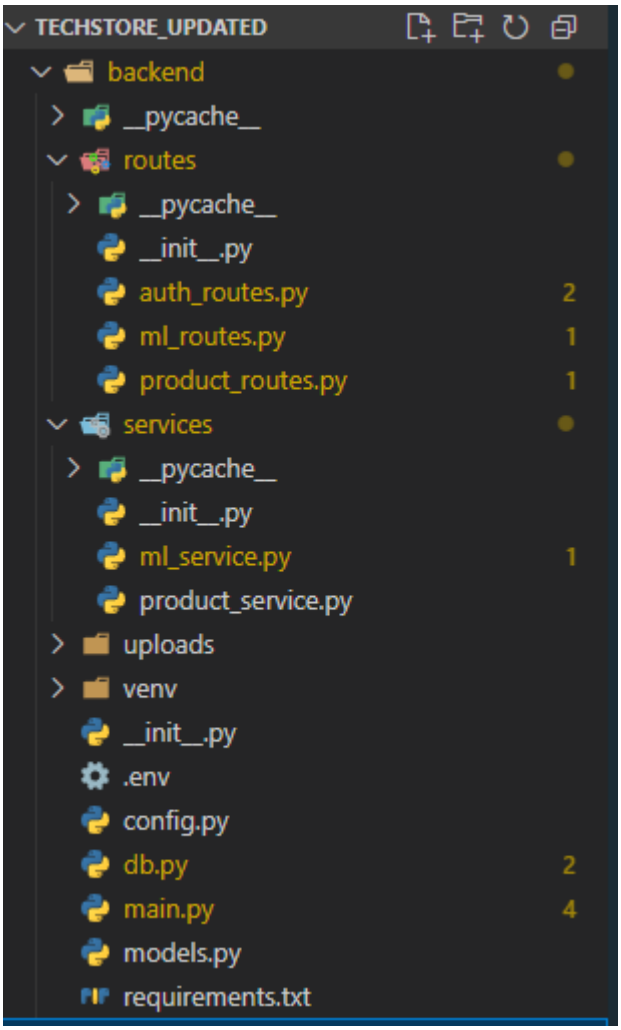

Рисунок 3.2 – Структура backend

З'єднання з базою даних. Для асинхронної роботи використовується бібліотека motor.

```
# db.py
from motor.motor_asyncio import AsyncIOMotorClient
from fastapi import FastAPI

MONGO_DETAILS = "mongodb://localhost:27017"
client = AsyncIOMotorClient(MONGO_DETAILS)
database = client.techstore

products_collection = database.get_collection("products")
```

```
users_collection = database.get_collection("users")
orders_collection = database.get_collection("orders")
reviews_collection = database.get_collection("reviews")
cart_collection = database.get_collection("carts")
```

Опис моделей даних (MongoDB documents). MongoDB дозволяє зберігати вкладені структури — наприклад, список товарів у замовленні зберігається прямо в документі замовлення, а відгуки — як окрема колекція.

Приклад моделі замовлення (order) у Pydantic.

```
from pydantic import BaseModel, Field
from typing import List

class OrderItem(BaseModel):
    product_id: str
    quantity: int
    price: float

class OrderCreate(BaseModel):
    user_id: str
    items: List[OrderItem]
    total_price: float
    status: str = "new"
    created_at: str
```

CRUD-операції з MongoDB (на прикладі створення замовлення).

```
# routes/orders.py
from fastapi import APIRouter, HTTPException
from db import orders_collection
from schemas import OrderCreate

router = APIRouter()
```

```
    @router.post("/orders/")
    async def create_order(order: OrderCreate):
        new_order = order.dict()
        result = await orders_collection.insert_one(new_order)
        if result.inserted_id:
            return { "id": str(result.inserted_id), **new_order }
        raise   HTTPException(status_code=500,   detail="Замовлення   не
створено")
```

Авторизація і захист. Взаємодія з MongoDB не впливає на реалізацію авторизації — як і раніше, для доступу до захищених ресурсів використовується JWT-токен.

Права користувача зберігаються у полі документу користувача (“is_admin”: true/false). Перевірка токена та прав відбувається аналогічно, як і для будь-якої іншої бази даних.

Пошук, фільтрація, агрегування. MongoDB дозволяє швидко фільтрувати товари за категоріями, брендами, ціною за допомогою відповідних запитів.

```
    # routes/products.py
    @router.get("/products/")
    async def get_products(category: str = None, brand: str = None):
        query = {}
        if category:
            query["category_id"] = category
        if brand:
            query["brand_id"] = brand
        products = await
products_collection.find(query).to_list(length=100)
    return products
```

Відгуки, рейтинг, корзина. Усі допоміжні колекції працюють за аналогічним принципом: CRUD-операції виконуються напряму над документами у відповідній колекції.

Наприклад, для кошика користувача достатньо зберігати document з user_id, масивом product_id та їхньою кількістю.

Використання MongoDB у тандемі з FastAPI робить бекенд максимально гнучким і готовим до швидких змін структури бізнес-логіки. Система легко масштабується, дозволяє працювати з великим обсягом неструктурованих чи вкладених даних (товари, замовлення, кошики), а також чудово підходить для інтернет-магазинів із частими оновленнями асортименту чи зміною товарних атрибутів.

3.4 Взаємодія front-end і back-end. Інтеграція з MongoDB

Ефективна взаємодія між клієнтською та серверною частинами є критично важливою для стабільної та швидкої роботи сучасного вебзастосунку електронної комерції. У розробленій системі комунікація між front-end (React) і back-end (FastAPI + MongoDB) реалізована через стандартизовані REST API, що гарантує швидкість, масштабованість та простоту інтеграції.

Механізм взаємодії: REST API. Усі основні дії користувача (отримання списку товарів, додавання у кошик, оформлення замовлення, аутентифікація, залишення відгуків) виконуються через HTTP(S)-запити до бекенд-ендпоінтів. Кожен запит із фронтенду надсилається у форматі JSON, а сервер повертає відповідь у тому ж форматі. Це дозволяє працювати з даними асинхронно, забезпечує оновлення інтерфейсу в реальному часі й дає змогу легко масштабувати систему — наприклад, додавати мобільні додатки чи сторонні інтеграції.

Основний "workflow" запитів. Користувач заходить на сайт, і React через Axios або Fetch звертається до бекенду для отримання каталогу товарів.

```
// services/api.js
import axios from "axios";

export const fetchProducts = async () => {
  const response = await axios.get("/api/products/");
```

```
    return response.data;
  };
```

Коли користувач додає товар у кошик, відбувається POST-запит із ідентифікатором товару й кількістю.

```
  export const addToCart = async (productId, quantity) => {
    return axios.post("/api/cart/add", { product_id: productId, qty: quantity });
  };
```

На бекенді FastAPI отримує цей запит, обробляє дані, проводить потрібні перевірки та записує/оновлює документ кошика в колекції MongoDB.

```
  # routes/cart.py
  @router.post("/cart/add")
  async def add_to_cart(data: dict, user=Depends(get_current_user)):
      user_id = user["_id"]
      product_id = data["product_id"]
      qty = data["qty"]
      cart = await cart_collection.find_one({"user_id": user_id})
      if not cart:
          # створити новий кошик
          await cart_collection.insert_one({"user_id": user_id, "items": [{"product_id": product_id, "qty": qty}]})
      else:
          # оновити існуючий кошик
          items = cart["items"]
          found = False
          for item in items:
              if item["product_id"] == product_id:
                  item["qty"] += qty
                  found = True
```

```
            if not found:
                items.append({"product_id": product_id, "qty": qty})
            await     cart_collection.update_one({"user_id":     user_id},
{"$set": {"items": items}})
        return {"message": "Товар додано у кошик"}
```

Аналогічно відбувається взаємодія для оформлення замовлення, перегляду профілю, відправки відгуку тощо. Усі зміни на сервері відразу фіксуються у відповідній колекції MongoDB — наприклад, оформлення замовлення створює новий документ у колекції “orders”, який містить масив товарів, інформацію про користувача, статус і дату.

Авторизація та захист запитів. Для всіх персональних дій (робота з кошиком, оформлення замовлення, додавання відгуків, доступ до профілю) front-end автоматично додає JWT-токен у заголовок Authorization при кожному запиті.

```
    axios.post("/api/orders/", orderData, {
      headers: { Authorization: `Bearer ${userToken}` }
    });
```

На сервері цей токен перевіряється — і тільки після успішної валідації запит дозволяється, що гарантує захист даних користувача.

Робота з MongoDB у бекенді. Завдяки MongoDB система не має жорстких схем і легко підтримує вкладені структури: наприклад, у замовленні прямо зберігається масив товарів, кожен з яких містить детальну інформацію (product_id, назва, кількість, ціна на момент покупки). Оновлення чи пошук здійснюється простими запитами.

```
    # routes/orders.py
    @router.get("/orders/")
    async def get_orders(user=Depends(get_current_user)):
        orders      =      await      orders_collection.find({"user_id":
user["_id"]}).to_list(length=100)
        return orders
```

Автоматичне оновлення даних у React. React-клієнт реагує на відповіді бекенду: після успішного додавання у кошик чи оформлення замовлення оновлюється стан додатку, виводяться відповідні повідомлення для користувача, перераховується сума, очищується кошик тощо.

Завдяки архітектурі "frontend–REST API–MongoDB backend" система працює швидко, асинхронно й безпроблемно масштабується під новий функціонал. Такий підхід забезпечує повну незалежність інтерфейсу від реалізації логіки, дозволяє розвивати front-end і back-end окремо, а MongoDB спрощує роботу з великими, гнучкими масивами даних, що характерно для магазинів комплектуючих.

3.5 Методи тестування вебзастосунку

Якість і стабільність роботи вебзастосунку мають вирішальне значення для довіри користувачів та успіху інтернет-магазину комплектуючих. Тому у розробленому проєкті впроваджено комплексний підхід до тестування, що охоплює як клієнтську, так і серверну частини системи, а також взаємодію із базою даних MongoDB.

Модульне тестування back-end. Серверна частина (FastAPI) тестувалася за допомогою бібліотеки pytest з використанням асинхронних клієнтів (наприклад, httpx). Для кожного API-ендпоінта (товари, замовлення, користувачі, кошик) були написані окремі тести, які перевіряють правильність обробки запитів, відповідність статус-кодів, валідацію даних та коректність роботи з MongoDB.

