Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка інтерактивного вебдодатку конфігуратора персонального комп'ютера»

Виконав: студент 4 курсу
групи 2ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Ткачук В. Ю.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Рейда О. М.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Томчук М. А.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н._____
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Ткачуку Володимиру Юрійовичу

1. Тема роботи – «Розробка інтерактивного вебдодатку конфігуратора персонального комп'ютера»

Керівник роботи: Рейда Олександр Миколайович, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. № 97.

2. Строк подання студентом роботи 13 червня 2025 р.

3. Вихідні дані до роботи: мови програмування JavaScript, фреймворк React для клієнтської частини, Node.js + Express для серверної логіки, система керування базами даних PostgreSQL, бібліотеки Redux Toolkit, Axios, JWT, середовище розробки Visual Studio Code, система контролю версій Git.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз стану розвитку додатків для купівлі та продажу комплектуючих; розробка моделі та алгоритмів роботи системи; розробка програмного застосунку; тестування застосунку; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження, наукова новизна, практична цінність

одержаних результатів; порівняльний аналіз аналогів; інтерфейсні макети, приклади конфігурацій ПК.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Рейда О.М., к.т.н., доц. кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану розвитку додатків для купівлі та продажу комплектуючих	25.03.25 – 04.05.25	вик.
2	Розробка моделі та алгоритмів роботи системи	05.04.25 – 17.04.25	вик.
3	Варіантний аналіз та вибір мови програмування	18.04.25 – 21.04.25	вик.
4	Розробка програмного застосунку	22.04.25 – 16.05.25	вик.
5	Тестування застосунку	17.05.25 – 25.05.25	вик.
6	Оформлення матеріалів до захисту БКР	25.05.25 – 30.05.25	вик.

Студент _____ Ткачук В.Ю.
(підпис) (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи _____ Рейда О.М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.415:004.738.5

Ткачук В.Ю. Розробка інтерактивного вебдодатку конфігуратора персонального комп’ютера: бакалаврська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця : ВНТУ, 2025. 88 с.

У бакалаврській кваліфікаційній роботі представлено розробку інтерактивного вебзастосунку конфігуратора персонального комп’ютера. Застосунок призначено для автоматизації процесу підбору сумісних комп’ютерних комплектуючих відповідно до потреб користувача. Основною функціональністю є формування конфігурацій ПК із врахуванням технічної сумісності, розрахунком загальної вартості та можливістю збереження обраної збірки. Вебзастосунок реалізовано з використанням JavaScript-технологій: React для клієнтської частини, Node.js – для серверної логіки, PostgreSQL – як системи управління базами даних. Особливу увагу приділено зручності інтерфейсу, інтерактивності та адаптивності системи, що дозволяє ефективно взаємодіяти з платформою як новачкам, так і досвідченим користувачам. Розроблене рішення може бути інтегроване до онлайн-магазинів або використовуватись як окремий сервіс для індивідуального конфігурування ПК.

ANNOTATION

UDC 004.415:004.738.5

Tkachuk V.Y. *Development of an Interactive Web Application for a Personal Computer Configurator*: Bachelor's Qualification Work in the specialty 121 Software Engineering, educational program – Software Engineering. Vinnytsia: VNTU, 2025. 88 p.

This bachelor’s qualification work presents the development of an interactive web application for configuring a personal computer. The application is designed to automate the process of selecting compatible computer components based on the user’s needs. Its core functionality includes the creation of PC configurations with automatic compatibility checks, total cost calculation, and the ability to save the selected build. The system is implemented using modern JavaScript technologies: React for the client side, Node.js for the server side, and PostgreSQL as the database management system. Special attention is paid to interface convenience, interactivity, and adaptability, enabling effective interaction with the platform for both novice and advanced users. The developed solution can be integrated into online marketplaces or used as a standalone service for personalized PC configuration.

ЗМІСТ

ВСТУП 4
1 АНАЛІЗ СТАНУ РОЗВИТКУ ДОДАТКІВ ДЛЯ КУПІВЛІ ТА ПРОДАЖУ КОМПЛЕКТУЮЧИХ 7
1.1 Аналіз стану питання розробки 7
1.2 Порівняльний аналіз аналогів 8
1.3 Аналіз методів розв’язання задачі 12
1.4 Постановка задач дослідження 13
1.5 Висновки 14
2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ РОБОТИ СИСТЕМИ 15
2.1 Аналіз інформаційного забезпечення 15
2.2 Розробка моделі застосунку 17
2.3 Розробка структури бази даних 20
2.4 Розробка алгоритмів роботи застосунку 23
2.5 Висновки 31
3 РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ 33
3.1 Варіантний аналіз та вибір мови програмування 33
3.2 Реалізація клієнтської частини 34
3.3 Реалізація серверної частини 37
3.4 Висновки 41
4 ТЕСТУВАННЯ ЗАСТОСУНКУ 43
4.1 Аналіз методів тестування 43
4.2 Тестування застосунку 48
4.3 Висновки 55
ВИСНОВКИ 57
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 58
ДОДАТКИ 60
Додаток А – ТЕХНІЧНЕ ЗАВДАННЯ ..... **Помилка! Закладку не визначено.**
Додаток Б – ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ 63

Додаток В – ЛІСТИНГ ПРОГРАМИ..... 65
Додаток Г – ІЛЮСТРАТИВНА ЧАСТИНА 81

ВСТУП

Обґрунтування вибору теми дослідження.

У сучасному світі розвиток інформаційних технологій значно вплинув на спосіб взаємодії користувачів з технікою, зокрема у сфері підбору та придбання комп’ютерного обладнання. Із зростанням популярності індивідуальних збірок ПК, користувачі дедалі частіше прагнуть самостійно формувати конфігурацію персонального комп’ютера відповідно до своїх потреб: для роботи, навчання, ігор чи професійної діяльності. При цьому важливими є не лише технічна сумісність компонентів, а й зручність процесу конфігурації, що робить актуальним використання спеціалізованих вебзастосунків-конфігураторів.

Попри наявність окремих платформ, які частково реалізують можливість підбору комплектуючих, більшість із них мають технічні обмеження, що ускладнюють їхнє використання в якості повноцінного конфігуратора. Зокрема, спостерігається відсутність автоматизованої перевірки технічної сумісності компонентів, низький рівень інтеграції з базами постачальників, що ускладнює актуалізацію цін та наявності в режимі реального часу та відсутність шаблонів формування запитів. Деякі платформи не підтримують динамічне оновлення інтерфейсу при зміні вибраних компонентів, що знижує продуктивність системи. Це створює значні труднощі при побудові персоналізованих рішень. Як наслідок, зростає потреба у розробці інтерактивного інструменту, що дозволяє швидко сформувати специфікацію ПК на основі заданих параметрів.

Таким чином, розробка сучасного вебзастосунку конфігуратора персонального комп’ютера є актуальним завданням, що дозволяє поєднати гнучкість налаштувань, зручність взаємодії та автоматичну перевірку технічної сумісності комплектуючих.

Зв'язок роботи з науковими програмами, планами, темами.

Робота виконувалася згідно з планом виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження.

Метою бакалаврської кваліфікаційної роботи є підвищення ефективності формування конфігурації та якості рекомендації, щодо оптимального складу ПК, відповідно до вимог користувача, що формуються в інтерактивному вебзастосунку конфігуратора персонального комп'ютера.

Основними задачами дослідження є:

- провести аналіз існуючих рішень у сфері ПК-конфігураторів;
- розробити архітектуру вебзастосунку;
- модифікувати метод перевірки сумісності комп'ютерних компонентів;
- розробити інтуїтивно зрозумілий інтерфейс користувача;
- розробити механізм збереження та експорту шаблонів конфігурацій;
- провести тестування системи.

Об'єкт дослідження – процес розробки спеціалізованого вебзастосунку маркетплейсу для конфігурування комп'ютерних систем.

Предмет дослідження – методи і засоби розробки вебзастосунків для індивідуального підбору комп'ютерних комплектуючих з урахуванням технічної сумісності та функціональних вимог.

Методи дослідження.

У роботі використано методи теорії прийняття рішень для обґрунтування вибору компонентів з урахуванням пріоритетів користувача, теорії баз даних для проектування реляційної моделі бази даних, функціональний аналіз, комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів.

Модифіковано метод перевірки сумісності комп'ютерних компонентів, який на відміну від існуючих базується на використанні вагових коефіцієнтів при формалізації технічних параметрів з метою оптимізації процесу підбору та підвищення якості конфігурації ПК.

Практична цінність отриманих результатів.

Розроблений застосунок може бути використаний як кінцевими користувачами, так і компаніями у сфері комп’ютерного продажу, для надання клієнтам зручного інструменту онлайн-конфігурації. Він може бути інтегрований у маркетплейси або використаний як окрема SaaS-платформа.

Особистий внесок здобувача.

Усі етапи розробки, включаючи аналіз, проєктування, реалізацію й тестування застосунку, виконані автором самостійно.

Апробація матеріалів бакалаврської кваліфікаційної роботи.

Результати роботи було представлено на Міжнародній науково-практичній конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)».

Публікації.

Основні результати дослідження опубліковано в матеріалах конференції «МН-2025» [1].

Аналіз.

У пояснювальній записці до бакалаврської кваліфікаційної роботи було розглянуто 4 розділи та використано 20 літературних джерел.

1 АНАЛІЗ СТАНУ РОЗВИТКУ ДОДАТКІВ ДЛЯ КУПІВЛІ ТА ПРОДАЖУ КОМПЛЕКТУЮЧИХ

1.1 Аналіз стану питання розробки

На сьогоднішній день розробка вебдодатків у сфері електронної комерції є одним із найактуальніших напрямків у галузі програмної інженерії. Згідно з останніми звітами аналітичних компаній, обсяги онлайн продажів комп'ютерної техніки стабільно зростають, а більшість користувачів віддають перевагу інтернет-магазинам, які дозволяють швидко знайти потрібний товар, порівняти характеристики, прочитати відгуки та замовити доставку [2].

Серед існуючих платформ можна виокремити великі універсальні маркетплейси на кшталт Rozetka, Hotline, Foxtrot, а також більш гнучкі системи, такі як OLX або Prom.ua, що надають продавцям більше свободи у публікації оголошень та управлінні асортиментом. Проте, попри високу популярність цих платформ, більшість з них не спеціалізуються виключно на продажу комп'ютерних комплектуючих, що ускладнює навігацію та пошук потрібного товару для цільової аудиторії IT-спеціалістів, геймерів, комп'ютерних майстрів та ентузіастів.

Існуючі системи часто мають такі недоліки:

- недостатньо детальну категоризацію товарів
- перевантажений або застарілий інтерфейс;
- відсутність фокусування на технічному сегменті, що призводить до великої кількості нецільових товарів;
- складність у роботі з боку продавця, а саме обмежені можливості для керування асортиментом, складна модерація або високі комісії.

У свою чергу, популярність кастомних рішень зростає. У мережі можна знайти багато фреймворків і шаблонів для створення власних маркетплейсів на основі Node.js, React, Next.js, Django, Laravel, однак їх інтеграція часто вимагає значних зусиль для налаштування під конкретну предметну область, особливо якщо вона потребує специфічної логіки.

Таким чином, можна зробити висновок, що існує потреба у розробці окремого вебдодатку, який:

- фокусуватиметься виключно на сфері комп’ютерних комплектуючих;
- матиме вбудований конфігуратор для підбору комплектуючих;
- забезпечуватиме зручну навігацію, пошук та сортування за технічними параметрами;
- міститиме функціонал особистих кабінетів для покупців і продавців;
- дозволятиме легко масштабувати функціонал у майбутньому.

Такий підхід дозволить створити вузькоспеціалізований, ефективний інструмент для користувачів, які зацікавлені саме у сегменті комп’ютерних комплектуючих, що відрізняється специфічними вимогами до інтерфейсу, структури товарів та логіки взаємодії.

1.2 Порівняльний аналіз аналогів

На ринку електронної комерції представлено значну кількість відомих вебплатформ, які частково або повністю виконують функції маркетплейсу-конфігуратору для різних видів техніки, зокрема й комп’ютерних комплектуючих. Ці платформи забезпечують користувачам можливість здійснювати онлайн-покупки, а продавцям – розміщувати товари, отримувати замовлення та взаємодіяти зі своєю цільовою аудиторією. Серед найпопулярніших рішень в Україні можна виокремити такі застосунки, як Rozetka, Hotline, OLX та Prom.ua, кожен із яких має свої унікальні особливості, переваги та обмеження, що впливають на зручність користування з точки зору як покупця, так і продавця [3].

Rozetka є одним із найбільших та найвпізнаваніших інтернет-магазинів в Україні, який займає провідні позиції на ринку онлайн-торгівлі. Її основними перевагами є висока довіра з боку широкої аудиторії користувачів, налагоджена логістика, великий асортимент товарів, а також зручна система оплати й доставки, яка включає кілька варіантів для кінцевого споживача. Крім того, платформа має добре продуману систему фільтрації товарів, що дозволяє швидко

знайти необхідний продукт за технічними характеристиками, брендом, ціною та іншими параметрами. Водночас важливо зазначити, що Rozetka не є повноцінною відкритою платформою.

Інтерфейс Rozetka наведено на рисунку 1.1.

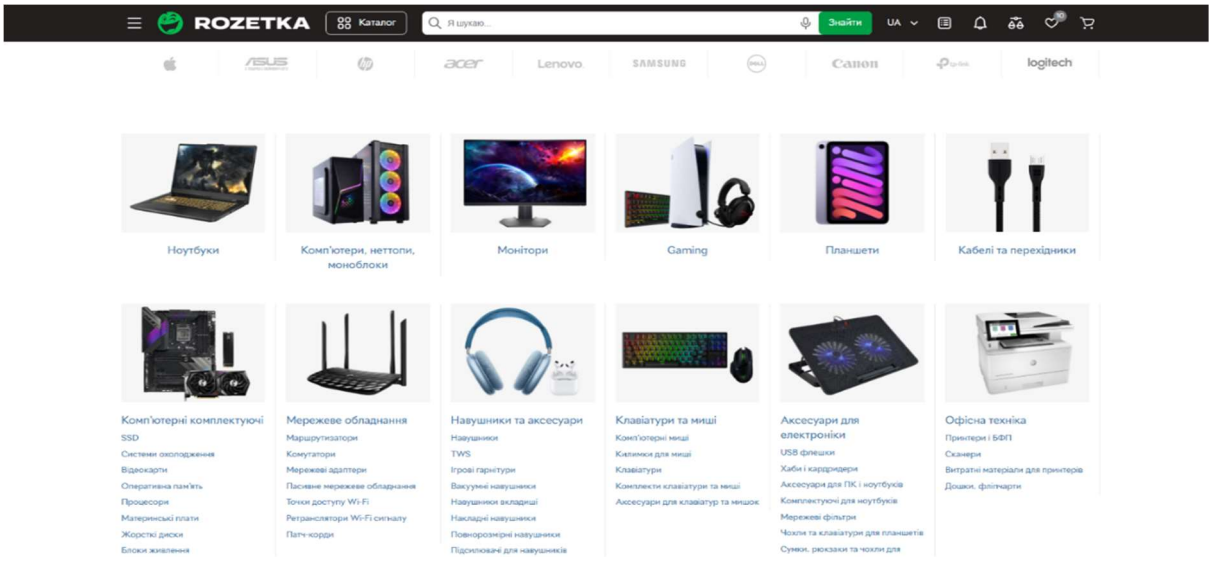

Рисунок 1.1 — Інтерфейс Rozetka

Hotline виступає в ролі агрегатора цін, порівнюючи пропозиції від різних продавців, але не реалізує повноцінний функціонал кошика та взаємодії продавець–покупець у межах однієї платформи. Інтерфейс Hotline наведено на рисунку 1.2.

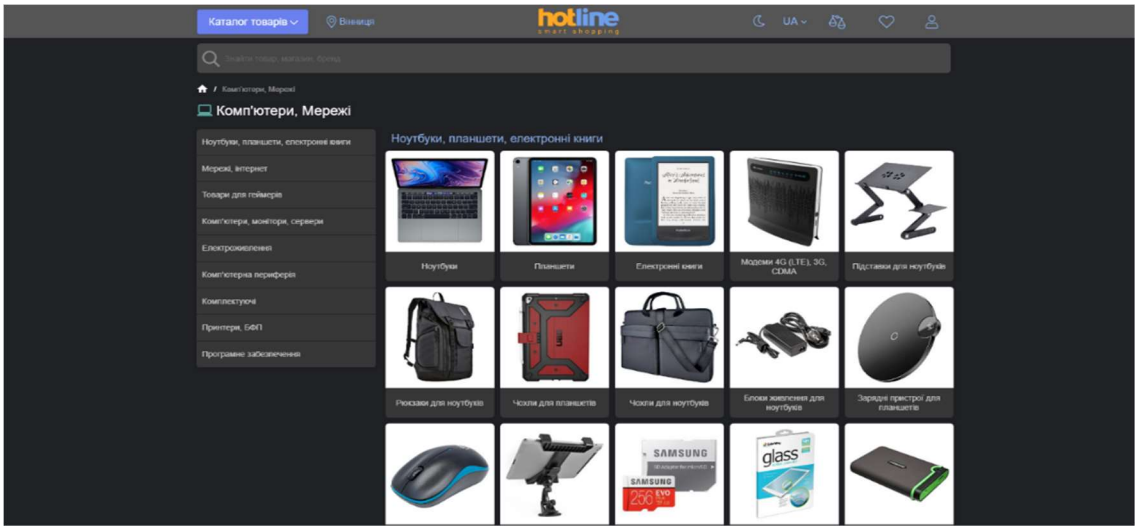

Рисунок 1.2 — Інтерфейс Hotline

OLX є універсальною дошкою оголошень, де можна продавати будь-які товари. Система проста та доступна, однак відсутність категоризації за технічними характеристиками, ризики шахрайства та відсутність підтримки транзакцій створюють обмеження для використання у сфері високовартісних комплектуючих.