Приклад тесту для створення нового товару.

```
import pytest
from httpx import AsyncClient


@pytest.mark.asyncio
async def test_create_product():
    async with AsyncClient(app=app, base_url="http://test") as ac:
```

```
        response = await ac.post("/api/products/", json={
            "name": "NVIDIA RTX 4060",
            "category_id": "gpu",
            "brand_id": "nvidia",
            "price": 23500,
            "stock": 5,
            "description": "8GB GDDR6",
            "image_url": "https://site.ua/images/rtx4060.jpg"
        })
        assert response.status_code == 200
        data = response.json()
    assert data["name"] == "NVIDIA RTX 4060"
```

Інтеграційне тестування. Взаємодія front-end і back-end тестувалася шляхом перевірки “живого” обміну даними: від додавання товару в кошик до повного циклу оформлення замовлення й отримання підтвердження.

Використовувалися як інструменти розробника у браузері (DevTools), так і автоматизовані end-to-end тести з бібліотекою Cypress (або Playwright).

Приклад E2E-тесту для оформлення замовлення.

```
// cypress/integration/order.spec.js
describe('Order flow', () => {
  it('Користувач може додати товар і оформити замовлення', () => {
    cy.visit('/')
    cy.get('[data-testid="product-add-btn"]').first().click()
    cy.get('[data-testid="cart-btn"]').click()
    cy.get('[data-testid="checkout-btn"]').click()
    cy.get('input[name="name"]').type('Іван Петренко')
    cy.get('input[name="email"]').type('ivan@example.com')
    cy.get('input[name="address"]').type('Київ, вул. Миру, 1')
    cy.get('[data-testid="submit-order-btn"]').click()
    cy.contains('Дякуємо за замовлення!')
```

```
    })
})
```

Тестування front-end. Клієнтська частина тестувалася як вручну, так і автоматизовано. Для окремих компонентів застосовувалася бібліотека Jest разом із React Testing Library. Перевірялися коректність відображення елементів, обробка стану кошика, валідація форм і навігація між сторінками.

Фрагмент тесту компоненту кошика.

```
import { render, screen } from "@testing-library/react";
import Cart from "../components/Cart";

test("Відображає порожній кошик", () => {
  render(<Cart items={[]} />);
  expect(screen.getByText(/кошик порожній/i)).toBeInTheDocument();
});
```

Тестування безпеки. Окремо проводилася перевірка захищеності API: тестувалися сценарії неавторизованого доступу, спроби підробки JWT-токена, валідація вхідних даних, відсутність XSS/CSRF/SQL-ін'єкцій (наскільки це актуально для MongoDB). Також тестувалися обмеження прав користувача і адміністратора.

Тестування продуктивності. Для найкритичніших операцій (отримання списку товарів, оформлення замовлення) проводилося навантажувальне тестування з використанням Locust або аналогічних інструментів. Це дозволило виявити й оптимізувати вузькі місця в запитах до MongoDB та API.

Реалізована система тестування охоплює всі ключові аспекти життєвого циклу застосунку: від окремих компонентів до комплексної взаємодії між фронтендом, бекендом і базою даних. Це забезпечує стабільність, безпечність і прогнозовану поведінку магазину в умовах реального навантаження.

Висновки до розділу 3

У третьому розділі було детально розглянуто практичні аспекти розробки та впровадження вебзастосунку для електронної комерції з продажу комп'ютерних комплектуючих. На основі сформульованих у попередніх розділах вимог і обґрунтованого вибору технологій, реалізовано повноцінну систему, що відповідає сучасним стандартам якості, продуктивності й безпеки.

Було описано ключові функціональні можливості магазину: формування та керування каталогом товарів, зручний кошик, оформлення замовлень, особистий кабінет користувача, система відгуків, адміністративна панель для керування асортиментом і обробки замовлень. Окрему увагу приділено інтеграції front-end і back-end через REST API, що забезпечує швидку і надійну взаємодію між усіма компонентами системи та дозволяє масштабувати застосунок у майбутньому.

Завдяки використанню MongoDB дані зберігаються у гнучкому форматі, що полегшує адаптацію моделі під змінні бізнес-процеси і дозволяє ефективно працювати з великими обсягами інформації. FastAPI забезпечує високу продуктивність і простоту розширення API, а React — сучасний адаптивний інтерфейс для користувачів.

Особливу увагу в процесі реалізації було приділено питанням безпеки, авторизації, валідації даних та захисту від основних загроз.

Проведено комплексне тестування — як автоматизоване, так і інтеграційне — що дозволило виявити й усунути потенційні недоліки на ранніх етапах, забезпечивши стабільну роботу магазину.

Розроблений вебзастосунок повністю відповідає поставленим вимогам, є масштабованим, зручним для користувачів і адміністраторів, а також готовим до подальшого розвитку та інтеграції нових сервісів.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ ДЛЯ ЕЛЕКТРОННОЇ КОМЕРЦІЇ

4.1 Опис процесу розгортання вебзастосунку

Розгортання вебзастосунку складається з декількох послідовних етапів, кожен із яких важливий для забезпечення коректної роботи всієї системи. Процес включає підготовку серверного середовища, встановлення необхідного програмного забезпечення, налаштування бази даних MongoDB, запуск серверної (бекенд) і клієнтської (фронтенд) частин, а також перевірку доступності сервісів для користувача.

Перш за все, на сервер або комп'ютер, де планується запускати вебзастосунок, необхідно встановити Python версії не нижче 3.9, оскільки саме на цій мові реалізована back-end частина. Для цього можна скористатися офіційним сайтом python.org або стандартним менеджером пакетів для вашої операційної системи. Після встановлення Python рекомендується одразу встановити менеджер пакетів pip, який часто входить до стандартної поставки.

Далі потрібно встановити середовище виконання для клієнтської частини — Node.js. Для цього також слід завантажити останню LTS-версію Node.js з офіційного сайту nodejs.org. Після встановлення Node.js автоматично буде доступна утиліта npm, яка використовується для управління залежностями фронтенд-проєкту.

Окремо встановлюється база даних MongoDB. Для локальної роботи можна використати дистрибутив MongoDB Community Edition, який доступний для Windows, Linux та macOS. Після встановлення сервер бази даних запускається як окрема служба, і дані будуть зберігатися у спеціальній директорії на диску. Для тестового розгортання можна залишити стандартні параметри, для продуктивного — рекомендується змінити пароль адміністратора та обмежити мережевий доступ.

Після підготовки середовища потрібно налаштувати серверну частину проєкту. Спочатку створюється окрема директорія для back-end, у яку копіюється весь необхідний код.

У цій директорії через термінал або командний рядок створюється ізольоване віртуальне середовище командою `python -m venv venv`.

Далі віртуальне середовище активується Всі необхідні бібліотеки встановлюються командою `pip install -r requirements.txt`,

де requirements.txt — це файл зі списком залежностей проєкту (FastAPI, motor, pydantic, та інші).

У конфігураційному файлі (`.env` та `config.py`) вказується адреса підключення до MongoDB, секретний ключ для JWT та інші параметри середовища (наприклад, порт, email для адміністрування). Після налаштування можна запускати серверну частину командою `uvicorn main:app --reload`,

де `main:app` — це шлях до основного файлу додатку FastAPI.

Паралельно з цим у окремій директорії готується клієнтська частина (React). Сюди копіюються всі файли front-end частини. В терміналі або командному рядку виконується команда `npm install`, яка встановлює всі бібліотеки, необхідні для роботи застосунку (React, axios, tailwindcss тощо). У файлі конфігурації вказується адреса бекенду — звідки клієнтська частина буде отримувати дані через API (`REACT_APP_API_URL=http://localhost:8000`). Після цього проект запускається через `npm start`. За замовчуванням React-застосунок буде доступний за адресою http://localhost:3000.

На цьому етапі обидві частини проєкту працюють незалежно, але взаємодіють через HTTP-запити. Після запуску бекенду і фронтенду перевіряється доступність головної сторінки магазину, робота API (можна скористатися інтерактивною документацією FastAPI — /docs), а також працездатність підключення до MongoDB (наприклад, спробувати створити тестовий товар чи замовлення).

Процес розгортання не потребує складних інструментів чи спеціальних платформ, що робить проєкт універсальним і доступним для запуску як локально, так і на хостингу.

4.2 Керівництво користувача

Вебзастосунок розроблений так, щоб кожен користувач міг легко знаходити, переглядати та купувати комп’ютерні комплектуючі, не маючи спеціальних технічних знань. Головна сторінка магазину доступна за прямим посиланням у браузері. При першому відвідуванні користувача зустрічає зручний, інтуїтивний інтерфейс із сучасним дизайном, який автоматично підлаштовується під комп’ютер чи мобільний пристрій.