Інтерфейс OLX наведено на рисунку 1.3.

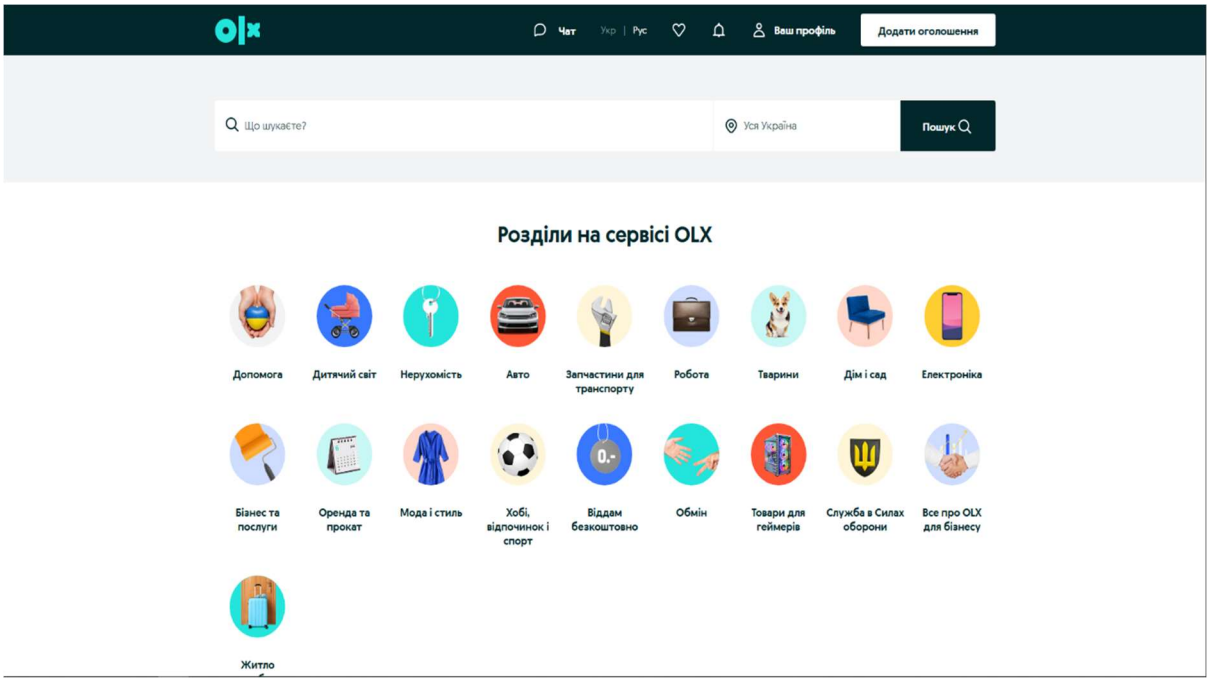

Рисунок 1.3 — Інтерфейс OLX

Prom.ua дозволяє малому та середньому бізнесу створювати повноцінні онлайн-магазини на базі єдиної платформи, що значно спрощує процес виходу на ринок електронної комерції. Завдяки готовій інфраструктурі та інструментам адміністрування, продавці можуть швидко розпочати торгівлю, не маючи глибоких технічних знань. Платформа підтримує оформлення замовлень, керування асортиментом, налаштування доставки й оплати, а також інтеграцію з популярними службами логістики та платіжними системами. Інтерфейс Prom.ua наведено на рисунку 1.4.

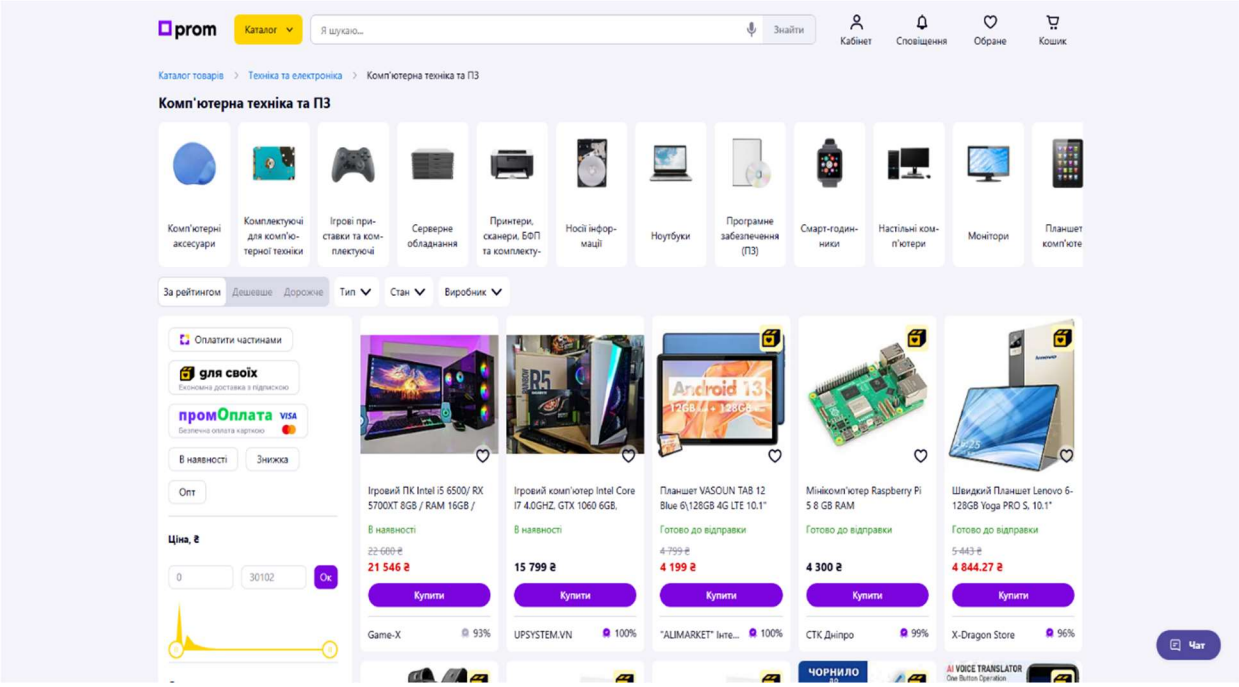

Рисунок 1.4 — Інтерфейс Prom.ua

Для порівняння характеристик цих застосунків із власною розробкою, було сформовано таблицю 1.1.

Таблиця 1.1 – Порівняльний аналіз аналогів

Функція	Rozetka	Hotline	OLX	Prom.ua	Власна розробка
Пошук і фільтрація	1	1	0	1	1
Реєстрація продавців	1	0	1	1	1
Особистий кабінет продавця	0	0	1	1	1
Інтегрований кошик	1	1	0	1	1
Статистика замовлень	1	0	0	1	1
Платформа без комісій	0	1	0	0	1
Орієнтація саме на комп’ютерні комплектуючі	0	0	0	0	1
Сумарний коефіцієнт	4	3	2	5	7

Як видно з аналізу, жодна з розглянутих платформ не об’єднує повний набір функцій, необхідних для зручної роботи з комп’ютерними комплектуючими.

Власна розробка дозволяє інтегрувати потрібний функціонал, уникнути зайвих обмежень та зробити систему спеціалізованою під технічну аудиторію.

1.3 Аналіз методів розв’язання задачі

Розробка вебдодатку типу маркетплейс-конфігуратор передбачає вибір ефективного підходу до побудови архітектури, який забезпечить зручність користування, масштабованість, безпечну обробку даних і простоту супроводу системи. Існує декілька поширених методів реалізації подібних систем, серед яких монолітна архітектура, мікросервісний підхід, serverless-рішення та використання no-code/low-code платформ [4].

Монолітна архітектура є класичним методом, коли вся бізнес-логіка, інтерфейс та робота з базою даних об'єднані в єдину програму. Такий підхід підходить для невеликих і середніх за масштабом систем, дозволяє швидко створити головний продукт, є простим у розгортанні та підтримці. Недоліком є складність масштабування й модифікацій у майбутньому при збільшенні функціоналу [5].

Альтернативою є мікросервісна архітектура, при якій система розділяється на незалежні модулі. Перевагами такого підходу є висока гнучкість, незалежність команд розробки, можливість масштабувати окремі частини системи. Проте розробка мікросервісної архітектури потребує значно більших зусиль, досвіду та чітко налагоджених процесів DevOps [6].

Serverless-архітектура передбачає використання обчислювальних ресурсів за потребою у вигляді окремих функцій, що виконуються на стороні хмарного провайдера. Такий підхід забезпечує хорошу масштабованість, спрощене обслуговування та оплати лише за використані ресурси. Однак у випадку складного застосунку з постійним зверненням до бази даних, великою кількістю взаємозалежних процесів така модель може бути неефективною [7].

Ще одним варіантом є використання no-code або low-code платформ, які дозволяють створювати вебзастосунки без глибоких знань у програмуванні. Цей підхід є швидким для створення простих прототипів або адміністративних

панелей, однак його можливості обмежені в контексті побудови повноцінного маркетплейсу з розширеним функціоналом і кастомною логікою.

З урахуванням вищезгаданого, для реалізації вебдодатку маркетплейс комп'ютерних комплектуючих було обрано класичну клієнт-серверну архітектуру із чітким розмежуванням frontend і backend частин. Такий підхід дозволяє забезпечити достатню гнучкість, розширюваність і простоту у підтримці системи. Для клієнтської частини обрано фреймворк React, що забезпечує побудову інтерактивного інтерфейсу користувача. Серверна частина реалізована з використанням Node.js у поєднанні з Express, що дозволяє ефективно обробляти запити та працювати з REST API. У якості системи управління базами даних застосовується PostgreSQL, яка надає широкі можливості для роботи зі структурованими даними та гарантує надійність і масштабованість зберігання [8].

Обрана модель реалізації є оптимальною для проєкту середнього масштабу, забезпечує можливість подальшого масштабування, інтеграції з платіжними системами, службами доставки та додавання нового функціоналу без порушення цілісності системи.

1.4 Постановка задач дослідження

У результаті проведеного аналізу предметної області, оцінки існуючих аналогів та методів реалізації, було сформульовано основні задачі дослідження, які необхідно розв'язати в межах кваліфікаційної роботи:

- провести аналіз існуючих рішень у сфері ПК-конфігураторів;
- розробити архітектуру вебзастосунку;
- модифікувати метод перевірки сумісності компонентів;
- розробити інтуїтивно зрозумілий інтерфейс користувача;
- розробити механізм збереження та експорту шаблонів конфігурацій;
- провести тестування системи.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У першому розділі бакалаврської кваліфікаційної роботи було проведено детальний аналіз стану питання щодо розробки вебзастосунків типу маркетплейс-конфігуратор, а також вивчено сучасні технології, які застосовуються у створенні подібних систем. Було розглянуто найбільш відомі існуючі рішення, виділено їхні переваги та недоліки, здійснено порівняльний аналіз функціоналу, що дозволило обґрунтувати доцільність створення власного спеціалізованого додатку, орієнтованого саме на сферу комп'ютерних комплектуючих. Акцент зроблено на потребі у зручному, адаптивному інтерфейсі та гнучкій системі управління контентом.

Окремо було проаналізовано методи реалізації маркетплейсів – від класичних монолітних архітектур до сучасних мікросервісних і serverless-рішень. Кожен підхід було оцінено з позиції складності впровадження, ефективності масштабування, вартості супроводу та відповідності поставленим завданням. У результаті технічного порівняння та оцінки можливостей інтеграції обрано оптимальний варіант – клієнт-серверну модель з використанням React, Node.js та PostgreSQL, що дозволяє розробити додаток із належним рівнем гнучкості, адаптивності, продуктивності та подальшого розширення функціоналу без суттєвих змін в архітектурі.

Також було сформульовано основні задачі, які необхідно вирішити в межах бакалаврської кваліфікаційної роботи, зокрема: розробка архітектури, реалізація інтерфейсу користувача, створення серверної частини та бази даних, впровадження ключових функцій застосунку, включно з пошуком, фільтрацією, кошиком і обробкою замовлень, а також тестування системи на предмет працездатності, зручності та відповідності функціональним вимогам. Усе це створює ґрунтовну основу для реалізації повноцінного вебдодатку маркетплейсу-конфігуратора в подальших розділах, де буде детально описано процес розробки, реалізацію алгоритмів та проведення практичної перевірки ефективності розробленої системи.

2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ РОБОТИ СИСТЕМИ

2.1 Аналіз інформаційного забезпечення

Інформаційне забезпечення є однією з ключових складових при розробці вебдодатку маркетплейсу-конфігуратора, оскільки саме воно визначає структуру, обсяг і формат даних, що циркулюють у системі. Від ефективності організації інформаційного середовища безпосередньо залежить стабільність роботи додатку, зручність користування, швидкість взаємодії між компонентами системи та можливість масштабування функціоналу в майбутньому. Інформаційне забезпечення виступає базовим елементом, на основі якого функціонує уся логіка застосунку, починаючи з реєстрації користувача й завершуючи формуванням аналітики для адміністратора.

До інформаційного забезпечення даного застосунку входить набір структурованих даних, які необхідні для повноцінного функціонування таких підсистем, як реєстрація та автентифікація, керування товарами, оформлення замовлень, відображення каталогу, фільтрація, сортування, обробка запитів та формування звітів. Всі ці процеси потребують чітко визначених структур даних, що забезпечують уніфіковану взаємодію між клієнтською частиною, сервером та базою даних.

Основними категоріями інформації в системі є:

- дані користувачів – облікові записи покупців і продавців, які містять логін, пароль, електронну пошту, роль у системі, контактну інформацію, історію замовлень. Вони дозволяють реалізувати персоніфіковану взаємодію, розмежування прав доступу, а також забезпечити безпечну авторизацію;
- дані товарів – інформація про комп’ютерні комплектуючі та периферійні пристрої: назва, опис, ціна, кількість на складі, наявність, зображення, технічні характеристики дата додавання та ID продавця;
- категорії та підкатегорії товарів – структурована інформація, яка дозволяє групувати товари за типами: процесори, відеокарти, материнські плати, оперативна пам’ять, SSD, блоки живлення, корпуси, кулери, монітори

тощо. Ієрархічна побудова категорій дозволяє гнучко реалізувати навігацію та фільтрацію;

- замовлення – сукупність даних, що відображає оформлену покупку: перелік обраних товарів, загальна сума, контактні дані покупця, статус (нове, в обробці, відправлено, виконано), дата створення, спосіб доставки, оплати, коментарі до замовлення;
- коментарі, рейтинги та відгуки – інформація, що формується користувачами після взаємодії з товарами або продавцями. Вона дозволяє формувати рейтинг продукції, підвищувати довіру до платформи та впливати на репутацію продавців;
- адміністративні дані – включають лог подій, налаштування системи, результати модерації товарів, перегляд статистики активності, кількість замовлень по категоріях, популярність товарів, тощо. Ці дані використовуються для контролю роботи платформи та ухвалення управлінських рішень.

Усі ці дані мають бути представлені у формалізованому вигляді та збережені в базі даних, яка забезпечує цілісність, уніфікованість, взаємозв’язки між таблицями та швидкий доступ до інформації. У якості системи управління базами даних для реалізації додатку обрано PostgreSQL, яка дозволяє створювати повноцінні реляційні структури, підтримує складні SQL-запити, індексацію, транзакції та збереження зв’язків між сутностями. Такий вибір забезпечує як горизонтальне, так і вертикальне масштабування, що є особливо важливим для систем, які передбачають обробку великої кількості товарів і одночасних користувачів [9].

Інформація між клієнтською частиною та сервером передається у форматі JSON через REST API, що дозволяє ефективно структурувати запити та відповіді, а також легко інтегрувати додаткові модулі або мобільні версії застосунку. Кожен запит має чітко визначену структуру з передбаченими полями, що мінімізує ризик помилок під час взаємодії компонентів системи [10].

Також важливим аспектом є форматування і перевірка даних, які вводить користувач, як під час реєстрації, так і при додаванні нових товарів, редагуванні

профілю чи оформленні замовлення. Уся вхідна інформація проходить перевірку на коректність за допомогою бібліотек валідації що забезпечує послідовність і правильність збережених даних, а також захищає систему від SQL-ін’єкцій, XSS-атак та інших типових вразливостей [11].

Таким чином, інформаційне забезпечення вебдодатку маркетплейсу-конфігуратора охоплює набір пов’язаних між собою сутностей, які дозволяють організувати ефективну взаємодію між користувачами, продавцями та адміністрацією платформи. Від його якості залежить цілісність архітектури, швидкість роботи системи та можливість її масштабування у майбутньому. Продумана структура даних є основою надійності, функціональності та зручності застосунку, що є критично важливим для комерційних платформ.