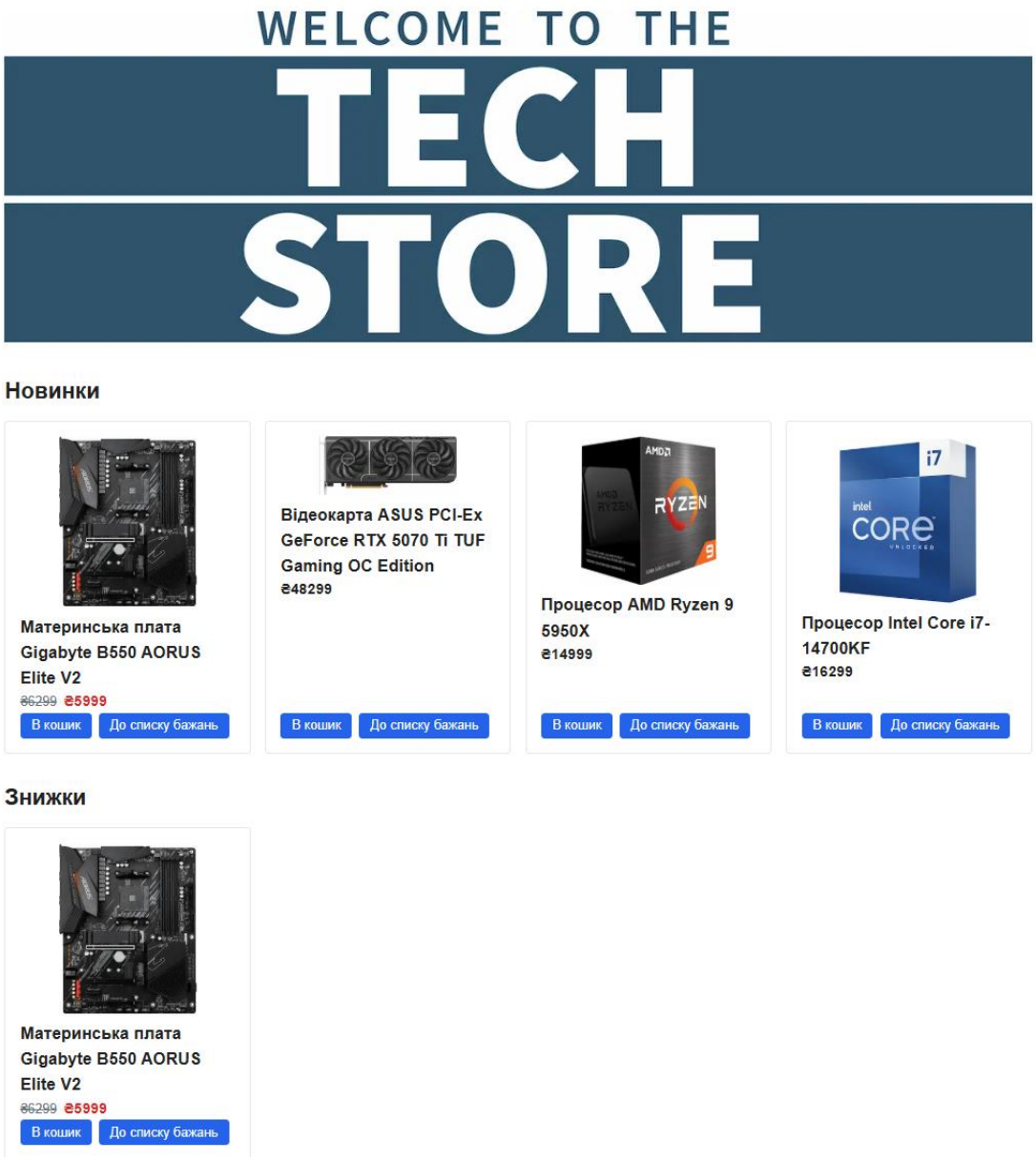

Рисунок 4.1 – Головна сторінка додатку

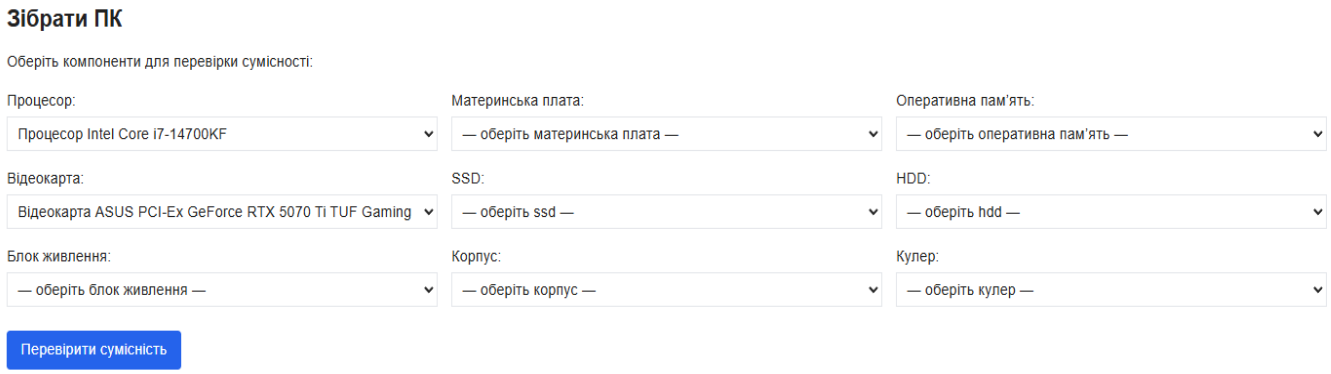

Рисунок 4.2 – Конфігуратор ПК з перевіркою на сумісність

Щоб ознайомитися з асортиментом, достатньо перейти у розділ “Каталог”. Тут доступний весь перелік комплектуючих, що розподілені за категоріями (наприклад, “Відеокарти”, “Процесори”, “Оперативна пам’ять” тощо). Для зручності реалізовано пошук за ключовими словами та система фільтрів: можна швидко відсортувати товари за виробником, ціною, обраною характеристикою чи наявністю на складі. Натисканням на будь-який товар відкривається його детальна сторінка із фотографіями, повним описом, характеристиками та актуальною ціною.

Коли користувач визначився з вибором, для додавання товару у кошик потрібно натиснути на відповідну кнопку (“Додати у кошик”). Система повідомить про успішне додавання та запропонує перейти до оформлення покупки або продовжити вибір. Для перегляду вмісту кошика слід натиснути на іконку “Кошик” у верхній частині сайту, після чого відкриється перелік усіх обраних позицій, кількість кожної та загальна сума.

Оформлення замовлення здійснюється у кілька простих кроків. На сторінці кошика натискається кнопка “Оформити замовлення”, після чого відкривається форма для введення контактних даних (ім’я, електронна пошта, номер телефону, адреса доставки). За необхідності користувач може обрати зручний спосіб оплати (онлайн-оплата карткою або післяплата) і бажану службу доставки. Після підтвердження замовлення система відобразить повідомлення про успішне оформлення, а на вказану електронну адресу буде надіслано підтвердження із деталями покупки.

Для постійних клієнтів передбачена можливість реєстрації та створення особистого кабінету. Після проходження короткої процедури реєстрації користувач отримує доступ до історії своїх замовлень, може відслідковувати статус відправлення, повторювати попередні покупки, залишати відгуки на придбані товари, а також змінювати свої контактні дані.

На будь-якій сторінці магазину доступна форма зворотного зв’язку для консультацій або вирішення питань. Якщо у користувача виникли труднощі під час пошуку чи оформлення замовлення, він може скористатися онлайн-чатом (якщо такий функціонал реалізовано) або написати повідомлення через форму підтримки — відповідь надійде на вказану електронну адресу.

Завдяки продуманому інтерфейсу й автоматичному керуванню основними процесами, користувач може швидко знайти необхідний товар, дізнатися всі деталі й оформити замовлення буквально за кілька хвилин. Навіть без реєстрації сайт залишається повністю функціональним — але створення облікового запису відкриває додаткові переваги, як-от збереження історії покупок чи персональні знижки.

4.3 Керівництво адміністратора

Адміністративна панель вебзастосунку створена для того, щоб відповідальні особи могли оперативно керувати асортиментом магазину, обробляти замовлення та підтримувати високу якість сервісу для покупців. Вхід до панелі адміністрування можливий лише після авторизації з правами адміністратора. Після успішного входу відкривається головна сторінка адміністративного інтерфейсу, яка відображає основну аналітику щодо поточного стану магазину: кількість нових замовлень, оновлені товари, статистику переглядів або продажів, якщо такий функціонал впроваджено.