2.2 Розробка моделі застосунку

Вебдодаток маркетплейсу-конфігуратора комп’ютерних комплектуючих складається з низки взаємопов’язаних компонентів, які разом формують цілісну екосистему для взаємодії покупців, продавців та адміністрації платформи. Ці компоненти охоплюють клієнтську та серверну частини, базу даних, а також інтерфейс взаємодії між ними – REST API. Усі елементи системи працюють узгоджено, забезпечуючи злагоджену обробку запитів, відповідей, збереження даних і відображення інформації у браузері користувача [12].

Архітектура додатку побудована на основі класичної клієнт-серверної моделі, що передбачає чіткий розподіл відповідальностей між фронтендом і бекендом. Такий підхід дозволяє ефективно масштабувати систему, окремо оновлювати або оптимізувати її частини, а також забезпечує можливість майбутнього підключення мобільного клієнта або сторонніх сервісів.

Користувацький інтерфейс реалізовано за допомогою популярної бібліотеки React, яка дозволяє будувати гнучкий, адаптивний та високопродуктивний UI. React забезпечує компонентний підхід, завдяки якому всі частини інтерфейсу винесені в окремі незалежні компоненти. Це підвищує зручність розробки, спрощує повторне використання коду, покращує

читабельність і підтримку проєкту.

Для реалізації навігації між основними розділами додатку використовується бібліотека React Router, яка забезпечує маршрутизацію на клієнтській стороні. Це дозволяє реалізувати глибокі посилання, вкладені маршрути, а також приватні маршрути, доступні лише авторизованим користувачам. Останні є критично важливими для захисту сторінок на кшталт «Мої замовлення» або «Кабінет продавця» [13].

Управління глобальним станом застосунку централізовано реалізовано через Redux у поєднанні з Redux Toolkit – офіційною обгорткою, яка спрощує написання редукторів, дій і обробку асинхронних запитів. Це дозволяє тримати всі важливі дані в одному сховищі, доступному для будь-якого компонента. Арітектура Redux-стану розбита на функціональні області: auth, products, orders, cart, userProfile, що відповідає принципам чистої архітектури [14].

Для комунікації між клієнтом і сервером використовується бібліотека Axios, яка забезпечує надсилання HTTP-запитів до API з підтримкою токенів авторизації, обробкою помилок, таймаутів та перехоплення відповідей. Обмін даними відбувається у форматі JSON, що забезпечує універсальність, зручність у логуванні й сумісність з більшістю сучасних інтерфейсів.

З боку бекенду застосунок реалізований на основі Node.js, з використанням фреймворку Express. Цей стек дозволяє створити легкий, але гнучкий сервер, який підтримує обробку REST-запитів, побудову middleware-ланцюжків, обробку помилок, логування, захист CORS та багато іншого. Усі маршрути додатку згруповані в окремі файли за логікою, кожен з яких має відповідний контролер, що містить бізнес-логіку.

Для автентифікації та авторизації користувачів використовується JWT (JSON Web Token) – токени, які створюються під час входу та передаються клієнтом у заголовках запитів. На сервері ці токени перевіряються на валідність і термін дії, а також дозволяють отримати ID користувача та його роль, що дає змогу реалізовувати доступ лише до дозволених дій [15].

Зберігання даних реалізовано у реляційній базі даних PostgreSQL, яка є

однією з найпотужніших та найнадійніших систем з відкритим кодом. PostgreSQL забезпечує високий рівень безпеки, масштабованість та розширюваність, що робить її ідеальним вибором для складних веб-застосунків та комерційних проєктів. У базі даних створено детально структуровану схему, яка включає ключові таблиці: users, products, categories, orders, order_items, reviews, а також допоміжні – наприклад, для зберігання адрес доставки, платіжних методів або історії змін.

Кожна таблиця має чітко визначені первинні ключі, які забезпечують унікальність записів, а також зв’язки між таблицями реалізовано через зовнішні ключі. Це дозволяє ефективно організовувати дані та забезпечити цілісність інформації. Для пришвидшення пошуку та оптимізації виконання запитів, найчастіше використовувані поля індексуються. Це особливо важливо при роботі з великим обсягом даних, зокрема в таблицях orders та products, де швидкість доступу критично важлива для користувацького досвіду.

PostgreSQL також підтримує транзакції, що гарантує консистентність даних у разі виникнення помилок під час виконання операцій – наприклад, при оформленні замовлення або зміні стану кошика. Це дає змогу уникнути частково виконаних або пошкоджених записів, що має ключове значення для систем електронної комерції.

Схема бази даних розроблялась із дотриманням принципів нормалізації – для уникнення надмірного дублювання даних та підвищення логічної узгодженості структури. Водночас було враховано можливість розширення структури у майбутньому.

У результаті побудована модель застосунку поєднує в собі модульність, масштабованість, безпеку та гнучкість, що дозволяє реалізувати повноцінний маркетплейс-конфігуратор із можливістю подальшого розширення. Така архітектура є оптимальною для проєктів середнього масштабу з перспективою розвитку у SaaS-платформу або комерційний продукт. Схема архітектури застосунку зображена на рисунку 2.1.

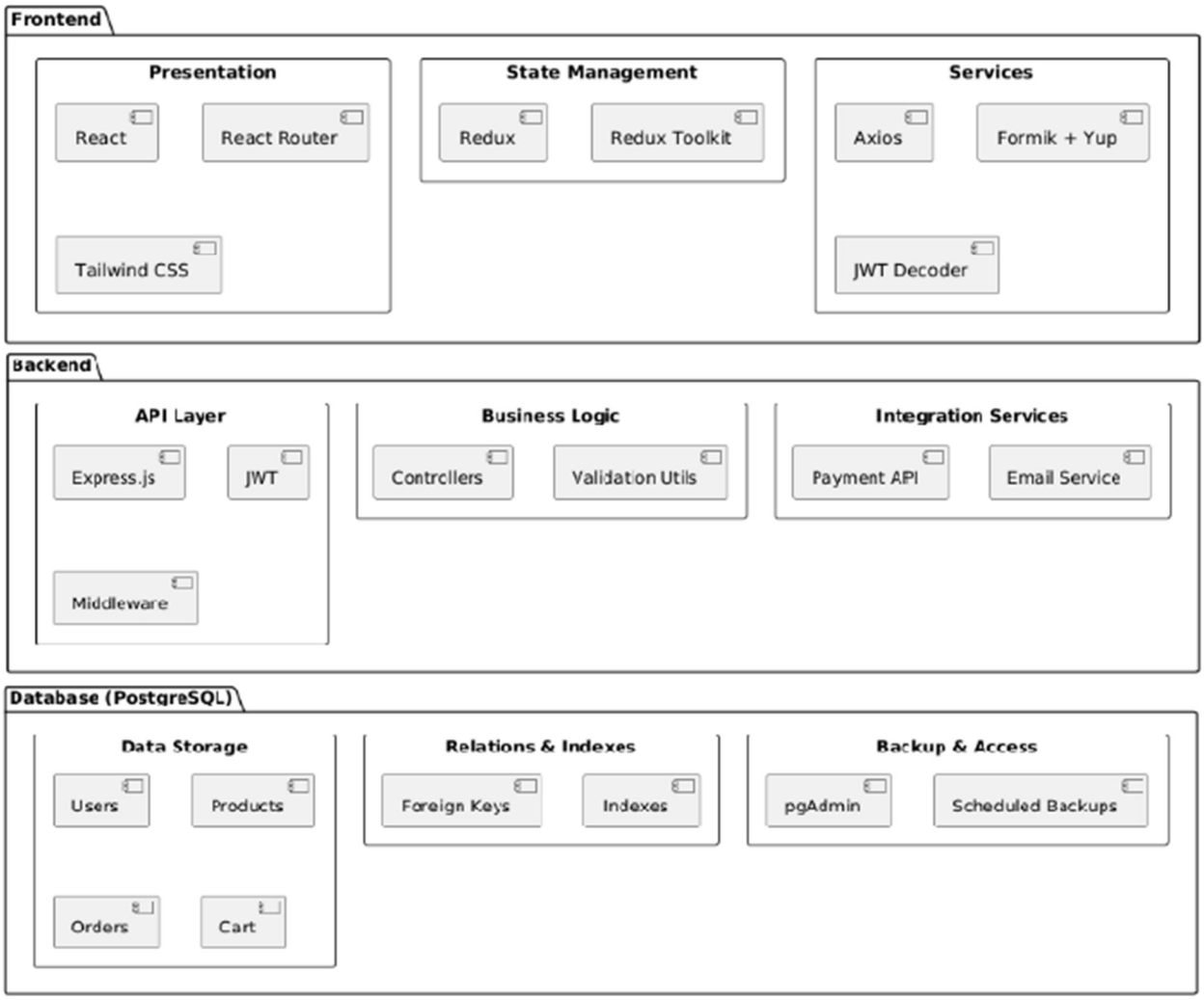

Рисунок 2.1 – Схема архітектури застосунку

2.3 Розробка структури бази даних

База даних є критично важливою складовою додатку, оскільки забезпечує централізоване зберігання, обробку та взаємозв’язок усіх основних сутностей системи. Саме через базу даних реалізується повноцінна взаємодія між користувачами, товарами, кошиком, замовленнями, категоріями, відгуками та адміністративними модулями. Для реалізації цієї частини застосунку обрана реляційна система управління базами даних PostgreSQL, яка зарекомендувала себе як потужна, гнучка, надійна та масштабована СУБД з підтримкою складних транзакцій і глибокої індексації.

У структурі бази даних розроблено такі основні таблиці:

- Users – зберігає дані зареєстрованих користувачів, включаючи логін, email, захешований пароль, роль користувача, дату створення облікового запису, статус. Ця таблиця є базовою для автентифікації та управління доступом;
- Products – основна таблиця товарів, що містить назву товару, короткий опис, повну характеристику, ціну, кількість у наявності, посилання на фото, технічні параметри, ID категорії, а також ID продавця, що дозволяє відслідковувати власників товарів;
- Categories – таблиця ієрархічної структури категорій товарів. Кожен запис містить назву категорії, унікальний ID, а також посилання на батьківську категорію;
- Orders – фіксує кожне замовлення, включаючи ID покупця, дату оформлення, суму замовлення, статус виконання, спосіб оплати, метод доставки тощо;
- OrderItems – проміжна таблиця, яка зв’язує Orders і Products, містить ID товару, кількість одиниць, ціну товару на момент оформлення замовлення. Це дозволяє зберігати історичну інформацію незалежно від подальших змін у товарах;
- Reviews – таблиця відгуків, які залишають користувачі про конкретні товари. Містить текст коментаря, числовий рейтинг, дату публікації, ID автора відгуку та ID продукту. Завдяки їй формується оцінка якості товарів і репутації продавців;
- Cart – таблиця, яка зберігає поточний стан кошика для кожного користувача. Містить ID користувача, ID товару, кількість, а також мітку часу останнього оновлення. Ця структура дозволяє зберігати вміст кошика між сесіями;
- Images – окрема таблиця для збереження посилань на зображення товарів. Це дозволяє гнучко підтримувати декілька зображень на один товар і не перевантажувати основну таблицю Products.

Для підвищення продуктивності та забезпечення швидкого доступу до інформації реалізовано індексацію ключових полів, таких як назва товару, email

користувача, назва категорії, що часто використовуються під час фільтрації, сортування або пошуку. Це дозволяє досягти низької затримки навіть при роботі з великим обсягом даних – наприклад, при пошуку серед тисяч товарів за назвою або брендом.

Зв’язки між таблицями реалізовані через зовнішні ключі, що дозволяє забезпечити цілісність даних, а також спростити побудову JOIN-запитів у запитах на сервері. Наприклад, при виборі замовлень для конкретного користувача можна легко отримати відповідні товари з таблиці OrderItems, а згодом і їх характеристики з Products.

Структура бази даних спроєктована з урахуванням можливості її розширення. У подальшому до системи легко інтегруються додаткові таблиці, наприклад:

- Payments – для обробки платіжної інформації;
- ShippingInfo –для деталізації логістичних даних;
- Messages – для організації внутрішньої переписки між продавцем і покупцем;
- SupportTickets – для реалізації підсистеми технічної підтримки;
- Wishlists – для збереження обраних товарів користувачем.

Крім того, PostgreSQL дозволяє реалізовувати перевірки цілісності, каскадне видалення, унікальні обмеження, тригери для автоматичної обробки подій та інші корисні механізми, які допомагають підтримувати правильну бізнес-логіку навіть на рівні бази даних.

Загалом, така структура бази даних є гнучкою, масштабованою та безпечною, що дозволяє ефективно реалізовувати всі необхідні функції маркетплейсу-конфігуратора та підтримує розвиток платформи у майбутньому.

2.4 Розробка алгоритмів роботи застосунку

Алгоритм реєстрації користувача описує послідовність дій, необхідну для створення нового облікового запису в системі додатку. Реєстрація є ключовим кроком, який відкриває доступ до персоналізованого функціоналу платформи: додавання товарів, оформлення замовлень, перегляд історії покупок, залишення відгуків тощо. Вона передбачає введення основних даних – електронної пошти, пароля та вибору ролі, які потім проходять перевірку на коректність і унікальність. Успішна реєстрація супроводжується створенням токена авторизації, що надалі використовується для захисту дій користувача в системі. Алгоритм реалізовано з урахуванням безпеки та зручності, що дозволяє уникнути дублювання облікових записів і гарантує збереження конфіденційних даних.

Процес розпочинається з того, що користувач заповнює форму, в якій вказує електронну пошту, пароль та вибирає роль – покупець або продавець. Після надсилання форми дані надходять до серверної частини через HTTP POST-запит.

Сервер спочатку виконує перевірку унікальності email у базі даних, щоб запобігти дублюванню облікових записів. У разі виявлення наявного email реєстрація припиняється, і користувач отримує відповідне повідомлення про помилку.

Якщо email є унікальним, система переходить до етапу хешування пароля за допомогою криптографічної бібліотеки. Цей крок є надзвичайно важливим з точки зору безпеки – у базі даних зберігається не сам пароль, а його хеш, що унеможливлює розкриття навіть у разі витоку.

Далі створюється новий запис у таблиці Users, який містить email, хешований пароль, роль, дату створення та інші допоміжні атрибути

Після успішного створення облікового запису система генерує JWT-токен, який містить зашифровану інформацію про користувача — зокрема ID та роль. Цей токен надсилається у відповідь клієнту й використовується для авторизації при подальшій роботі з платформою.

Процес є повністю автоматизованим і займає менше секунди, забезпечуючи зручний, безпечний і швидкий вхід нового користувача в екосистему додатку.

Блок-схема цього алгоритму наведена на рисунку 2.2.

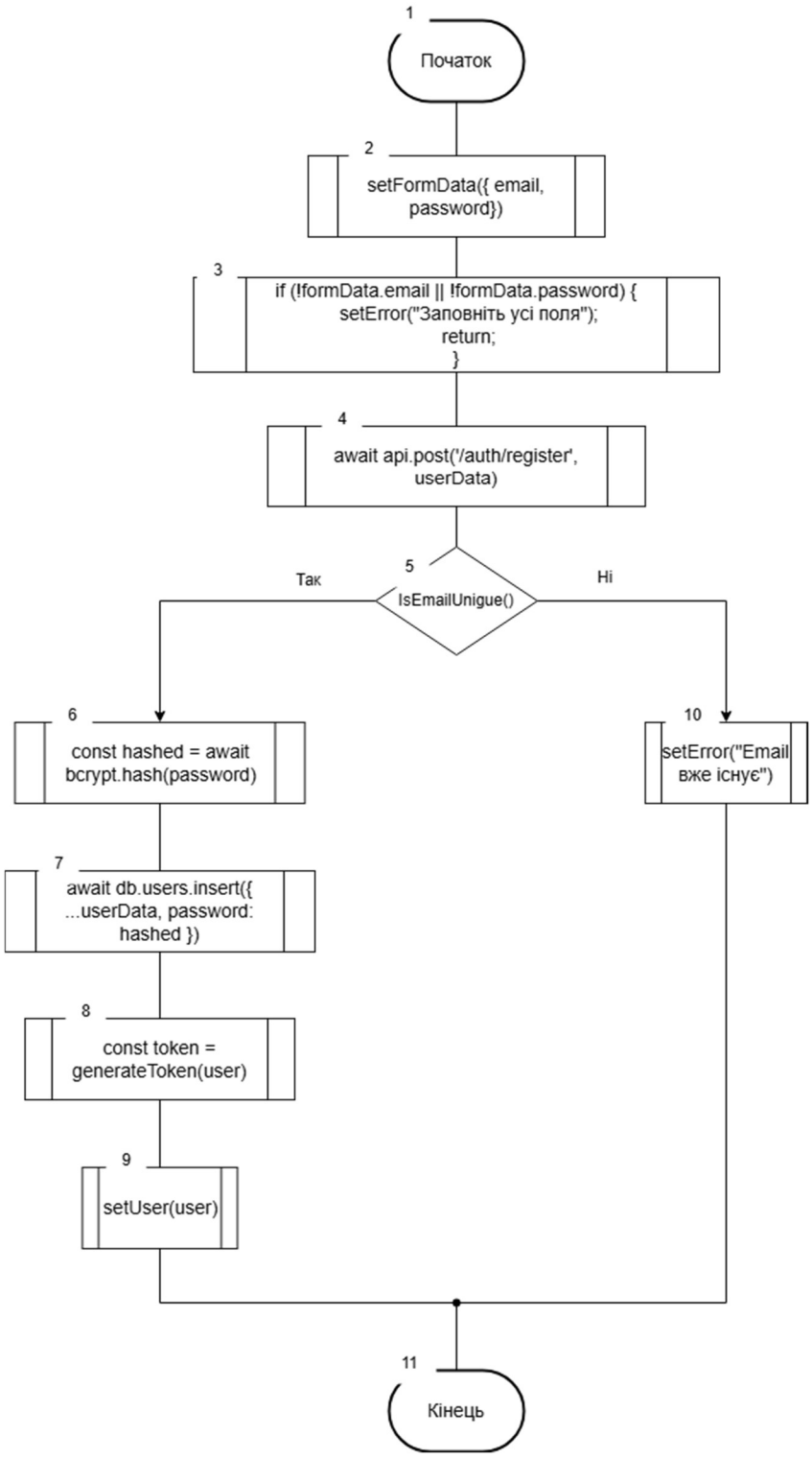

Рисунок 2.2 — Блок-схема алгоритму реєстрації користувача

Алгоритм додавання товару описує ключову послідовність дій, яку виконує продавець при публікації нової позиції в маркетплейсі. Цей процес є важливою складовою функціоналу особистого кабінету продавця, оскільки саме через нього формується асортимент товарів, доступних для покупців.