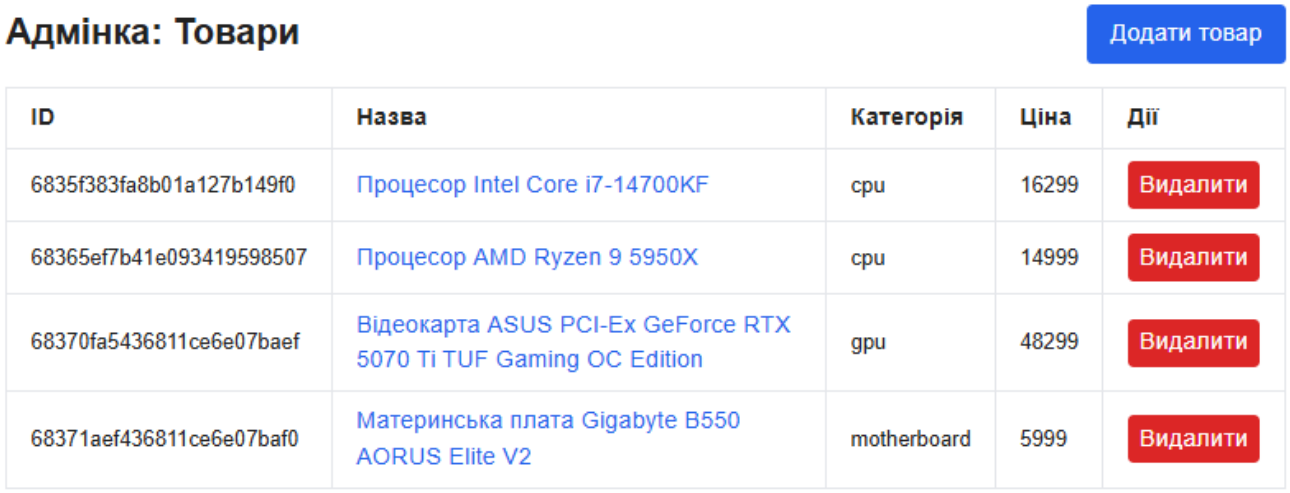

Рисунок 4.3 – Адмін панель

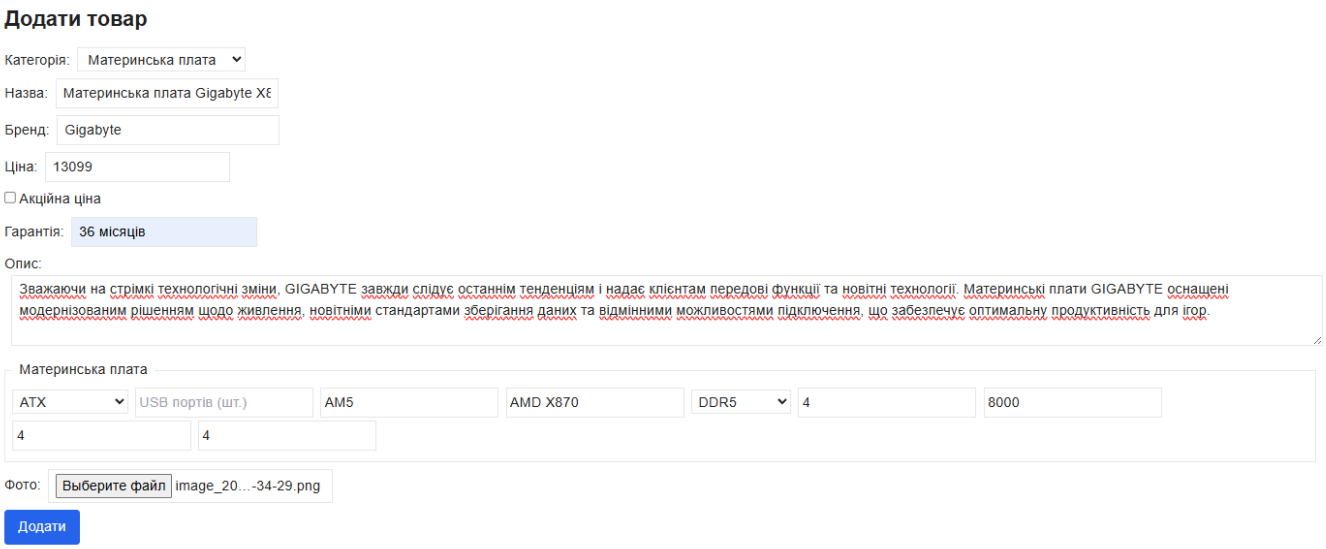

Рисунок 4.4 – Додавання товару на сайт

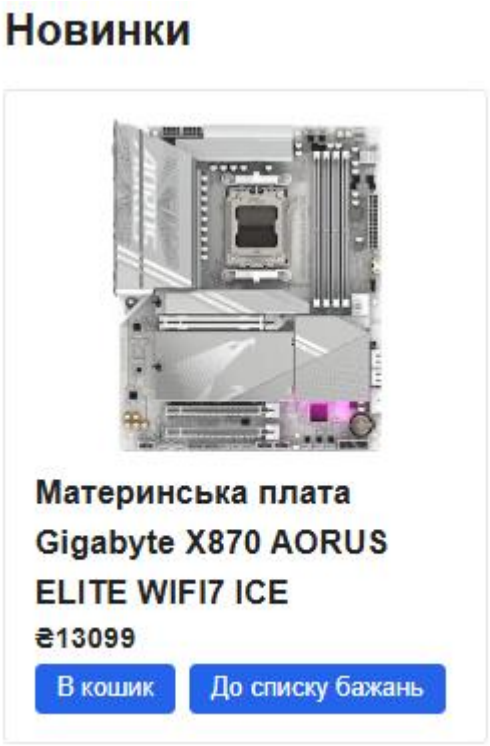

Рисунок 4.5 - Результат додавання товару на сайт

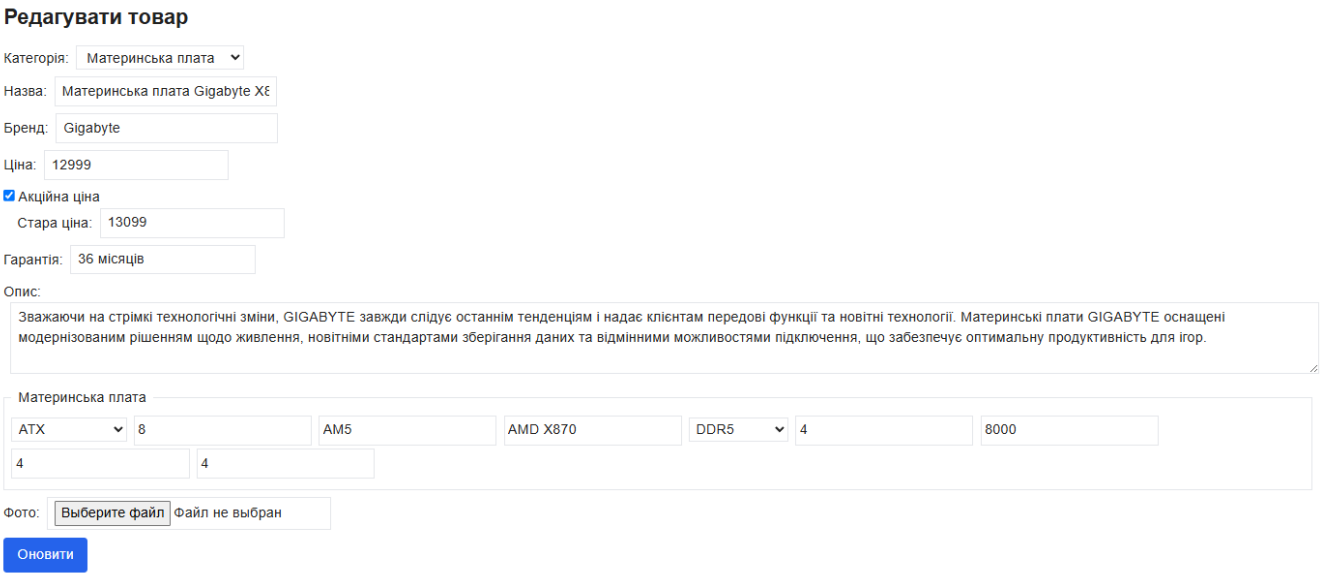

Рисунок 4.6 – Можливість редагування товару

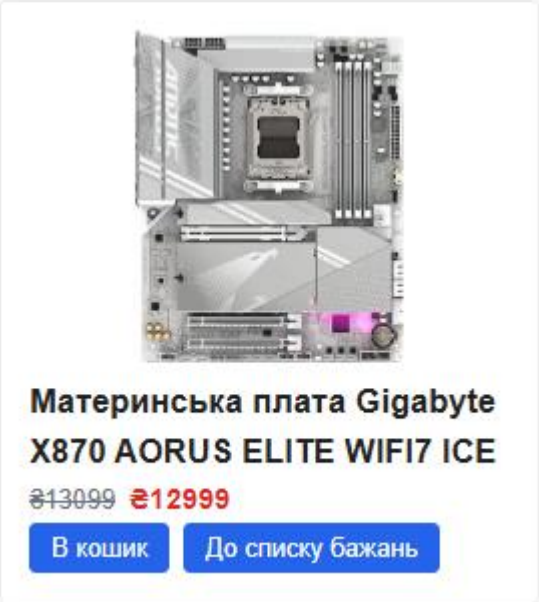

Рисунок 4.7 - Результат редагування товару

Основною функцією адміністратора є керування товарним каталогом. Через зручний інтерфейс можна додавати нові товари, вказуючи всі необхідні характеристики: назву, опис, категорію, бренд, ціну, залишок на складі, а також завантажувати фото для візуалізації на сторінці товару. Редагування вже існуючих позицій відбувається у кілька кліків — достатньо знайти потрібний товар у переліку, відкрити його сторінку та внести зміни. Також доступна можливість змінювати категорії, оновлювати списки брендів, приховувати або видаляти неактуальні товари.

Не менш важливою є робота з замовленнями. У відповідному розділі адміністратор бачить всі нові замовлення, детально ознайомлюється з інформацією про покупця, склад замовлення, обрану службу доставки та спосіб оплати. Система дозволяє змінювати статус замовлення (наприклад, “нове”, “у процесі обробки”, “відправлено”, “завершено”), залишати внутрішні коментарі чи передавати інформацію логістичній службі. Після виконання замовлення адміністратор може архівувати його для подальшої аналітики чи повторної обробки.

У разі надходження відгуків на товари, адміністратор має змогу модерувати їхній зміст. Це дає змогу підтримувати високу якість контенту на сайті, блокувати спам чи недоречні коментарі, а також видаляти образливі або неправдиві повідомлення. Якщо у платформі реалізовано систему “питань-відповідей” під товарами, адміністратор може відповідати на питання покупців напряму, підвищуючи якість сервісу.

Для забезпечення безпеки адміністративна панель підтримує багаторівневу авторизацію — тільки користувачі з відповідними правами можуть отримати доступ до редагування інформації або перегляду конфіденційних даних. Якщо потрібно, адміністратор може додавати нових співробітників у систему або обмежувати їхні повноваження, наприклад, дозволяти лише перегляд замовлень без права редагування товарів.

Весь функціонал адміністрування доступний із будь-якого пристрою — інтерфейс адаптовано під різні розширення екрана. Зміни, внесені адміністратором, відразу відображаються на головному сайті для покупців, що дає змогу підтримувати актуальність асортименту й оперативно реагувати на зміни ринку.

Завдяки зручному адміністративному інтерфейсу керування магазином не потребує глибоких технічних знань. Навіть при великій кількості товарів та замовлень адміністратор з легкістю контролює всі ключові процеси, підвищуючи ефективність роботи магазину та рівень обслуговування клієнтів.

4.4 Проведення функціонального тестування

Після завершення основних етапів розробки вебзастосунку особливу увагу було приділено функціональному тестуванню системи. Цей процес має на меті переконатися, що всі ключові можливості магазину працюють відповідно до вимог і забезпечують надійний користувацький досвід. Функціональне тестування охоплює як окремі модулі, так і комплексну взаємодію між ними — від початкового відображення каталогу до фіналізації покупки.

На першому етапі тестувалася робота основної навігації сайту. Перевірялося, чи всі посилання коректно ведуть на відповідні сторінки, чи швидко та коректно завантажується каталог товарів, чи відображаються категорії, бренди та ключові характеристики кожного продукту. Особливо важливим було перевірити фільтри й пошук: тестувалося введення різних запитів, застосування кількох фільтрів одночасно, а також сортування товарів за ціною, популярністю й наявністю.