Процедура починається з того, що авторизований продавець переходить до інтерфейсу додавання товару в особистому кабінеті. У відповідній формі він заповнює обов’язкові поля: назву товару, опис, ціну, кількість, категорію, технічні характеристики, гарантійні умови, а також додає зображення продукту. Інтерфейс реалізовано з використанням зручної форми з перевіркою даних у реальному часі.

Після натискання кнопки «Додати» або «Зберегти», на клієнтській стороні формується HTTP POST-запит, який надсилається на захищений маршрут API. Перед передачею запиту перевіряється, чи користувач автентифікований і має роль продавця. Ця перевірка виконується через middleware на сервері, що обробляє JWT-токен і декодує з нього роль користувача.

Далі відбувається серверна перевірка даних, щоб переконатись у коректності полів. У разі успішного проходження валідації, створюється новий запис у таблиці Products, до якого обов’язково прив’язується ID продавця, отриманий із токена. Це дозволяє у подальшому реалізувати розділення товарів по продавцях та створення особистих вітрин.

Новий товар створюється із статусом «на модерації», що означає, що він ще не відображається на загальнодоступній сторінці маркетплейсу. Такий підхід дозволяє адміністраторам платформи вручну перевіряти відповідність опису товару правилам, наявність недопустимого контенту або потенційні порушення.

Таким чином, алгоритм забезпечує безпечну, контрольовану та структуровану публікацію нових товарів, дозволяючи уникнути шахрайства, спаму або технічних помилок. Блок-схема даного алгоритму наведена на рисунку 2.3.

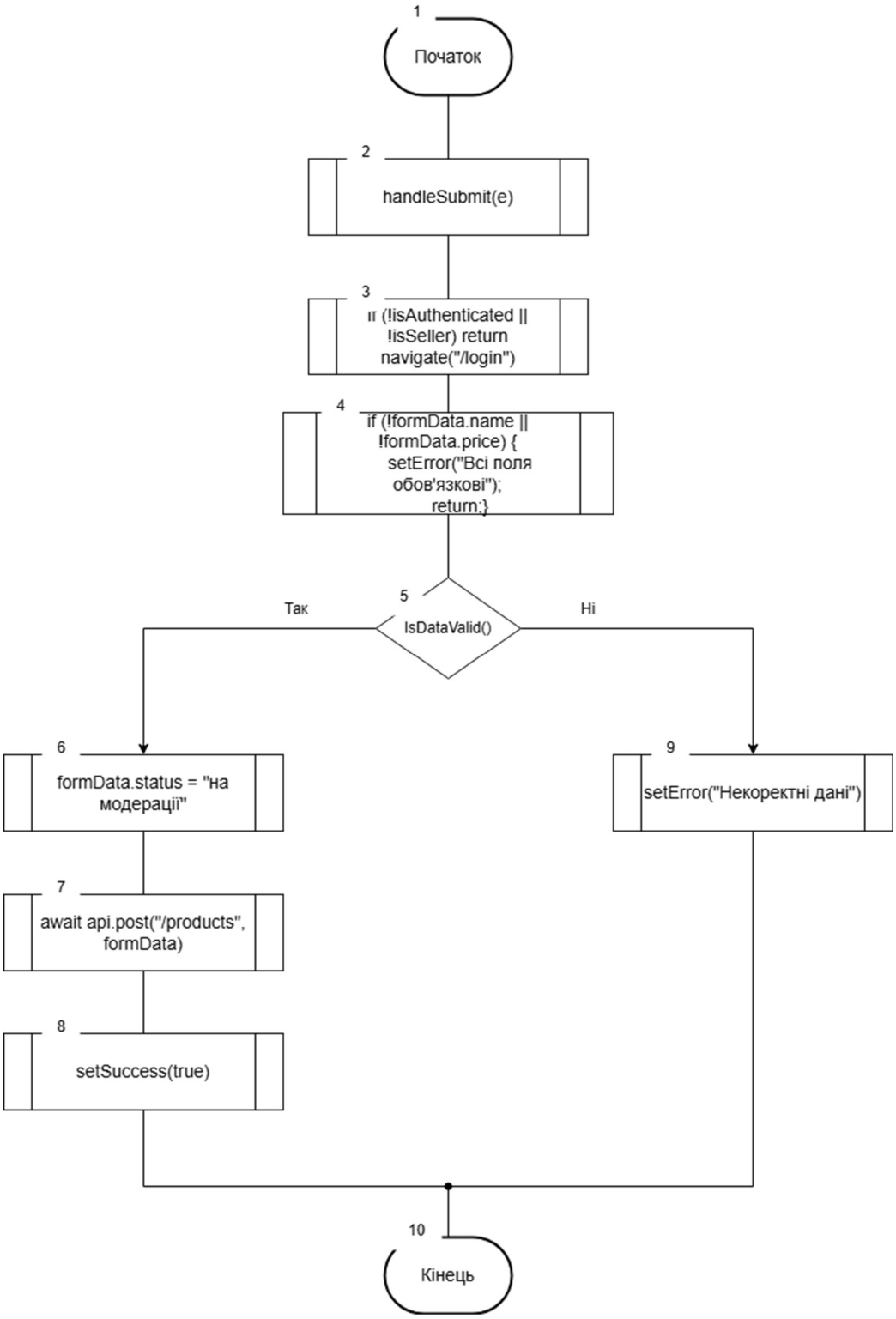

Рисунок 2.3 — Блок-схема алгоритму додавання товару продавцем

Алгоритм додавання товару до кошика описує одну з найтиповіших дій, яку здійснює покупець під час взаємодії з маркетплейсом. Цей процес є частиною користувацького сценарію формування замовлення та має бути реалізований інтуїтивно зрозуміло, швидко та безпомилково.

Процедура починається з того, що користувач переглядає каталог товарів і обирає позицію, яку бажає придбати. Натисканням кнопки «Додати до кошика» запускається обробник події, який формує відповідну Redux-дію або HTTP-запит при реалізації кошика на сервері. Перед цим система перевіряє, чи користувач авторизований, щоб вирішити, чи зберігати вміст локально для гостьових сесій, чи в базі даних для зареєстрованих користувачів.

У випадку авторизованого користувача система перевіряє, чи вже існує активний кошик, пов’язаний із його профілем у базі даних. Якщо такий кошик знайдено, новий товар або додається до нього якщо відсутній, або збільшується його кількість якщо вже є у списку. У разі, якщо це перше додавання товару, система створює новий кошик з відповідним записом у таблиці Cart.

Після оновлення або створення кошика система переобчислює загальну вартість і кількість позицій, що відображається у спеціальному індикаторі на інтерфейсі. Дії виконуються асинхронно, без оновлення сторінки, що створює приємний досвід взаємодії з інтерфейсом. У випадку локального зберігання неавторизовані сесії стан кошика утримується у Redux або LocalStorage, з подальшою синхронізацією після авторизації. Такий підхід дозволяє зберегти вибір користувача навіть при зміні сесії або перезапуску сторінки. Крім того, синхронізація з базою даних після входу гарантує цілісність даних і забезпечує коректне продовження взаємодії з кошиком без втрати інформації.

Такий алгоритм дозволяє забезпечити гнучкість, масштабованість і надійність функціоналу кошика, незалежно від статусу користувача. Блок-схема цього алгоритму наведена на рисунку 2.4.

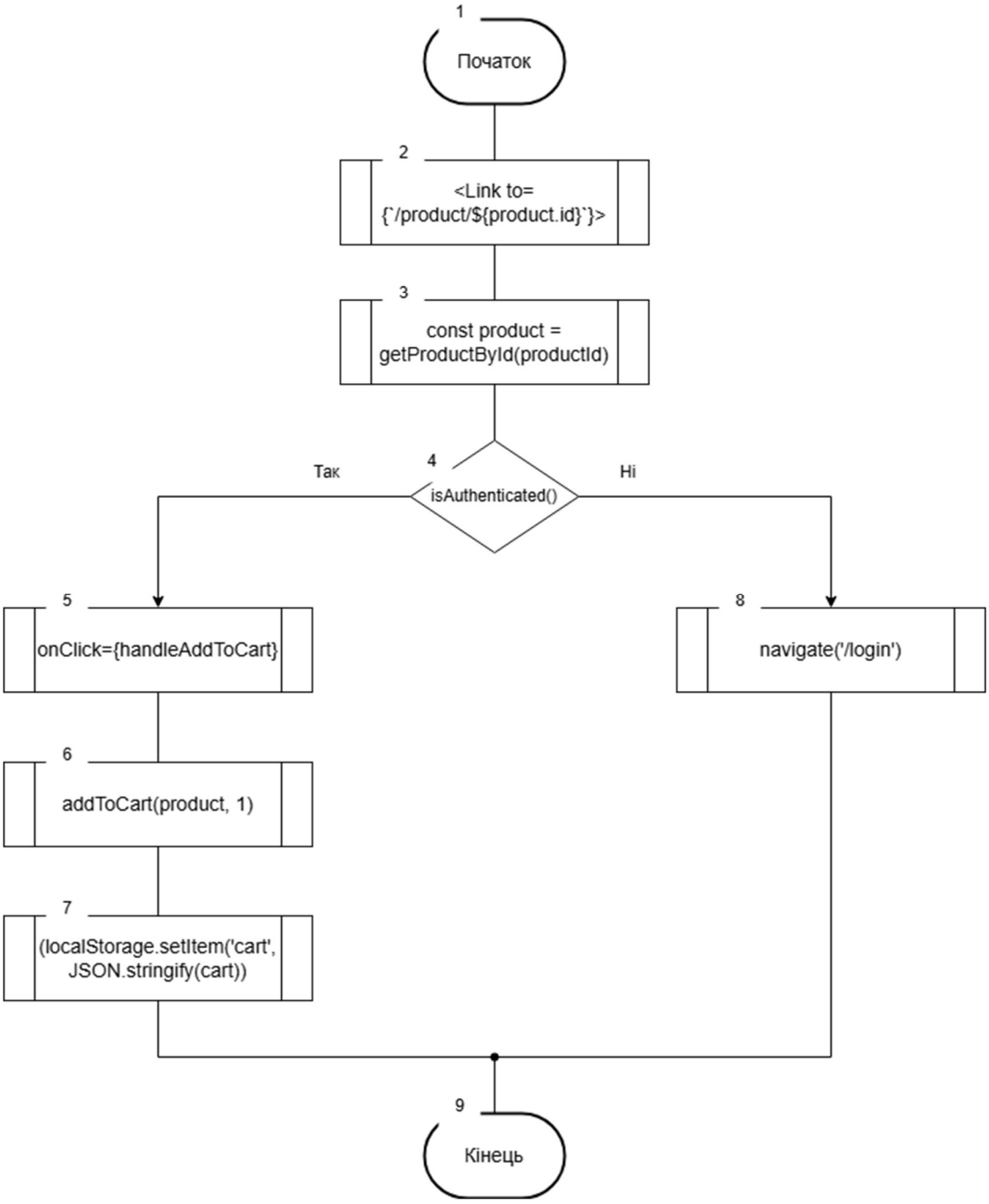

Рисунок 2.4 - Блок-схема алгоритму додавання товару до кошику

Алгоритм оформлення замовлення описує завершальний етап користувацького сценарію купівлі, який настає після формування кошика з вибраними товарами. Цей процес є критично важливим для реалізації основної функції маркетплейсу-конфігуратора – здійснення онлайн-продажів, і тому потребує особливої уваги до перевірок, точності та надійності.

Після переходу до сторінки оформлення замовлення користувач спочатку проходить перевірку авторизації. Якщо користувач не увійшов у систему, він буде

перенаправлений на сторінку входу, оскільки оформлення замовлень доступне лише зареєстрованим користувачам. Це дозволяє забезпечити збереження історії замовлень, підв'язування до профілю покупця та формування аналітики.

На етапі оформлення користувач заповнює форму доставки, де вказує ПІБ, номер телефону, адресу включаючи поштовий індекс, населений пункт, відділення служби доставки або адресу кур’єрської доставки, а також вибирає спосіб оплати. Для зручності форма реалізована з використанням компонентів з перевіркою даних у реальному часі.

Після заповнення форми та натискання кнопки «Підтвердити замовлення», клієнтська частина формує HTTP POST-запит на сервер, який включає дані доставки, контактну інформацію та перелік товарів у кошику. Сервер перевіряє наявність товарів, коректність переданих даних, а також знову виконує авторизацію користувача через JWT токен. У разі успішної перевірки створюється новий запис у таблиці Orders**,** який включає ID користувача, дату оформлення, загальну суму, статус замовлення – наприклад, «нове». Паралельно в таблиці OrderItems фіксується перелік замовлених товарів із зазначенням їхньої кількості та ціни на момент оформлення, що дає змогу зберігати історичну точність навіть у разі подальших змін у базі товарів. Такий підхід забезпечує цілісність замовлення, дозволяє уникнути плутанини в даних і є критично важливим для формування звітів про продажі, статистики та аналітики. Крім того, інформація про замовлення може бути використана для реалізації додаткового функціоналу – наприклад, сповіщення користувача про зміну статусу доставки, формування рахунків, генерації підтвердження замовлення на email або реалізації повторних покупок. У майбутньому така структура забезпечує гнучкість для інтеграції з платіжними системами, CRM-модулями або службами доставки.

Такий підхід дозволяє реалізувати транзакційно стійкий і зручний для користувача процес оформлення замовлення, що відповідає практикам сучасних e-commerce платформ. Блок-схема алгоритму оформлення замовлення наведена на рисунку 2.5.

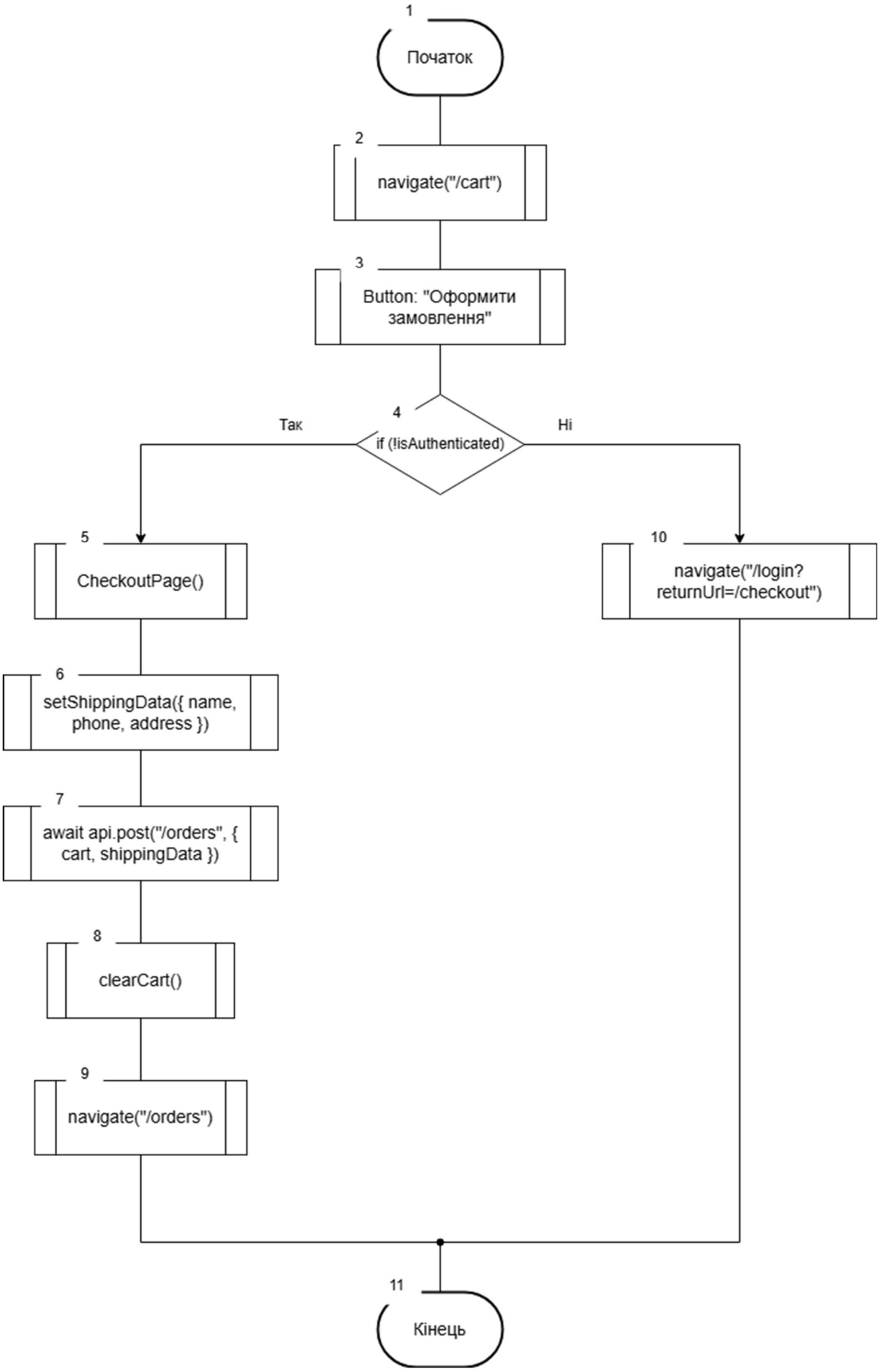

Рисунок 2.5 — Блок-схема алгоритму оформлення замовлення

Розроблені алгоритми забезпечують основний функціонал вебдодатку маркетплейсу-конфігуратора комп’ютерних комплектуючих та охоплюють найважливіші сценарії взаємодії користувача з платформою. Вони реалізують повний цикл роботи з системою: від реєстрації нового користувача до оформлення замовлення з обраних товарів. Алгоритм реєстрації гарантує створення безпечного облікового запису з перевіреними даними. Алгоритм додавання товару передбачає модерацію, що підвищує контроль якості вмісту. Механізми роботи кошика дозволяють динамічно керувати вмістом без необхідності авторизації, із подальшою синхронізацією. Оформлення замовлення реалізовано як транзакційний процес з перевірками, що унеможливлює втрату або спотворення даних.