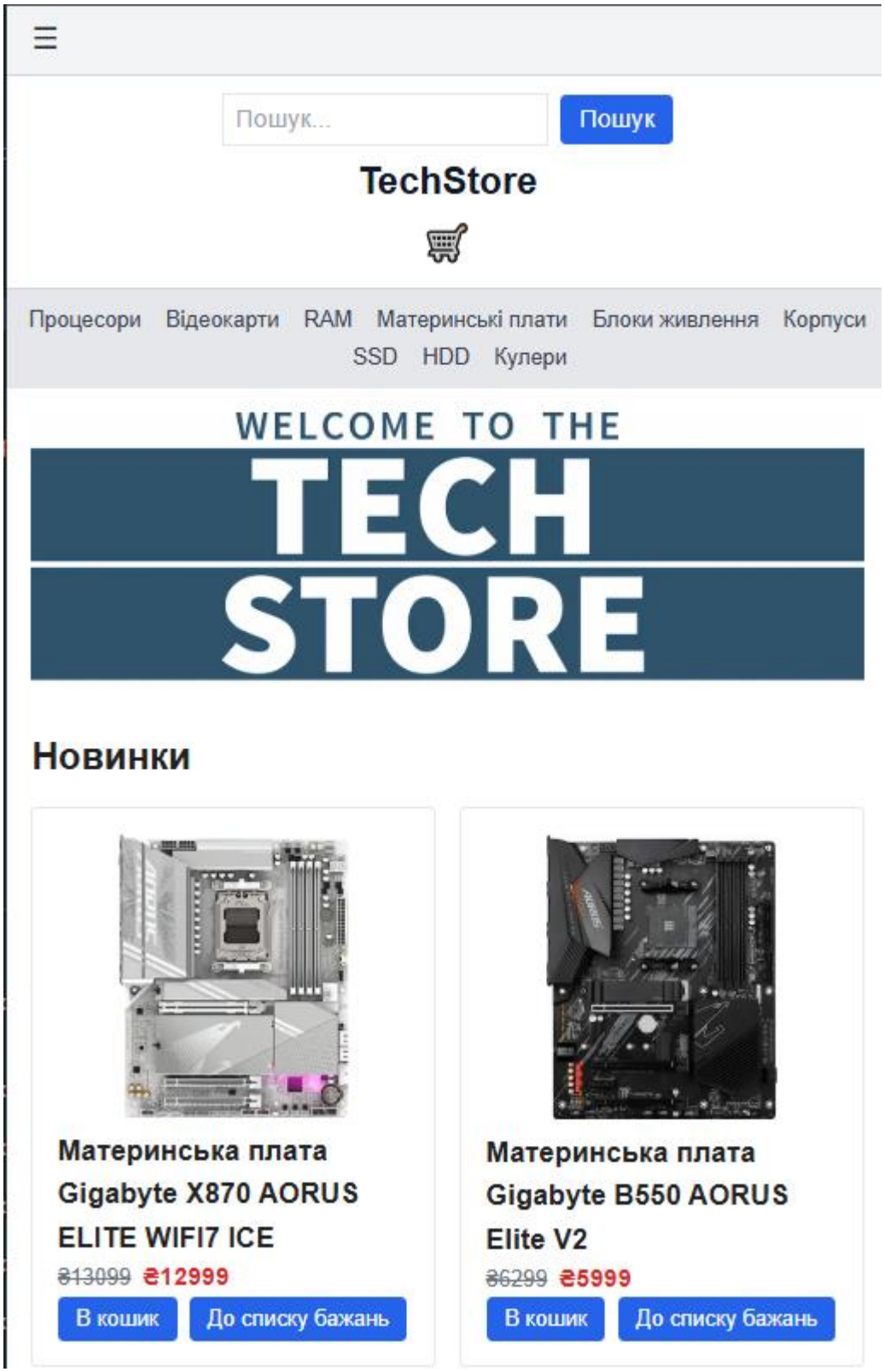

Рисунок 4.8 – Адаптивність сайту до мобільних пристроїв

Окремо приділялася увага процесу додавання товару в кошик. Випробовувалися різні сценарії — як із порожнім кошиком, так і з декількома товарами різних категорій. Перевірялося коректне відображення кількості товарів, правильний розрахунок загальної суми, можливість змінювати кількість чи видаляти окремі позиції з кошика. Оформлення замовлення тестувалося на предмет правильного збору контактних даних, валідації полів (наприклад, коректність e-mail, обов'язковість адреси доставки), а також обробки різних способів оплати й вибору служби доставки.

Також було проведено тестування процесу реєстрації та авторизації користувачів. Симулювалися спроби створити обліковий запис із валідними й невалідними даними, тестувалося відновлення пароля, перевірялися переходи між особистим кабінетом, історією замовлень і функціями залишення відгуків. Для зареєстрованих користувачів перевірявся доступ до історії покупок, можливість залишати коментарі на сторінках товарів і змінювати власні дані.

З боку адміністратора окремо тестувалися функції додавання, редагування й видалення товарів у каталозі, керування категоріями й брендами, обробка нових замовлень, зміна їх статусу та модерування відгуків. Важливим етапом була перевірка розмежування прав доступу — чи обмежується можливість внесення змін лише для адміністраторів, а звичайний користувач не має доступу до цих функцій.

Під час функціонального тестування використовувалися як ручні сценарії перевірки, так і автоматизовані тести — наприклад, для перевірки API-ендпоінтів або інтеграцій front-end із back-end. Всі критичні процеси — додавання товару, оформлення замовлення, авторизація, робота кошика — проходили перевірку на коректність виконання та відсутність неочікуваних помилок.

Підсумки функціонального тестування показали, що система виконує всі основні бізнес-процеси стабільно й передбачувано, а можливі незначні помилки легко ідентифікуються й оперативно усуваються завдяки продуманій структурі коду та інтерфейсу адміністрування.

4.5 Результати тестування та аналіз працездатності

Після проведення функціонального тестування всіх основних модулів вебзастосунку було підготовлено узагальнений аналіз працездатності та якості роботи системи. Результати тестування засвідчили, що всі основні функції магазину комплектуючих — від перегляду каталогу до оформлення замовлення й адміністрування — працюють стабільно, коректно та відповідають вимогам, поставленим на етапі проектування.

В першу чергу, була перевірена надійність роботи користувацького інтерфейсу. Система коректно відображає всі категорії й позиції товарів, швидко завантажує дані навіть при великій кількості запитів, а пошук і фільтрація працюють без помітних затримок. Навіть при одночасній роботі кількох користувачів не було виявлено зависань чи втрати продуктивності. Це свідчить про добру оптимізацію як front-end, так і back-end частини, а також про ефективну взаємодію з базою даних MongoDB.

У процесі тестування покупки було перевірено всі етапи оформлення замовлення: від додавання товару в кошик і введення контактних даних до вибору способу доставки та підтвердження оплати. Жодних критичних збоїв у процесі оформлення замовлень не зафіксовано. Система коректно обробляє помилки введення даних, попереджає користувача про некоректний формат адреси чи номера телефону, а також дозволяє повернутися на попередній крок для редагування інформації.

Особливу увагу приділено питанням безпеки — спроби несанкціонованого доступу до адміністративної панелі, редагування чужих замовлень або внесення змін у чужий кошик були успішно заблоковані завдяки коректній реалізації авторизації й розмежування прав доступу. Також були протестовані можливості відновлення пароля та зміни контактних даних через особистий кабінет користувача — всі сценарії відпрацьовують правильно, користувач отримує відповідні повідомлення й має можливість оперативно керувати своїми даними.

На рівні адміністрування магазином система дозволяє без затримок додавати, змінювати чи видаляти товари, обробляти замовлення та модерувати відгуки. Зміни одразу

відображаються на основному сайті, що дозволяє оперативно реагувати на потреби ринку чи запити клієнтів.

Крім функціонального тестування, проводилася базова перевірка продуктивності — система впевнено витримує паралельне навантаження кількох користувачів, не втрачає з'єднання з базою даних та не допускає дублювання або втрати інформації. Всі критичні дані захищені, а інтерфейс залишався стабільним навіть при багаторазових повторних діях.

В цілому, результати тестування підтвердили високу працездатність і надійність вебзастосунку в рамках заявлених функціональних і технічних вимог. Виявлені дрібні недоліки були своєчасно усунуті під час процесу тестування й не впливають на основні сценарії використання платформи. Таким чином, система готова до реальної експлуатації та може бути впроваджена для роботи з реальними користувачами.

Висновки до розділу 4

У четвертому розділі було детально описано всі етапи практичної реалізації та тестування вебзастосунку для електронної комерції комп'ютерних комплектуючих. Проведено послідовне розгортання системи — від встановлення програмного забезпечення та налаштування середовища до запуску серверної і клієнтської частин, підключення до бази даних MongoDB та перевірки працездатності всіх ключових компонентів.

На основі підготовлених інструкцій користувачі отримали зручний та інтуїтивний інтерфейс для вибору, замовлення та покупки товарів, а адміністратори — потужні інструменти для керування асортиментом, обробки замовлень і контролю за якістю обслуговування. Реалізований функціонал забезпечує повний цикл взаємодії — від додавання товарів у каталог до фінальної доставки продукту клієнту.

Особливу увагу було приділено проведенню функціонального тестування, яке охоплювало як окремі модулі, так і комплексну взаємодію між front-end, back-end і базою даних. Усі основні бізнес-процеси були ретельно перевірені: робота каталогу, кошика, оформлення замовлення, авторизація, залишення відгуків, адміністрування і модерація.

Результати тестування підтвердили стабільність та надійність системи, а також відповідність реалізованого функціоналу всім заявленим вимогам.

Програмна реалізація і впровадження вебзастосунку є успішними. Система демонструє високу працездатність, зручність для користувачів і адміністраторів, а також повністю готова до реального використання у сфері електронної комерції. Створене рішення можна рекомендувати для розгортання в умовах реального ринку з перспективою подальшого розвитку та масштабування.

ВИСНОВКИ

У ході виконання роботи було розроблено, впроваджено й протестовано повноцінний вебзастосунок для електронної комерції, орієнтований на продаж комп'ютерних комплектуючих. На основі детального аналізу предметної області, дослідження сучасних технологій та порівняння конкурентних рішень було обґрунтовано вибір оптимального стеку — React для клієнтської частини, FastAPI для серверної логіки та MongoDB для зберігання й обробки даних. Такий підхід дозволив створити гнучку, масштабовану й сучасну платформу, яка відповідає актуальним вимогам ринку електронної комерції.

У процесі проектування особливу увагу приділено архітектурі системи, структурі бази даних і організації взаємодії між основними модулями. Було розроблено всі необхідні інформаційні моделі, REST API для надійної комунікації між фронтендом і бекендом, а також зручний адаптивний інтерфейс для кінцевого користувача. Реалізовано механізми реєстрації, авторизації, захисту даних, оформлення й обробки замовлень, управління асортиментом, а також адміністрування і модерації.

Практична частина роботи охопила всі етапи — від підготовки та розгортання середовища до впровадження й налаштування системи в умовах тестового й робочого використання. Проведено комплексне функціональне тестування, яке підтвердило стабільність, працездатність і безпеку розробленої платформи. Всі основні сценарії — пошук і фільтрація товарів, робота з кошиком, оформлення замовлення, обробка платежів, управління замовленнями й відгуками — виконуються коректно, а можливі помилки і недоліки легко ідентифікуються та усуваються.

Завдяки використанню сучасних технологій та продуманій архітектурі, створена система готова до подальшого розвитку. Платформа дозволяє легко розширювати функціонал, інтегрувати додаткові сервіси (платіжні, логістичні, аналітичні), реалізовувати персоналізовані рекомендації та мобільні додатки. Система повністю відповідає сучасним стандартам безпеки, забезпечує захист персональних і фінансових даних користувачів, а також високу продуктивність при роботі з великими обсягами інформації.