Усі алгоритми побудовані за принципами модульності, розширюваності та високої надійності, що дозволяє легко адаптувати їх до нових вимог або інтегрувати з додатковими підсистемами – наприклад, оплатою, доставкою, службою підтримки. Завдяки чітко визначеним крокам кожного процесу, розроблені алгоритми сприяють зручній, безпечній та логічній роботі маркетплейсу-конфігуратора в режимі реального часу, з мінімальними затримками та високим рівнем користувацького задоволення.

2.5 Висновки

У другому розділі було розроблено структуру вебдодатку маркетплейсу-конфігуратора, побудовано модель його роботи та детально описано основні алгоритми взаємодії користувачів із системою. Було проаналізовано інформаційне забезпечення, спроєктовано архітектуру клієнтської та серверної частин, розроблено логічну структуру бази даних, яка забезпечує надійне та цілісне зберігання ключових сутностей. Алгоритми охоплюють найважливіші сценарії використання платформи – від реєстрації користувачів і додавання товарів до оформлення замовлень і керування кошиком.

Представлені рішення базуються на сучасних підходах до веброзробки, таких як REST-архітектура, токенна авторизація, модульність бізнес-логіки та

асинхронна взаємодія між компонентами. Запропонована модель дозволяє не лише реалізувати всі ключові функції додатку, але й забезпечує основу для масштабування, інтеграції з іншими сервісами та підтримки високого рівня надійності в умовах реального використання.

Таким чином, другий розділ закладає фундаментальну технічну базу для подальшої реалізації програмного коду, перевірки працездатності системи та оцінки її відповідності функціональним вимогам, що буде розглянуто у наступних розділах пояснювальної записки.

3 РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ

3.1 Варіантний аналіз та вибір мови програмування

Для розробки маркетплейсу-конфігуратора комп’ютерних комплектуючих було обрано сучасний і перевірений стек вебтехнологій, що дозволяє реалізувати масштабований, продуктивний і зручний у використанні застосунок.

Клієнтська частина реалізована за допомогою фреймворку React, який забезпечує компонентний підхід до побудови інтерфейсу, повторне використання коду та реактивне оновлення стану сторінок. Для керування глобальним станом використовується Redux Toolkit, що дозволяє централізовано контролювати дані в межах сесії користувача.

Серверна частина реалізована на базі Node.js з використанням фреймворку Express, який забезпечує обробку HTTP-запитів, маршрутизацію, підключення до бази даних, обробку помилок, авторизацію та інші ключові функції. Для авторизації обрано механізм JWT, що дозволяє захищати особисті дані та розмежовувати доступ користувачів [16].

База даних реалізована з використанням PostgreSQL, реляційної системи управління базами даних, яка забезпечує підтримку складних зв’язків між сутностями, транзакцій, індексації та цілісності даних.

Додатково в клієнтській частині використовуються:

- React Router – для маршрутизації між сторінками;
- Axios – для здійснення HTTP-запитів до API;
- Formik + Yup – для обробки форм і валідації даних;
- MUI або TailwindCSS – для створення адаптивного інтерфейсу.

Середовище розробки – Visual Studio Code, система контролю версій – Git.

Такий набір технологій дозволяє забезпечити високу продуктивність, зрозумілий та масштабований код, а також підтримку кросбраузерності та адаптивності для різних пристроїв.

3.2 Реалізація клієнтської частини

Клієнтська частина вебдодатку реалізована за допомогою бібліотеки React, що дозволяє побудувати сучасний компонентний інтерфейс користувача з високою продуктивністю. Для керування станом, маршрутизації, обробки форм та комунікації з сервером використано набір актуальних інструментів, що разом формують гнучку фронтенд-архітектуру.

Процес авторизації реалізовано через JWT-токени (рис. 3.1), що є одним із найпоширеніших і безпечних способів аутентифікації у вебзастосунках. Після заповнення форми входу користувач надсилає свої облікові дані у вигляді HTTP POST-запиту до відповідного ендпоінту API. Сервер здійснює перевірку наявності користувача в базі даних та правильності введеного пароля. У разі успішної перевірки створюється JWT-токен, який містить зашифровану інформацію про ID користувача, його роль та термін дії сесії [17].

Отриманий токен зберігається на клієнтській стороні в Redux-сховищі або у локальному сховищі при необхідності збереження після перезавантаження сторінки, після чого додається до заголовків усіх наступних запитів до захищених маршрутів. Це дозволяє здійснювати авторизовані дії, зокрема перегляд власного кабінету, оформлення замовлень, додавання товарів, залишення відгуків, перегляд історії покупок тощо. Такий механізм забезпечує безперервну сесію користувача без необхідності повторної автентифікації при кожному оновленні сторінки. У разі закінчення терміну дії токена система автоматично перенаправляє користувача на сторінку входу, що гарантує безпеку доступу до персональних даних. Використання JWT також дозволяє зручно реалізувати розмежування прав доступу – наприклад, відокремити функціонал для адміністратора, продавця та покупця. Це критично важливо для маркетплейсу-конфігуратора, де кожна категорія користувачів взаємодіє з системою по-різному. Зберігання токена на клієнті, з одного боку, забезпечує високу швидкодію і мінімальні затримки при обміні даними, а з іншого – вимагає додаткового захисту від XSS-атак, що реалізується за допомогою налаштувань безпечного доступу до сховища.

```
const authSlice = createSlice({
  name: 'auth',
  initialState: { token: null, user: null },
  reducers: {
    setCredentials: (state, action) => {
      state.token = action.payload.token;
      state.user = action.payload.user;
    },
    logout: (state) => {
      state.token = null;
      state.user = null;
    },
  },
});
```

Рисунок 3.1 – Реалізація процесу авторизації

Кошик реалізовано на основі Redux (рис. 3.2) – як глобальний стан застосунку, який доступний на будь-якій сторінці незалежно від глибини вкладеності компонентів. Такий підхід дозволяє централізовано управляти вмістом кошика та синхронізувати його з іншими частинами інтерфейсу, зокрема при оформленні замовлення чи переході між сторінками. При натисканні на кнопку «Додати до кошика» викликається відповідна Redux-дія, яка перевіряє, чи вже існує товар у поточному стані кошика. Якщо так – збільшується лічильник кількості товару, інакше – додається новий об’єкт з усіма атрибутами товару.

Кошик зберігається у пам’яті браузера під час поточної сесії, що дає змогу працювати з ним навіть без авторизації. Після входу до облікового запису реалізовано механізм синхронізації: локальний вміст кошика об’єднується з наявним у базі даних, що дозволяє зберегти вибір користувача незалежно від моменту входу в систему. Такий підхід забезпечує гнучкість, зручність та безперервність взаємодії з інтерфейсом, що є критично важливим для користувацького досвіду у маркетплейсах.

```
const cartSlice = createSlice({
  name: 'cart',
  initialState: { items: [] },
  reducers: {
    addToCart: (state, action) => {
      const item = state.items.find(i => i.id === action.payload.id);
      if (item) {
        item.quantity += 1;
      } else {
        state.items.push({ ...action.payload, quantity: 1 });
      }
    },
    removeFromCart: (state, action) => {
      state.items = state.items.filter(i => i.id !== action.payload);
    },
  },
});
```

Рисунок 3.2 – Реалізація кошику

Для зручної роботи з формами та перевірки введених користувачем даних у клієнтській частині застосунку використовується бібліотека Formik у поєднанні з Yup. Такий тандем дозволяє ефективно керувати станом форм, автоматично відслідковувати зміну значень, обробку помилок і валідацію введених даних відповідно до заданих правил. Наприклад, можна визначити, що поле email має бути у форматі електронної адреси, пароль – мати мінімальну довжину, а обов’язкові поля не повинні залишатись порожніми [18].

Це дозволяє уникнути типових помилок при реєстрації, вході та оформленні замовлень, ще до надсилання запиту на сервер. Всі перевірки виконуються на боці клієнта в режимі реального часу, що покращує користувацький досвід: система одразу повідомляє про неправильне введення даних або незаповнені обов’язкові поля. Крім того, Formik полегшує повторне використання логіки форм у різних компонентах, зменшуючи дублювання коду та підвищуючи підтримуваність проєкту.

Фрагмент коду, що відповідає за обробку форм і перевірку даних, зображено на рисунку 3.3.

```
<Formik
 initialValues={{ email: '', password: '' }}
 validationSchema={schema}
 onSubmit={handleLogin}
>
 {({ handleChange, handleSubmit, errors }) => (
  <form onSubmit={handleSubmit}>
   <input name="email" onChange={handleChange} />
   <div>{errors.email}</div>

   <input name="password" type="password" onChange={handleChange} />
   <div>{errors.password}</div>

   <button type="submit">Увійти</button>
  </form>
 )}
</Formik>
```

Рисунок 3.3 – Фрагмент коду що відповідає за обробку форм та валідацію

У підсумку, клієнтська частина додатку побудована з урахуванням принципів модульності, повторного використання компонентів та захисту приватних ресурсів. Це дозволяє забезпечити ефективну взаємодію користувачів із системою, комфортну навігацію та збереження даних у межах одного інтерфейсу.

3.3 Реалізація серверної частини

Серверна частина вебдодатку реалізована з використанням Node.js у поєднанні з фреймворком Express.js, що дозволяє створити гнучкий, масштабований та зручний у підтримці REST API для обробки HTTP-запитів від клієнтської частини. Така архітектура дає змогу чітко розділити відповідальність між різними шарами застосунку, дотримуючись принципів модульності та повторного використання коду. Уся логіка поділена на окремі контролери, що обробляють вхідні запити, маршрути, які визначають шляхи доступу до

функціоналу, та middleware-функції, які перехоплюють запити для валідації, авторизації, логування або обробки помилок.

Сервер виконує низку критично важливих задач, зокрема – автентифікацію користувачів, авторизацію, взаємодію з базою даних, перевірку введених даних, а також формування відповідей у форматі JSON, що забезпечує єдину структуру комунікації з фронтендом.

У процесі розробки було реалізовано механізм авторизації через JWT – сучасний підхід до збереження сесій без використання серверного стану. Після надсилання користувачем облікових даних через форму входу, сервер проводить перевірку відповідного запису в базі. Якщо обліковий запис знайдено, сервер порівнює отриманий пароль з хешованим значенням, збереженим у базі даних. У разі успішної автентифікації сервер генерує токен, який містить інформацію про ідентифікатор користувача, його роль у системі, а також термін дії сесії.

Отриманий токен передається клієнту та зберігається на стороні браузера – зазвичай у Redux, LocalStorage або HttpOnly cookie залежно від рівня безпеки. При кожному подальшому запиті до захищених маршрутів клієнт автоматично додає цей токен до заголовків запиту. Сервер, у свою чергу, перевіряє токен за допомогою секретного ключа, і якщо він дійсний – виконує необхідну дію. Усі запити без токена або з недійсним токеном блокуються відповідним статусом помилки, що забезпечує захист персоналізованих даних та обмеження доступу до функцій залежно від ролі.

```
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');

exports.login = async (req, res) => {
  const { email, password } = req.body;
  const user = await User.findOne({ where: { email } });

  if (!user || !(await bcrypt.compare(password, user.password))) {
    return res.status(401).json({ message: 'Невірний логін або пароль' });
  }

  const token = jwt.sign(
    { id: user.id, role: user.role },
    process.env.JWT_SECRET,
    {
      expiresIn: '2h',
    }
  );

  res.json({ token, user });
};
```

Рисунок 3.4 – Механізм авторизації через JWT

Щоб захистити приватні маршрути, такі як створення замовлень, перегляд особистого кабінету чи додавання нових товарів, у серверній частині застосунку використовується спеціальний middleware (рис. 3.5), який виконує роль проміжного рівня перевірки автентичності. Ця функція автоматично перехоплює всі запити, що надходять до захищених маршрутів, та виконує перевірку наявності та валідності JWT-токена у заголовках HTTP-запиту.

У випадку, якщо токен відсутній, має неправильний формат або не проходить верифікацію за допомогою секретного ключа, middleware негайно припиняє обробку запиту та повертає клієнту відповідь зі статусом 401 або 403. Це дозволяє запобігти несанкціонованому доступу до функціоналу, призначеного лише для зареєстрованих і автентифікованих користувачів. У свою чергу, якщо токен є валідним, middleware декодує його вміст і додає інформацію про користувача до об'єкта request, що робить її доступною для подальших контролерів.

Такий підхід забезпечує гнучкий механізм контролю доступу і дозволяє легко масштабувати систему – наприклад, додавати обмеження для ролей або захищати окремі ресурси залежно від прав доступу. В результаті middleware виступає ключовим елементом безпеки та керування правами в межах усієї серверної архітектури.

```
const jwt = require('jsonwebtoken');

module.exports = (req, res, next) => {
  const authHeader = req.headers.authorization;
  const token = authHeader && authHeader.split(' ')[1];
  if (!token) return res.sendStatus(401);

  jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
    if (err) return res.sendStatus(403);
    req.user = user;
    next();
  });
};
```

Рисунок 3.5 – Реалізація захисту приватних маршрутів

Коли авторизований продавець бажає додати товар до маркетплейсу, він заповнює відповідну форму на клієнтській частині інтерфейсу, яка містить поля для введення назви товару, опису, ціни, категорії, технічних характеристик та завантаження зображень. Після заповнення форми дані надсилаються до API сервера через HTTP POST-запит. Серверна частина обробляє цей запит, перевіряє валідність переданих даних, після чого створює новий запис у таблиці products бази даних.

Ключовим моментом є те, що до нового товару обов’язково прив’язується ID продавця, який визначається шляхом розшифрування JWT токена, отриманого з заголовку авторизованого запиту. Це дозволяє забезпечити зв'язок між продавцем і його товарами, а також реалізувати функціонал особистого кабінету, в якому кожен користувач бачить лише власні товари, може їх редагувати, видаляти або переглядати статистику продажів.

Такий підхід виключає можливість маніпуляцій з боку клієнта, оскільки ID продавця не передається явно з форми, а визначається на сервері автоматично. Це підвищує безпеку та надійність платформи. Фрагмент коду, що відповідає за розміщення товару на сторінці маркетплейсу, зображено на рисунку 3.6.

```
exports.createProduct = async (req, res) => {
 const { name, description, price, categoryId } = req.body;

 const product = await Product.create({
  name,
  description,
  price,
  categoryId,
  sellerId: req.user.id,
 });

 res.status(201).json(product);
};
```

Рисунок 3.6 – Реалізація функції додавання товару продавцем

Уся серверна частина написана з урахуванням принципів безпеки, поділу відповідальностей та можливості масштабування функціоналу у майбутньому. Це забезпечує надійність, продуктивність та стабільність вебдодатку в умовах реального використання.

3.4 Висновки

У третьому розділі було реалізовано клієнтську та серверну частини вебдодатку маркетплейсу-конфігуратора комп’ютерних комплектуючих відповідно до обраної архітектури. Для фронтенд-частини застосовано бібліотеку React, що забезпечила модульну структуру застосунку та швидке оновлення інтерфейсу без перезавантаження сторінки. Управління глобальним станом реалізовано за допомогою Redux Toolkit, що дало змогу централізовано зберігати дані користувача, кошика та замовлень. Для маршрутизації

використано React Router, який дозволив реалізувати переходи між сторінками, зокрема приватні маршрути з обмеженням доступу. Валідація форм здійснюється за допомогою бібліотек Formik і Yup, що гарантує коректність введених даних ще до надсилання їх на сервер.

Серверна частина побудована на основі Node.js з використанням фреймворку Express, що забезпечує обробку HTTP-запитів, автентифікацію через JWT, перевірку прав доступу та взаємодію з базою даних. Усі основні API-маршрути згруповано за функціональними блоками: авторизація, робота з товарами, замовленнями, користувачами та кошиком. Для зберігання даних використано PostgreSQL, у якій реалізовано реляційні зв’язки між таблицями з урахуванням цілісності та нормалізації. Представлені фрагменти коду демонструють реалізацію ключових функцій, таких як: реєстрація та вхід користувача, додавання товарів продавцем, формування та збереження замовлень, обробка кошика.

Побудована архітектура застосунку дозволяє ефективно масштабувати систему, додавати нові модулі та інтегрувати додаткові сервіси без необхідності суттєвих змін у базовому коді.

4 ТЕСТУВАННЯ ЗАСТОСУНКУ

4.1 Аналіз методів тестування

Тестування – це важливий етап життєвого циклу програмного забезпечення, що дозволяє оцінити якість реалізованої системи та виявити потенційні помилки до її впровадження в реальне використання. Його основною метою є перевірка відповідності фактичної поведінки програмного продукту очікуваним результатам, а також перевірка стійкості системи до помилкових або непередбачуваних дій користувача.

Оскільки вебзастосунок має справу з реальними користувачами, персональними даними та замовленнями, важливо ще на етапі розробки забезпечити належний рівень перевірки кожного компонента. Недостатнє або неякісне тестування може призвести до критичних помилок, втрати довіри користувачів, фінансових втрат або навіть порушення законодавчих вимог щодо обробки персональних даних. Особливо це стосується таких функцій, як авторизація, реєстрація, оформлення замовлення чи доступ до особистого кабінету продавця – будь-які збої в цих модулях прямо впливають на функціональність і безпеку системи.

Сучасна практика програмної інженерії передбачає використання різних підходів до тестування, кожен з яких має свої переваги та недоліки. Основні типи тестування наведено в таблиці 4.1.

Таблиця 4.1 — Основні типи тестування

Тип тестування	Мета
Модульне	Перевірка окремих функцій або компонентів в ізоляції
Інтеграційне	Перевірка взаємодії між модулями або підсистемами
Системне	Оцінка поведінки повної системи в цілому
Регресійне	Перевірка, що нові зміни не порушили вже реалізований функціонал
Функціональне	Перевірка реалізації бізнес-вимог і сценаріїв користувача

Продовження таблиці 4.1

Тип тестування	Мета
Нефункціональне	Тестування продуктивності, безпеки, зручності використання

Одним із найрозповсюдженіших є модульне тестування, що дозволяє ізолювати окремі компоненти системи та перевірити їхню поведінку незалежно від інших частин. Цей підхід зручний при автоматизації перевірок і дозволяє швидко локалізувати джерело помилки. Для реалізації модульних тестів у Node.js зазвичай використовують бібліотеки типу Jest, Mocha або AVA.

Інтеграційне тестування орієнтоване на перевірку взаємодії між окремими модулями, наприклад, клієнтської частини та сервера, або сервера й бази даних. Воно допомагає переконатись у правильності обробки запитів і відповіді, але вимагає більш складного налаштування середовища.

Функціональне тестування – один з найважливіших етапів у межах вебзастосунків, оскільки перевіряє відповідність поведінки системи очікуванням користувача. Наприклад, натискання кнопки "Оформити замовлення" повинне призводити до створення відповідного запису в базі даних. Це тестування є найближчим до реального використання системи.

Окрім того, у проєктах із користувацьким інтерфейсом часто застосовується UI/UX тестування, яке включає візуальну перевірку елементів інтерфейсу, правильність відображення на різних пристроях та екранах. Для автоматизації UI тестів можуть використовуватись інструменти типу Selenium, Cypress або Puppeteer. Таке тестування дозволяє виявити проблеми з адаптивністю, контрастністю елементів, поведінкою модальних вікон та іншими аспектами взаємодії користувача з інтерфейсом.

Також важливим є поняття ручного функціонального тестування, яке було обране для перевірки реалізованого маркетплейсу-конфігуратора. Цей підхід дозволяє швидко перевірити основні бізнес-процеси без потреби у складному налаштуванні середовища. Особливо він є доцільним на ранніх етапах розробки

або при тестуванні мінімально життєздатного продукту, коли система постійно змінюється. У таких умовах ручне тестування дає змогу оперативно виявляти помилки та відразу вносити необхідні зміни до коду.

Приклад такого тестування наведено у таблиці 4.2.

Таблиця 4.2 – Тестування маркетплейсу-конфігуратора

ID	Назва	Методика тестування	Очікуваний результат	Результат
1	Реєстрація нового користувача	Відкрити сторінку реєстрації. Заповнити email, пароль та номер телефону. Натиснути на кнопку «Зареєструватись»	Створено обліковий запис, перенаправлення на головну сторінку	Виконано (рис. 4.1)
2	Авторизація користувача	Відкрити сторінку входу. Ввести правильні облікові дані. Натиснути кнопку «Увійти»	Перехід до персонального кабінету	Виконано
3	Додавання товару до кошика	Увійти в систему. Відкрити сторінку товару. Натиснути «Додати до кошика»	Товар з’являється у кошику, індикатор кількості оновлюється	Виконано (рис. 4.4)

Продовження таблиці 4.2

4	Оформлення замовлення	Перейти до кошика. Натиснути «Оформити замовлення». Заповнити контактні дані та підтвердити	Замовлення зберігається в базі. Повідомлення про успішну операцію.	Виконано (рис. 4.5)
5	Додавання нового товару продавцем	Увійти як продавець. Перейти до кабінету продавця. Заповнити форму додавання товару та натиснути «Додати»	Новий товар збережено в базі. Статус – «На модерації»	Виконано (рис. 4.6)
6	Некоректна електронна пошта при реєстрації	Відкрити форму реєстрації. Ввести неіснуючу електронну пошту. Натиснути «Зареєструватися»	Повідомлення про помилку через неіснуючу адресу електронної пошти	Виконано
7	Оформлення замовлення без товарів у кошику	Увійти до системи. Очистити кошик. Спробувати оформити замовлення	Відображення повідомлення про порожній кошик	Виконано
8	Синхронізація кошика після авторизації	Додати товар до кошика в неавторизованому режимі.	Дані з локального кошика збережено до профілю користувача	Виконано

Продовження таблиці 4.2

		Увійти до облікового запису.		
9	Фільтрація товарів	Перейти на головну сторінку. Встановити фільтр за ціною. Оновити список	Відображення лише тих товарів, що підходять під встановлену фільтрацію.	Виконано (рис. 4.3)
10	Створення конфігурації ПК	Відкрити конфігуратор. Вибрати комплектуючі з кожної категорії. Натиснути «Зберегти»	Формування збірки з перевіркою сумісності, вивід загальної вартості	Виконано (рис. 4.7)

Крім того, при тестуванні важливо враховувати граничні умови та можливі виключні ситуації – наприклад, спробу оформити замовлення без доданих товарів або введення некоректного формату email при реєстрації. Саме такі сценарії дозволяють виявити критичні вразливості та недоопрацювання у логіці застосунку ще до його публічного запуску.

Окремої уваги заслуговує юзер-тестування – це процес, під час якого реальні користувачі взаємодіють із системою, а розробники спостерігають за їх поведінкою, труднощами та реакціями. У межах даного проєкту такий формат може бути реалізований у вигляді спостереження за тестуванням з боку одногрупників або викладача. Юзер-тестування дозволяє не лише виявити технічні проблеми, а й отримати цінний зворотний зв’язок щодо зручності інтерфейсу, інтуїтивності навігації, швидкості виконання дій та загального задоволення від користування.

Таким чином, аналіз методів тестування свідчить про доцільність

поєднання ручного функціонального підходу на етапі розробки з потенційною автоматизацією модульних і інтеграційних тестів у майбутньому.

4.2 Тестування застосунку

Для оцінювання роботи застосунку були використані інтерактивні тестові сценарії, які охоплювали ключові дії користувача: реєстрацію нового облікового запису, авторизацію, перегляд товарів, додавання до кошика, застосування фільтрів, оформлення замовлення, а також додавання товару через кабінет продавця. Кожен сценарій передбачав виконання послідовності дій, аналогічних до поведінки реального користувача, з перевіркою результатів на кожному кроці. У процесі тестування зверталася увага на швидкість завантаження сторінок, коректність обробки введених даних, відображення повідомлень про помилки та валідацію форм.

Особлива увага приділялася перевірці адаптивності інтерфейсу – тестування проводилось як на комп’ютері, так і на мобільному пристрої. Було проаналізовано, наскільки зручно користувач може виконувати ті самі дії на різних екранах. Додатково були протестовані ситуації, коли користувач вводить некоректні дані, наприклад неправильний email, порожні поля або надто короткий пароль. Усі такі сценарії дозволили виявити дрібні неточності у валідації та відразу їх усунути. На основі проведеного ручного тестування було підтверджено коректність усіх зазначених функцій, стабільність роботи застосунку та відповідність реалізованого функціоналу технічному завданню.

Користувацький досвід було оцінено як позитивний – застосунок виявився інтуїтивно зрозумілим, із передбачуваною логікою дій, що відповідає сучасним вимогам до UX. Усі основні сценарії, зокрема реєстрація, додавання товарів, оформлення замовлень та керування кошиком, були успішно перевірені без виникнення критичних помилок. Перевірку даних, повідомлення про помилки та реакція інтерфейсу на дії користувача реалізовані коректно й сприяють комфортній роботі. Застосунок показав стабільну роботу як у десктопній, так і в мобільній версіях, що підтверджує адаптивність інтерфейсу та відповідність

очікуванням кінцевих користувачів.

Скріншот інтерфейсу форми реєстрації (рис. 4.1) демонструє можливість створення нового облікового запису користувача з базовими полями – електронною поштою, паролем та номером телефону. У разі успішної реєстрації користувач автоматично перенаправляється на головну сторінку.

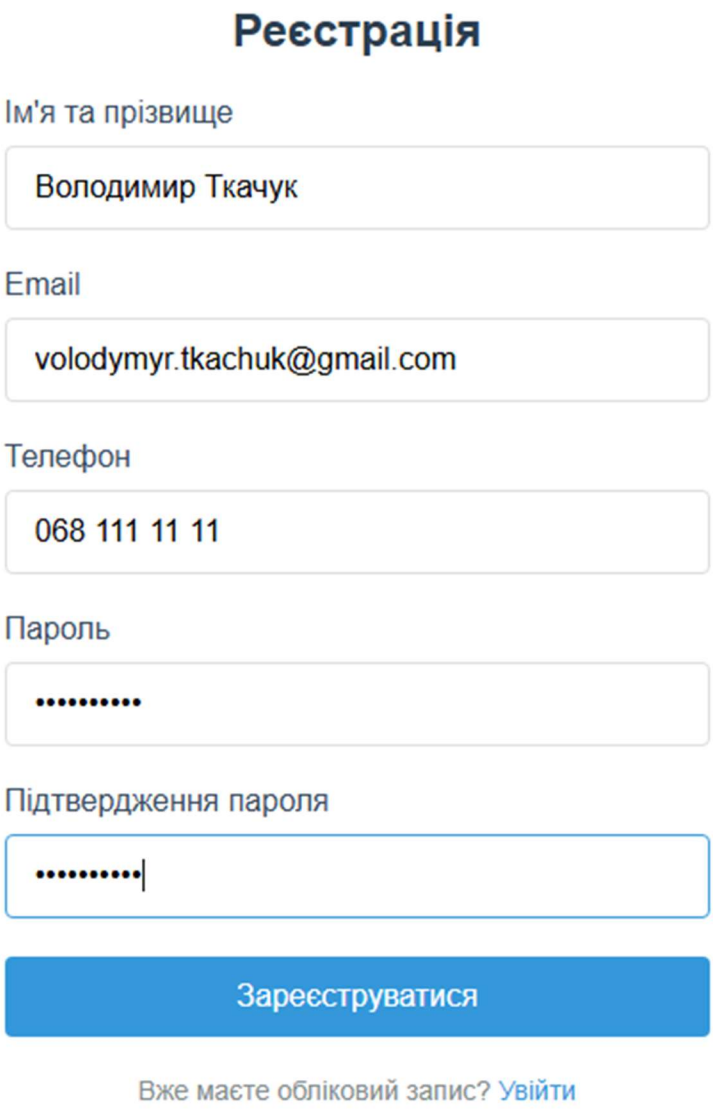

Рисунок 4.1 — Реєстрація користувача

Після входу в систему користувач отримує доступ до персоналізованих функцій, зокрема можливість переглядати історію своїх замовлень, редагувати профіль, залишати відгуки, додавати товари в кошик тощо. Інтерфейс головної сторінки (рис. 4.2) реалізований у вигляді адаптивної сітки, що відображає

каталог доступних товарів. Важливою особливістю є підтримка динамічного фільтрування – користувач може швидко обмежити результати пошуку, застосувавши один або кілька фільтрів за категоріями, ціновим діапазоном, технічними параметрами або наявністю товару (рис. 4.3).

Використання асинхронного оновлення через API дозволяє миттєво змінювати список товарів без перезавантаження сторінки, що суттєво підвищує зручність користування. Усі фільтри інтегровані з URL-рядком, що дозволяє зберігати стан пошуку при навігації або поширенні посилання. Така реалізація відповідає сучасним вимогам до e-commerce систем і створює комфортні умови для пошуку й вибору продукції на платформі.

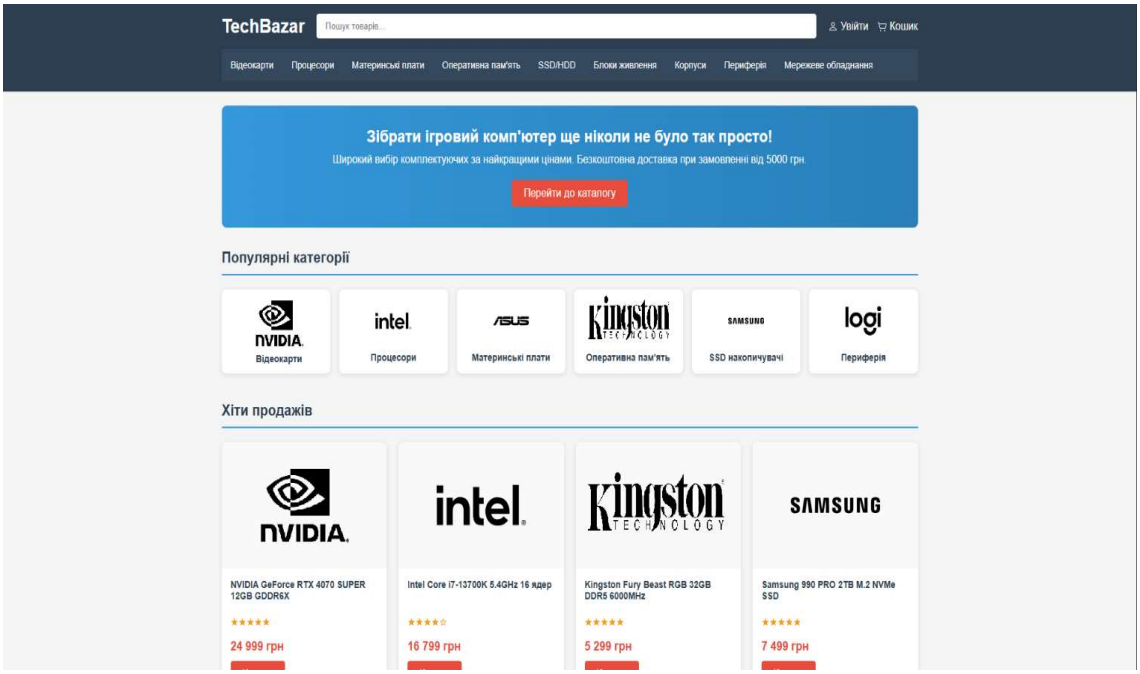

Рисунок 4.2 — Головна сторінка додатку

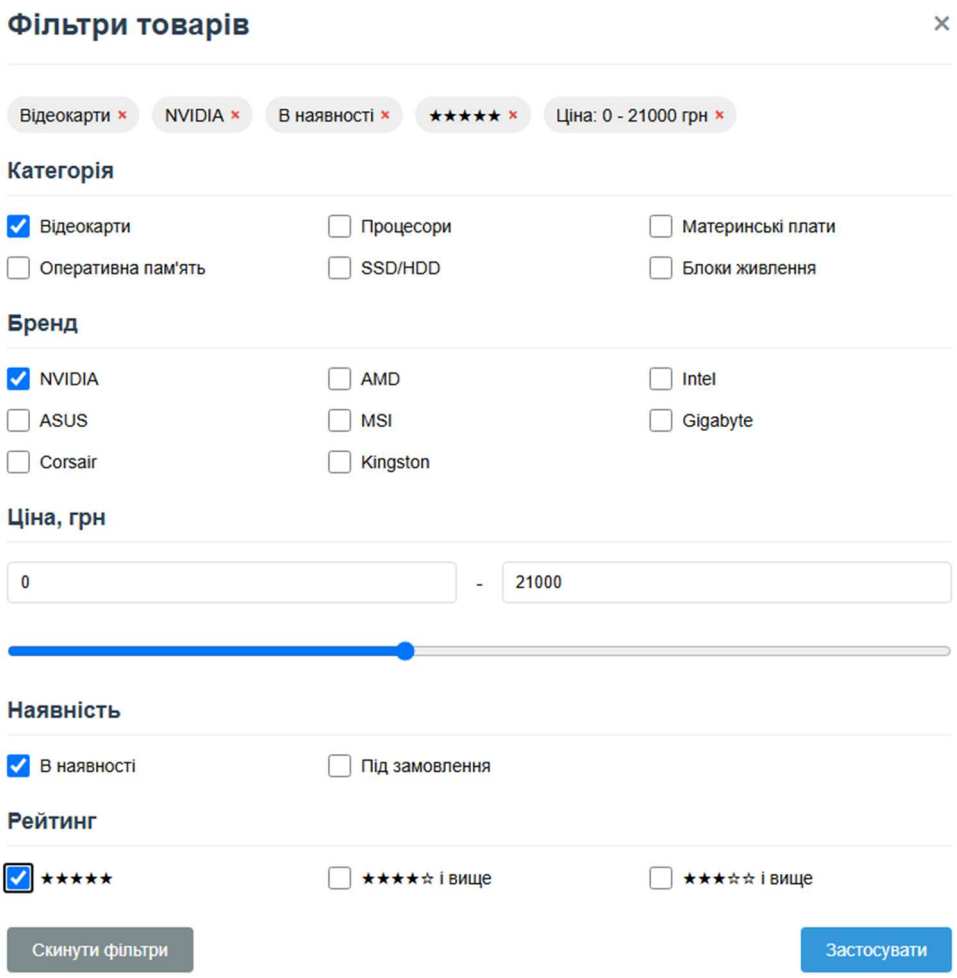

Рисунок 4.3 — Фільтри товарів

На зображенні сторінки кошика (рис. 4.4) продемонстровано механізм додавання товарів, зміну їх кількості та відображення загальної вартості замовлення. Користувач має змогу в один клік додати позицію до кошика, збільшити або зменшити кількість одиниць, а також переглянути підсумкову суму, що динамічно оновлюється при кожній зміні. Така функціональність реалізована через глобальний стан Redux, що забезпечує миттєву реакцію інтерфейсу без перезавантаження сторінки.