Розроблений вебзастосунок може бути впроваджений у реальних умовах і є надійною основою для розвитку онлайн-бізнесу у сфері комп'ютерних комплектуючих. Отримані результати підтверджують актуальність і ефективність обраного підходу, а напрацьовані методики й архітектурні рішення можуть бути використані для створення аналогічних систем у суміжних галузях електронної комерції.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Офіційний сайт MongoDB [Електронний ресурс]. – Режим доступу: https://www.mongodb.com/, вільний. – Дата звернення: 03.03.2024.

2. Офіційна документація FastAPI [Електронний ресурс]. – Режим доступу: https://fastapi.tiangolo.com/, вільний. – Дата звернення: 17.02.2024.

3. Офіційна документація React [Електронний ресурс]. – Режим доступу: https://react.dev/, вільний. – Дата звернення: 22.04.2024.

4. Tailwind CSS: документація [Електронний ресурс]. – Режим доступу: https://tailwindcss.com/docs/, вільний. – Дата звернення: 04.05.2024.

5. Інтернет-магазин Rozetka: функціональні можливості [Електронний ресурс]. – Режим доступу: https://rozetka.com.ua/, вільний. – Дата звернення: 01.02.2024.

6. Інтернет-магазин Telemart: каталог комп'ютерних комплектуючих [Електронний ресурс]. – Режим доступу: https://telemart.ua/, вільний. – Дата звернення: 07.05.2024.

7. Prom.ua: аналітика та досвід користування маркетплейсом [Електронний ресурс]. – Режим доступу: https://prom.ua/, вільний. – Дата звернення: 10.03.2024.

8. Інформаційні технології у сфері електронної комерції: підручник / За ред. М.І. Згуровського. – К.: Видавничий дім «Політехніка», 2021. – 342 с.

9. Офіційна документація Axios [Електронний ресурс]. – Режим доступу: https://axios-http.com/docs/intro, вільний. – Дата звернення: 09.05.2024.

10. Пайдалова, І.В. Веб-програмування: навч. посіб. – Київ: КНУ, 2020. – 219 с.

11. REST API Design — Best Practices [Електронний ресурс]. – Режим доступу: https://restfulapi.net/, вільний. – Дата звернення: 12.03.2024.

12. Постановка задачі електронної комерції / Матеріали Нац. конф. ІТ-2023. – Львів, 2023. – С. 154–159.

13. OpenAI API documentation [Електронний ресурс]. – Режим доступу: https://platform.openai.com/docs/, вільний. – Дата звернення: 25.02.2024.

14. Django REST framework documentation [Електронний ресурс]. – Режим доступу: https://www.django-rest-framework.org/, вільний. – Дата звернення: 27.04.2024.

15. Node.js documentation [Електронний ресурс]. – Режим доступу: https://nodejs.org/en/docs/, вільний. – Дата звернення: 10.02.2024.

16. JavaScript: офіційна документація [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/uk/docs/Web/JavaScript, вільний. – Дата звернення: 19.03.2024.

17. GitHub: розміщення відкритого коду [Електронний ресурс]. – Режим доступу: https://github.com/, вільний. – Дата звернення: 05.05.2024.

18. Мальцева, Л.В. Електронна комерція: навчальний посібник. – Львів: ЛНУ імені Івана Франка, 2021. – 238 с.

19. Пащенко, О.М. Системи електронної комерції: теорія та практика. – Харків: ХНЕУ, 2020. – 184 с.

20. React Testing Library documentation [Електронний ресурс]. – Режим доступу: https://testing-library.com/docs/react-testing-library/intro/, вільний. – Дата звернення: 14.04.2024.

21. Pytest documentation [Електронний ресурс]. – Режим доступу: https://docs.pytest.org/en/stable/, вільний. – Дата звернення: 06.02.2024.

22. Марченко, Т.А. Основи проєктування веб-застосунків: навчальний посібник. – Київ: НАУ, 2022. – 167 с.

23. Cloud MongoDB Atlas documentation [Електронний ресурс]. – Режим доступу: https://www.mongodb.com/atlas, вільний. – Дата звернення: 28.03.2024.

24. Nginx: офіційна документація [Електронний ресурс]. – Режим доступу: https://nginx.org/ru/docs/, вільний. – Дата звернення: 16.04.2024.

25. Python: офіційна документація [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/, вільний. – Дата звернення: 02.05.2024.