Крім того, система фільтрації товарів, представлена на окремому модальному вікні, дозволяє користувачеві зручно налаштувати відображення товарів відповідно до своїх потреб. Фільтри охоплюють основні параметри: категорії, бренди, діапазон цін, наявність та рейтинг товару. Вибрані параметри відображаються у вигляді тегів у верхній частині форми, які можна швидко

видалити. Передбачено як ручне введення цінового діапазону, так і переміщення повзунка, що особливо зручно на мобільних пристроях. Реалізація такої багаторівневої системи фільтрів дозволяє забезпечити точний підбір товарів, підвищити конверсію та покращити загальний досвід користувача під час взаємодії з додатком.

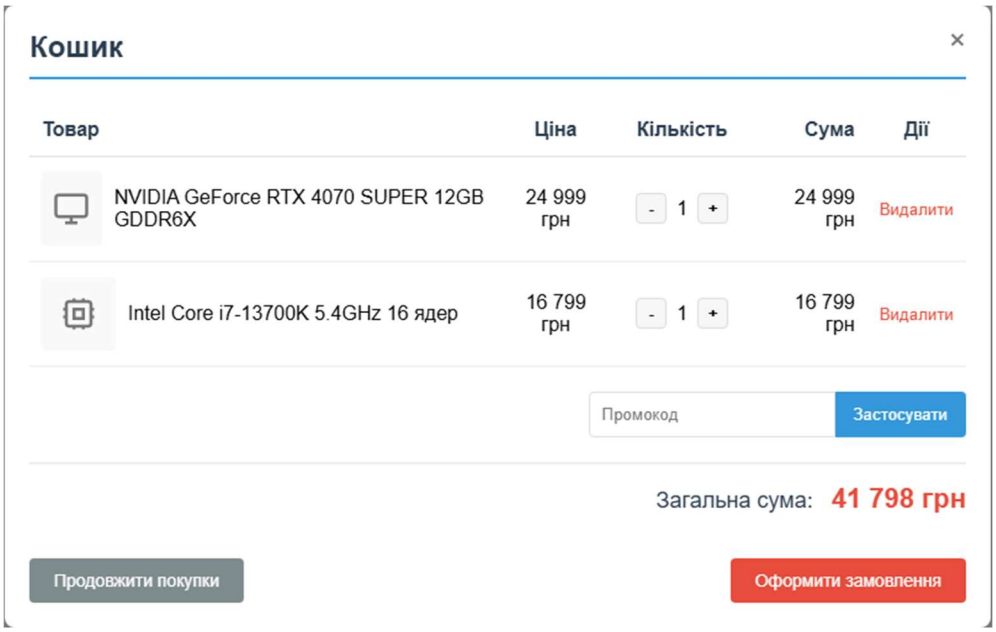

Рисунок 4.4 — Додавання товарів у кошик

Окремий інтерфейс оформлення замовлення (рис. 4.5) дозволяє користувачеві ввести контактні дані, вибрати спосіб доставки та підтвердити покупку. Форма побудована таким чином, щоб мінімізувати кількість дій і полегшити процес оформлення навіть для недосвідчених користувачів. Перед підтвердженням система додатково перевіряє заповнені поля на наявність помилок, що дозволяє уникнути некоректних або неповних даних. Після успішного оформлення замовлення користувач отримує повідомлення про завершення операції, а замовлення збігається в базі даних із присвоєним унікальним ідентифікатором.

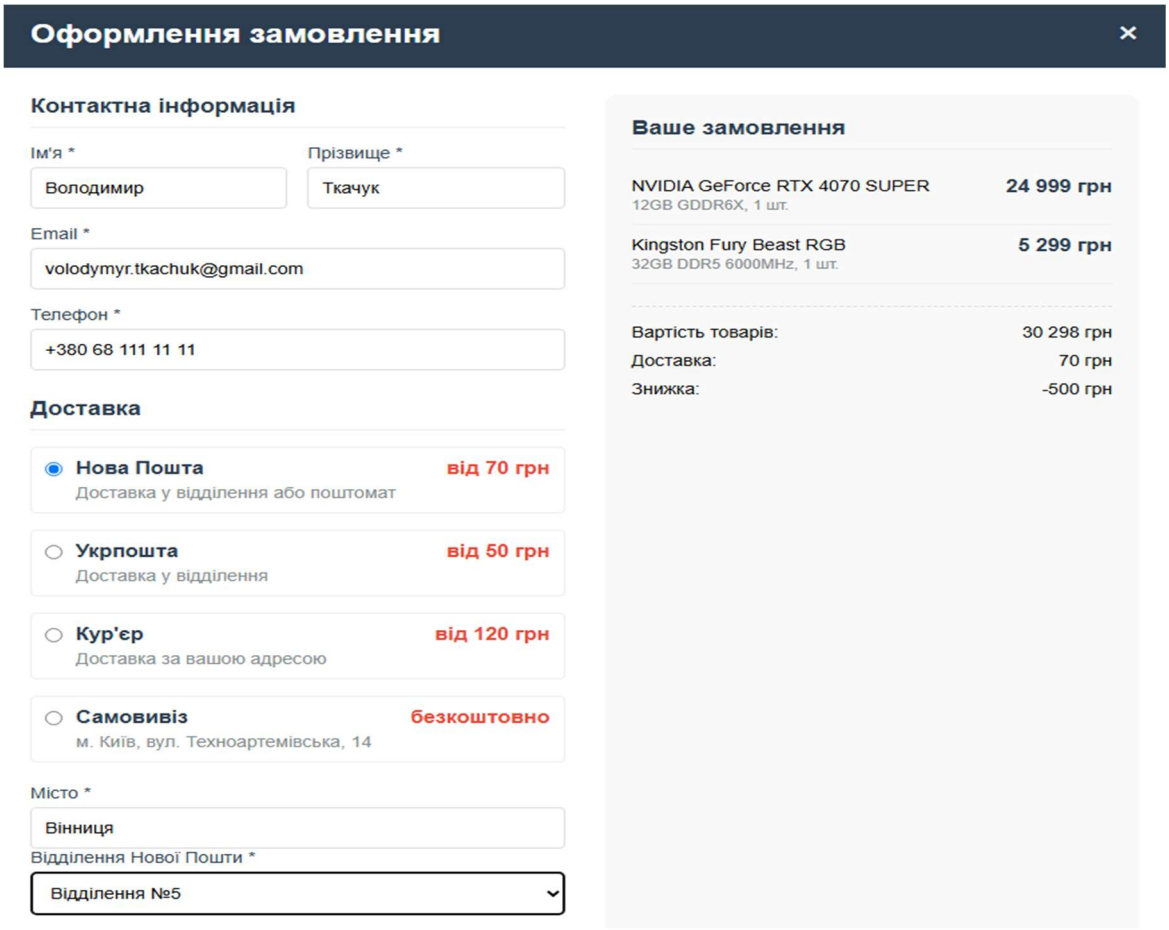

Рисунок 4.5 — Вікно оформлення замовлень

Також у межах реалізації застосунку було створено сторінку кабінету продавця (рис. 4.6) з можливістю додавання нового товару. Цей інтерфейс є доступним лише для користувачів із відповідною роллю, що перевіряється системою авторизації. Процес додавання товару реалізовано через зручну форму, яка передбачає введення основної інформації: назви, опису, ціни, вибору категорії, технічних характеристик, гарантійного терміну, а також можливість завантаження одного або кількох зображень товару. Перед відправленням даних проводиться перевірка правильності введення, а після створення товар автоматично зберігається у базі з прив'язкою до відповідного продавця.

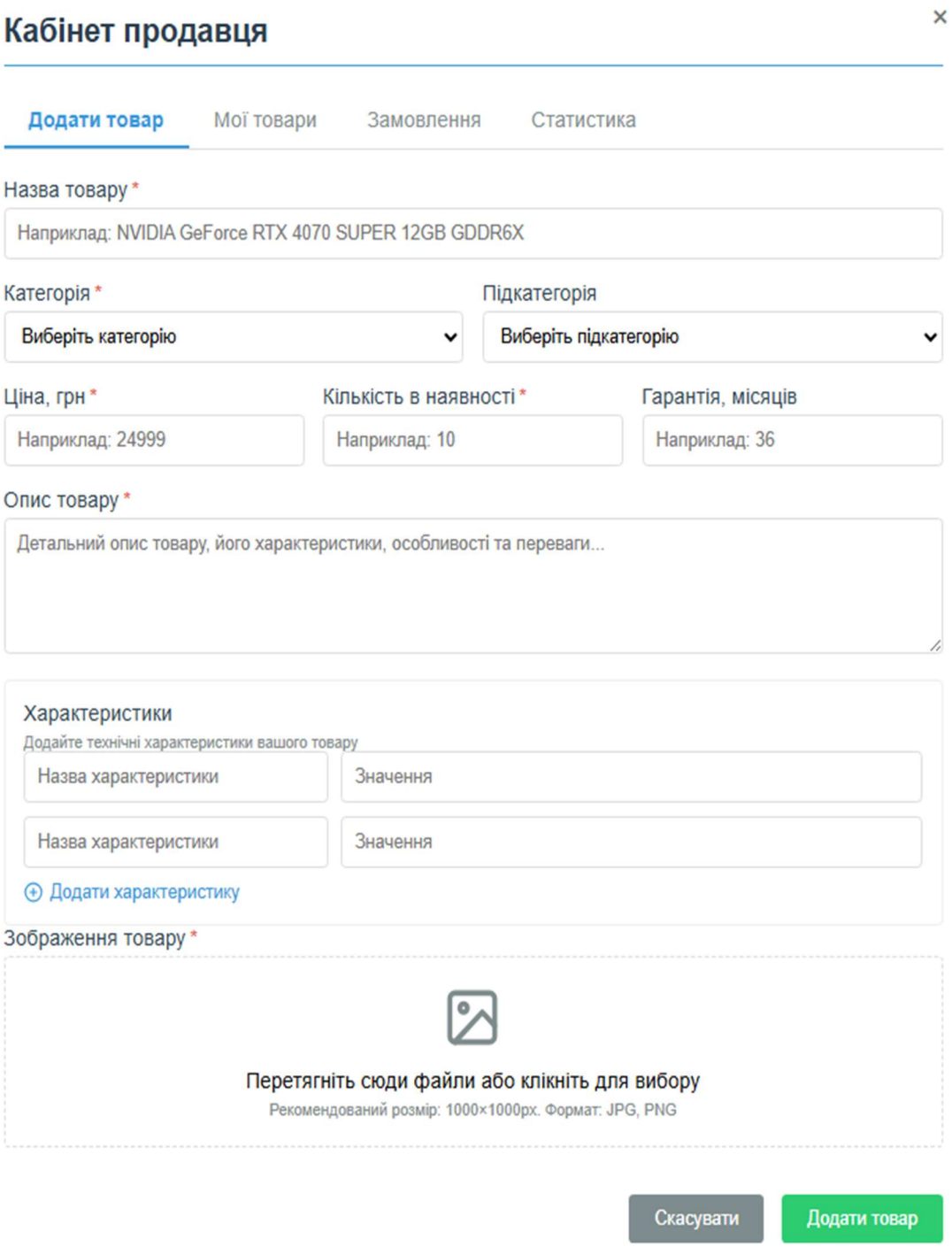

Рисунок 4.6 — Кабінет продавця

На зображенні інтерфейсу конфігуратора персонального комп’ютера (рис. 4.7) продемонстровано механізм вибору ключових компонентів системи, таких як процесор, материнська плата, відеокарта, оперативна пам’ять, накопичувачі, блок живлення та корпус. Для кожного з елементів вказано обов’язковість вибору, що запобігає формуванню некоректної конфігурації. У правій частині інтерфейсу

відображається підсумкова вартість, статус доставки та орієнтовна продуктивність системи у різних режимах. Функціональність реалізована на основі динамічної перевірки вибраних параметрів і забезпечує зручний спосіб збереження, очищення чи додавання конфігурації до кошика.

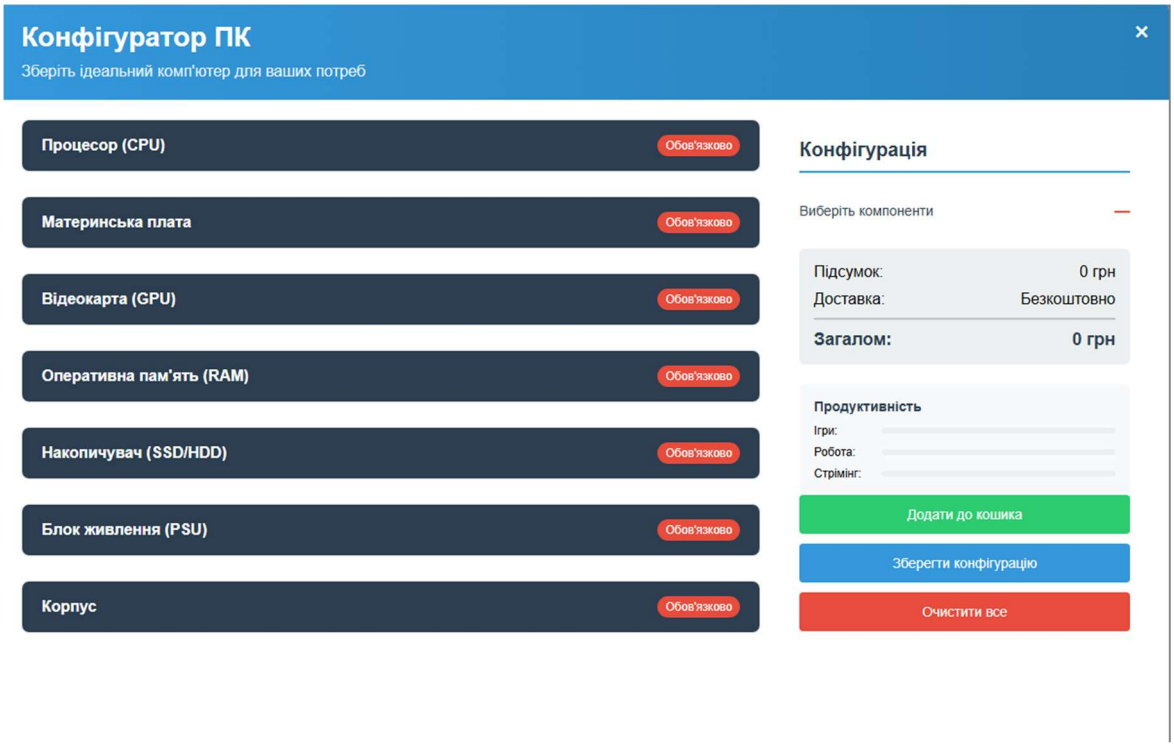

Рисунок 4.7 — Конфігуратор персонального комп’ютера

Проведене візуальне оцінювання та взаємодія з кожною із функціональних частин системи підтверджують відповідність застосунку вимогам, сформульованим у постановці задач. Інтерфейс є інтуїтивно зрозумілим, усі дії виконуються без помилок або некоректної поведінки. Реалізований функціонал покриває основні процеси, необхідні для повноцінної роботи додатку, а архітектура додатку дозволяє безперешкодно масштабувати систему у разі потреби.

4.3 Висновки

У четвертому розділі було проведено тестування вебдодатку та виконано оцінювання його роботи відповідно до сформульованих функціональних вимог.

Для перевірки працездатності системи було обрано підхід ручного функціонального тестування, що дав змогу послідовно перевірити ключові сценарії взаємодії користувача з платформою. Зокрема, було протестовано процеси реєстрації нового користувача, авторизації через JWT, додавання товарів продавцем, роботу з кошиком – включаючи додавання, оновлення кількості та видалення позицій – а також повний цикл оформлення замовлення. Кожен сценарій було виконано з різними комбінаціями вхідних даних для перевірки стійкості системи до типових помилок і виняткових ситуацій.