ДОДАТОК А

Програмна реалізація

```
    from ..services import product_service
    from ..services import embedding_service
    from fastapi import HTTPException
    import numpy as np

    def to_number(val):
        if val is None:
            return None
        try:
            return int(val)
        except Exception:
            try:
                return float(val)
            except Exception:
                return None

    # Compatibility check helper functions
    def check_cpu_motherboard(cpu, motherboard):
        issues = []
        cpu_socket = cpu['attributes'].get('socket')
        mb_socket = motherboard['attributes'].get('socket')
        if cpu_socket and mb_socket and cpu_socket != mb_socket:
            issues.append(f"Процесор ({cpu['name']}) не підходить до сокета материнської плати ({motherboard['name']}).")
        return issues

    def check_cpu_cooler(cpu, cooler):
        issues = []
        cpu_socket = cpu['attributes'].get('socket')
        cooler_support = cooler['attributes'].get('socketSupport') or cooler['attributes'].get('socket_support')
        if not cooler_support:
            cooler_support = cooler['attributes'].get('socketSupport') or cooler['attributes'].get('socket_support')
        if cpu_socket and cooler_support:
            supports_list = []
            if isinstance(cooler_support, list):
                supports_list = [str(s).strip().upper() for s in cooler_support]
            elif isinstance(cooler_support, str):
                supports_list = [s.strip().upper() for s in cooler_support.split(',')]
            if str(cpu_socket).strip().upper() not in supports_list:
```

```
                issues.append(f"Кулер ({cooler['name']}) не підтримує
сокет мат. плати ({cpu_socket})")
        return issues

    def check_motherboard_case(motherboard, case):
        issues = []
        mb_form = motherboard['attributes'].get('formFactor') or
motherboard['attributes'].get('form_factor')
        case_form = case['attributes'].get('formFactor') or
case['attributes'].get('form_factor')
        if mb_form and case_form:
            # Normalize to same case
            mb_form_val = str(mb_form).strip().lower()
            case_form_val = str(case_form).strip().lower()
            form_compat = {
                'mini-itx': ['mini-itx', 'microatx', 'atx', 'e-atx'],
                'microatx': ['microatx', 'atx', 'e-atx'],
                'atx': ['atx', 'e-atx'],
                'e-atx': ['e-atx']
            }
            if mb_form_val in form_compat and case_form_val:
                # Check if case_form can accommodate mb_form
                if mb_form_val not in form_compat.get(case_form_val,
[]) and mb_form_val != case_form_val:
                    issues.append(f"Материнська плата форм-фактору
{motherboard['attributes'].get('formFactor')} може не поміститися в
корпус {case['name']} ({case['attributes'].get('formFactor')}).")
        return issues

    def check_gpu_case(gpu, case):
        issues = []
        gpu_length = to_number(gpu['attributes'].get('length'))
        max_gpu_length =
to_number(case['attributes'].get('maxGpuLength'))
        if gpu_length and max_gpu_length and gpu_length >
max_gpu_length:
            issues.append(f"Відеокарта ({gpu['name']}) зависока для
корпусу {case['name']} (макс. довжина {max_gpu_length} мм).")
        return issues

    def check_cooler_case(cooler, case):
        issues = []
        cooler_height = to_number(cooler['attributes'].get('height'))
or to_number(cooler['attributes'].get('maxCoolerHeight'))
        max_cooler_height =
to_number(case['attributes'].get('maxCoolerHeight'))
        if cooler_height and max_cooler_height and cooler_height >
max_cooler_height:
```

```
            issues.append(f"Кулер ({cooler['name']}) зависокий для
корпуса {case['name']} (макс. висота кулера {max_cooler_height} мм).")
        return issues

    def check_psu_power(psu, cpu=None, gpu=None):
        issues = []
        wattage = to_number(psu['attributes'].get('wattage'))
        # Estimate needed wattage from CPU TDP + GPU TDP
        needed = 0
        if cpu:
            needed += to_number(cpu['attributes'].get('tdp')) or 0
        if gpu:
            needed += to_number(gpu['attributes'].get('gpuTDP')) or 0
        if wattage and needed and wattage < needed:
            issues.append(f"Потужності БЖ недостатньо: потрібно
~{needed}Вт, а блок живлення видає лише {wattage}Вт.")
        return issues

    def check_ram_motherboard(ram, motherboard):
        issues = []
        ram_type = ram['attributes'].get('ramType')
        mb_ram_type = motherboard['attributes'].get('ramType')
        if ram_type and mb_ram_type and str(ram_type).strip().upper()
!= str(mb_ram_type).strip().upper():
            issues.append(f"Тип пам’яті ОЗП ({ram_type}) не
підтримується мат. платою ({mb_ram_type}).")
        return issues

    # Main compatibility check function
    async def check_compatibility(selected_parts: dict):
        parts = {}
        issues = []
        # Fetch full product docs for each selected component ID
        for cat, prod_id in selected_parts.items():
            if prod_id:
                product = await product_service.get_product(prod_id)
                if product:
                    parts[cat] = product

        # Run all compatibility checks:
        if parts.get('cpu') and parts.get('motherboard'):
            issues.extend(check_cpu_motherboard(parts['cpu'],
parts['motherboard']))
        if parts.get('cpu') and parts.get('cooler'):
            issues.extend(check_cpu_cooler(parts['cpu'],
parts['cooler']))
        if parts.get('motherboard') and parts.get('case'):
```

```
            issues.extend(check_motherboard_case(parts['motherboard'],
parts['case']))
        if parts.get('gpu') and parts.get('case'):
            issues.extend(check_gpu_case(parts['gpu'], parts['case']))
        if parts.get('cooler') and parts.get('case'):
            issues.extend(check_cooler_case(parts['cooler'],
parts['case']))
        # If motherboard and cooler are selected without a CPU, check
their socket compatibility
        if parts.get('motherboard') and parts.get('cooler') and not
parts.get('cpu'):
            cpu_sock =
parts['motherboard']['attributes'].get('socket')
            cooler_support =
parts['cooler']['attributes'].get('socketSupport') or
parts['cooler']['attributes'].get('socket_support')
            if cpu_sock and cooler_support:
                supports_list = []
                if isinstance(cooler_support, list):
                    supports_list = [str(s).strip().upper() for s in
cooler_support]
                elif isinstance(cooler_support, str):
                    supports_list = [s.strip().upper() for s in
cooler_support.split(',')]
                if str(cpu_sock).strip().upper() not in supports_list:
                    issues.append(f"Кулер ({parts['cooler']['name']})
не підтримує сокет мат. плати ({cpu_sock})")
        if parts.get('psu'):
            issues.extend(check_psu_power(parts['psu'],
cpu=parts.get('cpu'), gpu=parts.get('gpu')))
        if parts.get('ram') and parts.get('motherboard'):
            issues.extend(check_ram_motherboard(parts['ram'],
parts['motherboard']))

        # Return compatibility result
        return {"compatible": len(issues) == 0, "issues": issues}

    # Recommendation logic based on embeddings (vector similarity)
    async def get_recommendations(product_id: str):
        """Get product recommendations based on embedding
similarity."""
        # Ensure product exists
        product = await product_service.get_product(product_id)
        if not product:
            raise HTTPException(status_code=404, detail="Product not
found")
        # Ensure embeddings are up to date
        await embedding_service.ensure_embeddings()
```

```
        # Find index of target product in embeddings list
        target_idx = None
        for idx, prod in enumerate(embedding_service.products_data):
            if prod.get('_id') == product_id:
                target_idx = idx
                break
        if target_idx is None:
            # If product not in embeddings (e.g., just added),
recompute embeddings and try again
            await embedding_service.ensure_embeddings()
            for idx, prod in
enumerate(embedding_service.products_data):
                if prod.get('_id') == product_id:
                    target_idx = idx
                    break
        if target_idx is None:
            raise HTTPException(status_code=404, detail="Product not
found in embeddings")
        target_vector =
embedding_service.product_embeddings[target_idx]
        # Get candidate indices (same category as target, excluding
target itself)
        target_category = product.get('category')
        if target_category and target_category in
embedding_service.category_index_map:
            candidate_indices = [i for i in
embedding_service.category_index_map[target_category] if i !=
target_idx]
        else:
            candidate_indices = [i for i in
range(len(embedding_service.products_data)) if i != target_idx]
        if not candidate_indices:
            return []
        candidate_embeddings =
embedding_service.product_embeddings[candidate_indices]
        # Compute cosine similarity between target and each candidate
        target_norm = np.linalg.norm(target_vector)
        candidates_norms = np.linalg.norm(candidate_embeddings,
axis=1)
        if target_norm == 0:
            similarities = np.zeros(len(candidate_indices))
        else:
            similarities = (candidate_embeddings.dot(target_vector)) /
(candidates_norms * target_norm)
            similarities = np.nan_to_num(similarities)  # handle any
NaN/inf
        # Select top 5 most similar products
        top_n = 5
```

```
        top_idx_sorted = similarities.argsort()[::-1][:top_n]  #
indices in descending similarity order
        top_indices = [candidate_indices[i] for i in top_idx_sorted]
        # Retrieve product documents for recommended indices
        recommended_products = [embedding_service.products_data[i] for
i in top_indices]
        # Ensure _id is str
        for prod in recommended_products:
            if '_id' in prod and not isinstance(prod['_id'], str):
                prod['_id'] = str(prod['_id'])
        return recommended_products

    # Search products by query (semantic search using embeddings)
    async def search_products(query: str):
        """Search for products by query text using semantic
embeddings."""
        # Ensure embeddings are ready
        await embedding_service.ensure_embeddings()
        # Compute embedding for the query text
        query_vector = embedding_service.model.encode(query,
convert_to_numpy=True)
        # Normalize query vector
        q_norm = np.linalg.norm(query_vector)
        if q_norm == 0:
            # If empty or all-zero query vector
            return []
        query_vector_normed = query_vector / q_norm
        # Normalize all product embeddings (if not already normalized)
        product_embeddings = embedding_service.product_embeddings
        norms = np.linalg.norm(product_embeddings, axis=1,
keepdims=True)
        norms[norms == 0] = 1  # avoid division by zero
        embeddings_normed = product_embeddings / norms
        # Compute cosine similarities
        similarities = embeddings_normed.dot(query_vector_normed)
        # Get top results (e.g., top 10)
        top_n = min(10, similarities.shape[0])
        top_idx_sorted = similarities.argsort()[::-1][:top_n]
        results = [embedding_service.products_data[i] for i in
top_idx_sorted]
        # Ensure _id is str in results
        for prod in results:
            if '_id' in prod and not isinstance(prod['_id'], str):
                prod['_id'] = str(prod['_id'])
        return results

    # Chat-bot assistant for product queries (uses semantic search for
now)
```

```
    async def chat_assistant(query: str):
        """Answer user queries about products (semantic search)."""
        # For simplicity, reuse the search logic to find relevant
products
        return await search_products(query)
```

```
    import { useState, useEffect } from 'react';
    import Header from '@/components/Header';
    import Footer from '@/components/Footer';

    export default function BuildPage() {
        const [products, setProducts] = useState([]);
        const [selected, setSelected] = useState({
            cpu: null,
            motherboard: null,
            ram: null,
            gpu: null,
            ssd: null,
            hdd: null,
            psu: null,
            case: null,
            cooler: null
        });
        const [compatibility, setCompatibility] = useState(null);

        // Load all products for selection
        useEffect(() => {
            const fetchProducts = async () => {
                try {
                    const apiUrl = process.env.NEXT_PUBLIC_API_URL;
                    const res = await fetch(`${apiUrl}/products`);
                    const data = await res.json();
                    // The API returns an array of products in
data.product
                    setProducts(data.product || []);
                } catch (error) {
                    console.error('Error fetching products:', error);
                }
            };
            fetchProducts();
        }, []);

        const categories = [
            { key: 'cpu', label: 'Процесор' },
            { key: 'motherboard', label: 'Материнська плата' },
            { key: 'ram', label: 'Оперативна пам’ять' },
            { key: 'gpu', label: 'Відеокарта' },
            { key: 'ssd', label: 'SSD' },
```

```
            { key: 'hdd', label: 'HDD' },
            { key: 'psu', label: 'Блок живлення' },
            { key: 'case', label: 'Корпус' },
            { key: 'cooler', label: 'Кулер' }
        ];

        // Prepare selected product objects for compatibility checks
        const selectedProducts = {};
        products.forEach(p => {
            if (selected[p.category] && selected[p.category] ===
p._id) {
                selectedProducts[p.category] = p;
            }
        });

        const handleSelect = (category, productId) => {
            setSelected(prev => ({ ...prev, [category]: productId }));
            setCompatibility(null);  // Reset compatibility results
when selection changes
        };

        // Determine if a product option is compatible given current
selections
        const isCompatibleOption = (cat, product) => {
            const attrs = product.attributes || {};
            switch (cat) {
                case 'cpu':
                    // CPU compatible if socket matches selected
motherboard
                    if (selectedProducts.motherboard) {
                        const mbSocket =
selectedProducts.motherboard.attributes.socket;
                        if (attrs.socket && mbSocket && attrs.socket
!== mbSocket) return false;
                    }
                    // CPU compatible with selected cooler?
                    if (selectedProducts.cooler) {
                        const support =
selectedProducts.cooler.attributes.socketSupport;
                        if (attrs.socket && support) {
                            const supportsList =
Array.isArray(support)
                                ? support.map(s =>
s.trim().toUpperCase())
                                : String(support).split(',').map(s =>
s.trim().toUpperCase());
                            if
(!supportsList.includes(attrs.socket.toUpperCase())) return false;
```

```
                        }
                    }
                    return true;
                case 'motherboard':
                    // Motherboard compatible if socket matches
selected CPU
                    if (selectedProducts.cpu) {
                        const cpuSocket =
selectedProducts.cpu.attributes.socket;
                        if (attrs.socket && cpuSocket && attrs.socket
!== cpuSocket) return false;
                    }
                    // Motherboard compatible with selected RAM type
                    if (selectedProducts.ram) {
                        const ramType =
selectedProducts.ram.attributes.ramType;
                        if (attrs.ramType && ramType && attrs.ramType
!== ramType) return false;
                    }
                    // Motherboard fits in selected case
                    if (selectedProducts.case) {
                        const caseSize =
selectedProducts.case.attributes.formFactor;
                        const mbSize = attrs.formFactor;
                        if (caseSize && mbSize) {
                            const order = ['Mini-ITX', 'MicroATX',
'ATX', 'E-ATX'];
                            const caseIndex = order.indexOf(caseSize);
                            const mbIndex = order.indexOf(mbSize);
                            if (caseIndex !== -1 && mbIndex !== -1 &&
caseIndex < mbIndex) {
                                return false;
                            }
                        }
                    }
                    // M.2 slots needed for NVMe SSD
                    if (selectedProducts.ssd) {
                        const ssdInterface =
selectedProducts.ssd.attributes.interface;
                        const m2Slots = parseInt(attrs.m2Slots) || 0;
                        if (ssdInterface && ssdInterface.toUpperCase()
=== 'NVMe' && m2Slots < 1) return false;
                    }
                    // SATA ports needed for drives
                    let sataNeeded = 0;
                    if (selectedProducts.ssd) {
                        const ssdInterface =
selectedProducts.ssd.attributes.interface;
```

```
                        if (!ssdInterface ||
ssdInterface.toUpperCase() === 'SATA') sataNeeded += 1;
                    }
                    if (selectedProducts.hdd) {
                        sataNeeded += 1;
                    }
                    if (sataNeeded > 0) {
                        const sataPorts = parseInt(attrs.sataPorts) ||
0;
                        if (sataPorts < sataNeeded) return false;
                    }
                    return true;
                case 'ram':
                    // RAM compatible if type matches selected
motherboard
                    if (selectedProducts.motherboard) {
                        const mbRamType =
selectedProducts.motherboard.attributes.ramType;
                        if (attrs.ramType && mbRamType &&
attrs.ramType !== mbRamType) return false;
                    }
                    return true;
                case 'gpu':
                    // GPU is generally compatible (assuming a PCIe
slot is present on any standard motherboard)
                    return true;
                case 'ssd':
                    // SSD compatible if required slot is available on
selected motherboard
                    if (selectedProducts.motherboard) {
                        const interfaceType = attrs.interface;
                        const m2Slots =
parseInt(selectedProducts.motherboard.attributes.m2Slots) || 0;
                        const sataPorts =
parseInt(selectedProducts.motherboard.attributes.sataPorts) || 0;
                        if (interfaceType) {
                            if (interfaceType.toUpperCase() === 'NVMe'
&& m2Slots < 1) return false;
                            if (interfaceType.toUpperCase() === 'SATA'
&& sataPorts < 1) return false;
                        } else {
                            // assume SATA if not specified
                            if (sataPorts < 1) return false;
                        }
                    }
                    return true;
                case 'hdd':
```

```
                    // HDD compatible if motherboard has at least one
SATA port
                    if (selectedProducts.motherboard) {
                        const sataPorts =
parseInt(selectedProducts.motherboard.attributes.sataPorts) || 0;
                        if (sataPorts < 1) return false;
                    }
                    return true;
                case 'psu':
                    // PSU compatibility: always list (final check
will verify wattage/connectors)
                    return true;
                case 'case':
                    // Case compatible if it fits selected motherboard
form-factor
                    if (selectedProducts.motherboard) {
                        const mbSize =
selectedProducts.motherboard.attributes.formFactor;
                        const caseSize = attrs.formFactor;
                        if (caseSize && mbSize) {
                            const order = ['Mini-ITX', 'MicroATX',
'ATX', 'E-ATX'];
                            const caseIndex = order.indexOf(caseSize);
                            const mbIndex = order.indexOf(mbSize);
                            if (caseIndex !== -1 && mbIndex !== -1 &&
caseIndex < mbIndex) {
                                return false;
                            }
                        }
                    }
                    // Case compatibility with selected GPU length and
cooler height handled in final check
                    return true;
                case 'cooler':
                    // Cooler compatible if it supports socket of
selected CPU
                    if (selectedProducts.cpu) {
                        const cpuSocket =
selectedProducts.cpu.attributes.socket;
                        const support = attrs.socketSupport;
                        if (cpuSocket && support) {
                            const supportsList =
Array.isArray(support)
                                ? support.map(s =>
s.trim().toUpperCase())
                                : String(support).split(',').map(s =>
s.trim().toUpperCase());
```

```
                                if
(!supportsList.includes(cpuSocket.toUpperCase())) return false;
                            }
                        }
                        return true;
                    default:
                        return true;
                }
            };

            // Perform deep compatibility checks and collect issues
            const handleCheckCompatibility = () => {
                const issues = [];

                // Required selections presence
                if (!selectedProducts.cpu) issues.push('Не обрано
процесор.');
                if (!selectedProducts.motherboard) issues.push('Не обрано
материнську плату.');
                if (!selectedProducts.ram) issues.push('Не обрано
оперативну пам’ять.');
                if (!selectedProducts.psu) issues.push('Не обрано блок
живлення.');
                if (!selectedProducts.case) issues.push('Не обрано
корпус.');
                // A cooler is required for any CPU
                if (selectedProducts.cpu && !selectedProducts.cooler) {
                    issues.push('Не обрано кулер для процесора.');
                }
                // At least one storage device required
                if (!selectedProducts.ssd && !selectedProducts.hdd) {
                    issues.push('Не обрано жодного накопичувача (SSD або
HDD).');
                }

                // CPU vs Motherboard socket
                if (selectedProducts.cpu && selectedProducts.motherboard)
{
                    const cpuSocket =
selectedProducts.cpu.attributes.socket;
                    const mbSocket =
selectedProducts.motherboard.attributes.socket;
                    if (cpuSocket && mbSocket && cpuSocket !== mbSocket) {
                        issues.push('Сокет процесора не співпадає з
сокетом материнської плати.');
                    }
                }
                // CPU vs Cooler support
```

```
            if (selectedProducts.cpu && selectedProducts.cooler) {
                const cpuSocket =
selectedProducts.cpu.attributes.socket;
                const support =
selectedProducts.cooler.attributes.socketSupport;
                if (cpuSocket && support) {
                    const supportsList = Array.isArray(support)
                        ? support.map(s => s.trim().toUpperCase())
                        : String(support).split(',').map(s =>
s.trim().toUpperCase());
                    if
(!supportsList.includes(cpuSocket.toUpperCase())) {
                        issues.push('Обраний кулер не підтримує сокет
процесора.');
                    }
                }
            }
            // Motherboard vs RAM
            if (selectedProducts.motherboard && selectedProducts.ram)
{
                const mbRamType =
selectedProducts.motherboard.attributes.ramType;
                const ramType =
selectedProducts.ram.attributes.ramType;
                if (mbRamType && ramType && mbRamType !== ramType) {
                    issues.push('Тип пам’яті ОЗП не підтримується
материнською платою.');
                }
                const maxFreq =
parseInt(selectedProducts.motherboard.attributes.maxRamFrequency) || 0;
                const ramFreq =
parseInt(selectedProducts.ram.attributes.ramFrequency) || 0;
                if (maxFreq && ramFreq && ramFreq > maxFreq) {
                    issues.push(`Частота оперативної пам’яті
(${ramFreq} МГц) перевищує максимальну, підтримувану материнською
платою (${maxFreq} МГц).`);
                }
            }
            // Motherboard vs Case form-factor
            if (selectedProducts.motherboard && selectedProducts.case)
{
                const mbSize =
selectedProducts.motherboard.attributes.formFactor;
                const caseSize =
selectedProducts.case.attributes.formFactor;
                if (mbSize && caseSize) {
                    const order = ['Mini-ITX', 'MicroATX', 'ATX', 'E-
ATX'];
```

```
                    const mbIndex = order.indexOf(mbSize);
                    const caseIndex = order.indexOf(caseSize);
                    if (mbIndex !== -1 && caseIndex !== -1 &&
caseIndex < mbIndex) {
                        issues.push('Форм-фактор материнської плати не
поміщається в обраний корпус.');
                    }
                }
            }
            // GPU vs Case dimensions
            if (selectedProducts.gpu && selectedProducts.case) {
                const gpuLen =
parseFloat(selectedProducts.gpu.attributes.length) || 0;
                const caseMaxLen =
parseFloat(selectedProducts.case.attributes.maxGpuLength) || 0;
                if (gpuLen && caseMaxLen && gpuLen > caseMaxLen) {
                    issues.push('Довжина вибраної відеокарти перевищує
максимальну, підтримувану корпусом.');
                }
            }
            // Cooler vs Case height
            if (selectedProducts.cooler && selectedProducts.case) {
                const coolerHeight =
parseFloat(selectedProducts.cooler.attributes.height) || 0;
                const caseMaxHeight =
parseFloat(selectedProducts.case.attributes.maxCoolerHeight) || 0;
                if (coolerHeight && caseMaxHeight && coolerHeight >
caseMaxHeight) {
                    issues.push('Висота вибраного кулера перевищує
максимально допустиму для корпусу.');
                }
            }
            // PSU vs GPU power connectors
            if (selectedProducts.gpu && selectedProducts.psu) {
                const gpuConnectors =
parseInt(selectedProducts.gpu.attributes.powerConnectors) || 0;
                const psuConnectors =
parseInt(selectedProducts.psu.attributes.pcieConnectors) || 0;
                if (gpuConnectors > 0 && psuConnectors > 0 &&
gpuConnectors > psuConnectors) {
                    issues.push('Блок живлення не має достатньої
кількості PCIe-роз’ємів для живлення відеокарти.');
                }
            }
            // PSU vs Power requirements
            if (selectedProducts.psu) {
                let totalRequired = 0;
                if (selectedProducts.cpu) {
```

```
                    totalRequired +=
parseInt(selectedProducts.cpu.attributes.tdp) || 0;
                }
                if (selectedProducts.gpu) {
                    const gpuReq =
parseInt(selectedProducts.gpu.attributes.gpuPowerRequirement) || 0;
                    totalRequired += gpuReq ||
(parseInt(selectedProducts.gpu.attributes.gpuTDP) || 0);
                }
                // Add some buffer for other components
                if (selectedProducts.ram) totalRequired += 10;
                if (selectedProducts.ssd) totalRequired += 5;
                if (selectedProducts.hdd) totalRequired += 5;
                if (selectedProducts.cooler) totalRequired += 5;
                const psuWattage =
parseInt(selectedProducts.psu.attributes.wattage) || 0;
                if (psuWattage && totalRequired && psuWattage <
totalRequired) {
                    issues.push(`Потужність блоку живлення
(${psuWattage}Вт) може бути недостатньою для вибраної конфігурації
(потребує ~${totalRequired}Вт).`);
                }
            }
            // CPU with no GPU scenario
            if (selectedProducts.cpu && !selectedProducts.gpu) {
                const hasIG =
selectedProducts.cpu.attributes.hasIntegratedGraphics;
                if (!hasIG) {
                    issues.push('Процесор не має вбудованої графіки, а
дискретна відеокарта не обрана.');
                }
            }

            setCompatibility({
                compatible: issues.length === 0,
                issues
            });
        };

        return (
            <>
                <Header />
                <main className="container mx-auto p-4">
                    <h1 className="text-2xl font-semibold mb-
4">Зібрати ПК</h1>
                    <p className="mb-4">Оберіть компоненти для
перевірки сумісності:</p>
```

```
                    <div className="grid grid-cols-1 md:grid-cols-2
lg:grid-cols-3 gap-4">
                        {categories.map(cat => (
                            <div key={cat.key} className="flex flex-
col">
                                <label className="font-medium mb-
1">{cat.label}:</label>
                                <select
                                    onChange={(e) =>
handleSelect(cat.key, e.target.value)}
                                    value={selected[cat.key] || ''}
                                    className="border p-2"
                                >
                                    <option value="">— оберіть
{cat.label.toLowerCase()} —</option>
                                    {products.filter(p => p.category
=== cat.key).map(product => (
                                        <option
                                            key={product._id}
                                            value={product._id}
                                            disabled={!isCompatibleOpt
ion(cat.key, product)}
                                        >
                                            {product.name}
                                        </option>
                                    ))}
                                </select>
                            </div>
                        ))}
                    </div>
                    <button
                        onClick={handleCheckCompatibility}
                        className="mt-6 px-4 py-2 bg-blue-600 text-
white rounded"
                    >
                        Перевірити сумісність
                    </button>

                    {compatibility && (
                        <div className="mt-6 p-4 border rounded">
                            {compatibility.compatible ? (
                                <p className="text-green-600 font-
semibold">Всі обрані компоненти сумісні.</p>
                            ) : (
                                <>
                                    <p className="text-red-600 font-
semibold mb-2">Знайдено проблеми сумісності:</p>
```

```
                                        <ul className="list-disc list-
inside text-red-600">
                                            {compatibility.issues.map((iss
ue, index) => (
                                                <li
key={index}>{issue}</li>
                                            ))}
                                        </ul>
                                    </>
                                )}
                            </div>
                        )}
                    </main>
                    <Footer />
                </>
            );
        }
```