Окрема увага приділялася перевірці валідації форм, поведінці інтерфейсу при некоректних введеннях та відображенню відповідних повідомлень. Також було перевірено коректну синхронізацію кошика при переході з неавторизованої до авторизованої сесії, що є важливою частиною користувацького досвіду. Аналіз отриманих результатів показав, що всі основні функції працюють стабільно, без критичних збоїв, і відповідають вимогам, поставленим у першому розділі. Результати тестування підтверджують технічну готовність застосунку до використання та закладають основу для подальшого розвитку системи з урахуванням зворотного зв’язку від користувачів.

Інтерфейс додатку продемонстрував зручність та доступність для користувачів, а сама система – готовність до подальшої експлуатації та масштабування [18].

ВИСНОВКИ

У межах бакалаврської кваліфікаційної роботи було створено інтерактивний вебзастосунок, що реалізує функціонал конфігуратора персонального комп'ютера. Проєкт розроблявся відповідно до сучасних підходів у вебінженерії з дотриманням вимог методичних рекомендацій [19, 20]. В якості середовища реалізації було обрано стек технологій JavaScript із використанням бібліотеки React для клієнтської частини та Node.js з Express для серверної логіки.

На початковому етапі дослідження було здійснено аналіз поточного стану ринку застосунків для підбору комп'ютерних комплектуючих. Проведено огляд найпоширеніших рішень, здійснено їх порівняння з розробленою системою, що дозволило виявити ключові обмеження аналогів і підтвердити практичну цінність створення власного вузькоспеціалізованого інструменту. На основі цього було чітко сформульовано задачі бакалаврської кваліфікаційної роботи.

У межах дослідження було успішно виконано всі поставлені задачі:

- проведено аналіз існуючих рішень у сфері ПК-конфігураторів;
- розроблено архітектуру вебзастосунку;
- модифіковано метод перевірки сумісності комп'ютерних компонентів;
- розроблено інтуїтивно зрозумілий інтерфейс користувача;
- реалізовано механізм збереження та експорту шаблонів конфігурацій;
- проведено повноцінне тестування системи.

Проведене тестування за типовими сценаріями користувача підтвердило стабільність, зручність використання та відповідність функціоналу поставленим вимогам. Застосунок може використовуватися як окремий сервіс або вбудовуватися до маркетплейсів комп'ютерної техніки, що підтверджує його практичну доцільність.

Таким чином, результати виконаної роботи повністю відповідають поставленій меті та окресленим задачам, а сам вебзастосунок є ефективним інструментом для підбору сумісних комплектуючих персонального комп'ютера.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ткачук В.Ю., Рейда О.М. Розробка інтерактивного вебзастосунку конфігуратора персонального комп'ютера// *Матеріали студентської науково-практичної конференції Вінницького національного технічного університету.* — Вінниця: ВНТУ, 2024. — С. 1–2. [Електронний ресурс]. – Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25610

2. Створення вебзастосунків. [Електронний ресурс]. – Режим доступу: https://wezom.com.ua/ua/blog/kak-sozdat-veb-prilozhenie

3. Що таке електронна комерція. [Електронний ресурс]. – Режим доступу: https://interkassa.com/blog/shho-take-elektronna-komerciya-e-commerce-dlya-pochatkivciv

4. Low-code/no-code application development overview. [Електронний ресурс]. – Режим доступу: https://www.sap.com/products/technology-platform/build/what-is-low-code-no-code.html

5. Монолітна архітектура ПЗ. [Електронний ресурс]. – Режим доступу: https://qalight.ua/baza-znaniy/shho-take-monolitna-arhitektura/

6. Мікросервісна архітектура ПЗ. [Електронний ресурс]. – Режим доступу: https://qalight.ua/baza-znaniy/shho-take-mikroservisna-arhitektura-pz/

7. Що таке Serverless-архітектура. [Електронний ресурс]. – Режим доступу: https://www.agiliway.com/uk-ua/shho-take-serverless-arhitektura-oglyad-vartist-masshtabuvannya-ta-bezpeka/

8. Клієнт-серверна архітектура. [Електронний ресурс]. – Режим доступу: https://javarush.com/ua/quests/lectures/ua.questservlets.level14.lecture00

9. What is Postgresql. [Електронний ресурс]. – Режим доступу: https://www.ibm.com/think/topics/postgresql

10. Onyx Rose. JSON & APIs: A Developer's Guide to Data Exchange. 1st ed. (Kindle e-book) 3 April 2025. 282 с.

11. Бібліотека для валідації даних. [Електронний ресурс]. – Режим доступу: https://ela.kpi.ua/items/19060ff3-4352-49bd-810b-609e4a043c79

12. Фронтенд та Бекенд розробка. [Електронний ресурс]. – Режим доступу: https://webtune.com.ua/statti/web-rozrobka/frontend-i-backend-rozrobka/

13. React Router. [Електронний ресурс]. – Режим доступу: https://reactrouter.com/

14. Mark Erikson. Modern Redux: A Guide to Redux Toolkit and Best Practices. 1st ed. San Jose: Redux Masters Press, 2023. 304 с.

15. Як використовувати JWT [Електронний ресурс]. – Режим доступу: https://devzone.org.ua/post/iak-vykorystovuvaty-json-web-tokens-jwt-dlia-avtentyfikatsiyi

16. David Herron. Node.js Web Development: Server-side web development made easy with Node 14 using practical examples. 5th ed. Birmingham: Packt Publishing, 2020. 760 с.

17. Kin Lane. The API-First World: Building and Managing APIs for Modern Software. 2nd ed. Farnham: O'Reilly Media, 2022. 312с.

18. Alex Banks, Eve Porcello. Learning React: Modern Patterns for Developing React Apps. 5th ed. Farnham: O'Reilly Media, 2023. 448с.

19. Методичні вказівки до виконання курсових проєктів з дисципліни "Проєктний практикум" для студентів спеціальності 121 "ІПЗ" / Уклад. В.В. Войтко, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 44 с.

20. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення» / уклад. : О. Н. Романюк, Г. О. Черноволик. – Вінниця : ВНТУ, 2022. – 51 с.

ДОДАТКИ

Додаток А – ТЕХНІЧНЕ ЗАВДАННЯ

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«25» березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка інтерактивного вебдодатку конфігуратора персонального комп'ютера»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. каф. ПЗ Рейда О.М.
«24» березня 2025 року.

Виконав:
студент гр. 2ПІ-21б Ткачук В.Ю.
«24» брезня 2025 року.

м. Вінниця – 2025 рік

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка інтерактивного вебдодатку конфігуратора персонального комп'ютера».

Галузь застосування – електронна комерція.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол № 97 від «20» березня 2025 р.

3. Мета та призначення розробки.

Метою кваліфікаційної роботи є створення інтерактивного вебзастосунку конфігуратора персонального комп'ютера, який дозволяє користувачам підбирати сумісні компоненти, формувати конфігурації та отримувати рекомендації щодо оптимального складу ПК залежно від обраної мети використання.

4. Вихідні дані для проведення НДР

1. Фронтенд та Бекенд розробка. [Електронний ресурс]. – Режим доступу: https://webtune.com.ua/statti/web-rozrobka/frontend-i-backend-rozrobka/

2. React Router. [Електронний ресурс]. – Режим доступу: https://reactrouter.com/

3. Express - Node.js web application framework [Електронний ресурс]. – Режим доступу: https://expressjs.com/

4. Alex Banks, Eve Porcello. Learning React: Modern Patterns for Developing React Apps. 5th ed. Farnham: O'Reilly Media, 2023. 448c.

5. Технічні вимоги

Вхідні дані – обрані користувачем комп'ютерні комплектуючі.

Вихідні дані – перевірка сумісності компонентів, загальна вартість конфігурації, сформований список комплектуючих з технічними характеристиками.

6. Конструктивні вимоги.

Інтерфейс вебзастосунку має відповідати сучасним вимогам до зручності,

доступності та адаптивності. Система має бути простою у навігації та забезпечувати зрозуміло взаємодію з користувачем.

Графічна і текстова документація вебзастосунку повинна відповідати чинним стандартам України щодо розробки програмного забезпечення.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

- пояснювальна записка до бакалаврської кваліфікаційної роботи;
- технічне завдання;
- лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану розвитку додатків для купівлі та продажу комплектуючих	25.03.25 – 04.05.25	Вик.
2	Розробка моделі та алгоритмів роботи системи	05.04.25 – 17.04.25	Вик.
3	Варіантний аналіз та вибір мови програмування	18.04.25 – 21.04.25	Вик.
4	Розробка програмного застосунку	22.04.25 – 16.05.25	Вик.
5	Тестування застосунку	17.05.25 – 25.05.25	Вик.
6	Оформлення матеріалів до захисту БКР	25.05.25 – 30.05.25	Вик.

10. Порядок контролю та прийняття.

Усі етапи бакалаврської кваліфікаційної роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

Додаток Б – ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Розробка інтерактивного вебдодатку конфігуратора персонального комп'ютера»
Тип роботи: Бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)
Підрозділ кафедра програмного забезпечення, ФІТКІ
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 7,2 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Рейда О. М. к.т.н., доц. каф. ПЗ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Ткачук В. Ю.
(підпис) (прізвище, ініціали)

Додаток В – ЛІСТИНГ ПРОГРАМИ

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { BrowserRouter } from 'react-router-dom';
import App from './App';
import { AuthProvider } from './context/AuthContext';
import { CartProvider } from './context/CartContext';
import './styles/index.css';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <AuthProvider>
        <CartProvider>
          <App />
        </CartProvider>
      </AuthProvider>
    </BrowserRouter>
  </React.StrictMode>
);

import React from 'react';
import { Routes, Route } from 'react-router-dom';
import Header from './components/layout/Header';
import Footer from './components/layout/Footer';
import HomePage from './pages/HomePage';
import ProductPage from './pages/ProductPage';
import CategoryPage from './pages/CategoryPage';
import CartPage from './pages/CartPage';
import CheckoutPage from './pages/CheckoutPage';
import LoginPage from './pages/LoginPage';
import RegisterPage from './pages/RegisterPage';
import ProfilePage from './pages/ProfilePage';
import SellerDashboardPage from './pages/SellerDashboardPage';
import AddProductPage from './pages/AddProductPage';
import EditProductPage from './pages/EditProductPage';
```

```
import OrdersPage from './pages/OrdersPage';
import NotFoundPage from './pages/NotFoundPage';
import PrivateRoute from './components/routes/PrivateRoute';
import SellerRoute from './components/routes/SellerRoute';
import './styles/App.css';

function App() {
 return (
  <div className="app">
   <Header />
   <main className="main-content">
    <Routes>
     <Route path="/" element={<HomePage />} />
     <Route path="/category/:categoryId" element={<CategoryPage />} />
     <Route path="/product/:productId" element={<ProductPage />} />
     <Route path="/cart" element={<CartPage />} />
     <Route path="/login" element={<LoginPage />} />
     <Route path="/register" element={<RegisterPage />} />
     <Route path="/checkout" element={
      <PrivateRoute>
       <CheckoutPage />
      </PrivateRoute>
     } />
     <Route path="/profile" element={
      <PrivateRoute>
       <ProfilePage />
      </PrivateRoute>
     } />
     <Route path="/orders" element={
      <PrivateRoute>
       <OrdersPage />
      </PrivateRoute>
     } />
     <Route path="/seller/dashboard" element={
      <SellerRoute>
       <SellerDashboardPage />
      </SellerRoute>
     } />
```

```
          <Route path="/seller/add-product" element={
            <SellerRoute>
              <AddProductPage />
            </SellerRoute>
          } />
          <Route path="/seller/edit-product/:productId" element={
            <SellerRoute>
              <EditProductPage />
            </SellerRoute>
          } />
          <Route path="*" element={<NotFoundPage />} />
        </Routes>
      </main>
      <Footer />
    </div>
  );
}

export default App;

import React, { createContext, useState, useEffect } from 'react';
import api from '../api/api';

export const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  // Перевірка аутентифікації при завантаженні застосунку
  useEffect(() => {
    const checkAuth = async () => {
      try {
        const token = localStorage.getItem('token');
        if (token) {
          api.defaults.headers.common['Authorization'] = `Bearer ${token}`;
          const response = await api.get('/users/me');
```

```
        setUser(response.data);
      }
    } catch (err) {
      console.error('Authentication check failed:', err);
      localStorage.removeItem('token');
      delete api.defaults.headers.common['Authorization'];
    } finally {
      setLoading(false);
    }
  };

  checkAuth();
}, []);

// Функція для входу
const login = async (credentials) => {
  try {
    setError(null);
    const response = await api.post('/auth/login', credentials);
    const { token, user } = response.data;

    localStorage.setItem('token', token);
    api.defaults.headers.common['Authorization'] = `Bearer ${token}`;

    setUser(user);
    return user;
  } catch (err) {
    setError(err.response?.data?.message || 'Помилка входу');
    throw err;
  }
};

// Функція для реєстрації
const register = async (userData) => {
  try {
    setError(null);
    const response = await api.post('/auth/register', userData);
    const { token, user } = response.data;
```

```
    localStorage.setItem('token', token);
    api.defaults.headers.common['Authorization'] = `Bearer ${token}`;

    setUser(user);
    return user;
  } catch (err) {
    setError(err.response?.data?.message || 'Помилка реєстрації');
    throw err;
  }
};

// Функція для виходу
const logout = () => {
  localStorage.removeItem('token');
  delete api.defaults.headers.common['Authorization'];
  setUser(null);
};

// Функція оновлення інформації користувача
const updateUserProfile = async (userData) => {
  try {
    setError(null);
    const response = await api.put('/users/profile', userData);
    setUser(response.data);
    return response.data;
  } catch (err) {
    setError(err.response?.data?.message || 'Помилка оновлення профілю');
    throw err;
  }
};

// Функція для зміни користувача на продавця
const becomeSeller = async () => {
  try {
    setError(null);
    const response = await api.post('/users/become-seller');
    setUser(response.data);
```

```
      return response.data;
    } catch (err) {
      setError(err.response?.data?.message || 'Помилка створення облікового запису продавця');
      throw err;
    }
  };

  const value = {
    user,
    loading,
    error,
    login,
    register,
    logout,
    updateUserProfile,
    becomeSeller,
    isAuthenticated: !!user,
    isSeller: user?.role === 'seller' || user?.role === 'admin'
  };

  return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>;
};

import React, { createContext, useState, useEffect } from 'react';

export const CartContext = createContext();

export const CartProvider = ({ children }) => {
  const [cart, setCart] = useState([]);
  const [totalItems, setTotalItems] = useState(0);
  const [totalPrice, setTotalPrice] = useState(0);

  // Завантаження кошика з localStorage при ініціалізації
  useEffect(() => {
    const savedCart = localStorage.getItem('cart');
    if (savedCart) {
      try {
        const parsedCart = JSON.parse(savedCart);
```

```
      setCart(parsedCart);
    } catch (error) {
      console.error('Помилка при парсингу кошика з localStorage:', error);
      localStorage.removeItem('cart');
    }
  }
}, []);

// Оновлення localStorage при зміні кошика
useEffect(() => {
  localStorage.setItem('cart', JSON.stringify(cart));

  // Перерахунок загальної кількості та ціни
  const itemCount = cart.reduce((total, item) => total + item.quantity, 0);
  const price = cart.reduce((total, item) => total + (item.price * item.quantity), 0);

  setTotalItems(itemCount);
  setTotalPrice(price);
}, [cart]);

// Додавання товару до кошика
const addToCart = (product, quantity = 1) => {
  setCart(prevCart => {
    const existingItemIndex = prevCart.findIndex(item => item.id === product.id);

    if (existingItemIndex !== -1) {
      // Якщо товар вже є в кошику, оновлюємо кількість
      const updatedCart = [...prevCart];
      updatedCart[existingItemIndex] = {
        ...updatedCart[existingItemIndex],
        quantity: updatedCart[existingItemIndex].quantity + quantity
      };
      return updatedCart;
    } else {
      // Якщо товару немає в кошику, додаємо його
      return [...prevCart, { ...product, quantity }];
    }
  });
```

```
  };

  // Оновлення кількості товару в кошику
  const updateQuantity = (productId, quantity) => {
    if (quantity <= 0) {
      removeFromCart(productId);
      return;
    }

    setCart(prevCart =>
      prevCart.map(item =>
        item.id === productId ? { ...item, quantity } : item
      )
    );
  };

  // Видалення товару з кошика
  const removeFromCart = (productId) => {
    setCart(prevCart => prevCart.filter(item => item.id !== productId));
  };

  // Очищення кошика
  const clearCart = () => {
    setCart([]);
  };

  const value = {
    cart,
    totalItems,
    totalPrice,
    addToCart,
    updateQuantity,
    removeFromCart,
    clearCart
  };

  return <CartContext.Provider value={value}>{children}</CartContext.Provider>;
};
```

```
import axios from 'axios';

const api = axios.create({
  baseURL: process.env.REACT_APP_API_URL || 'http://localhost:5000/api',
  headers: {
    'Content-Type': 'application/json',
  },
});

// Додавання перехоплювача для автоматичного додавання токену авторизації
api.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem('token');
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => Promise.reject(error)
);

// Перехоплювач для обробки помилок
api.interceptors.response.use(
  (response) => response,
  (error) => {
    // Обробка відключення сесії користувача
    if (error.response && (error.response.status === 401 || error.response.status === 403)) {
      localStorage.removeItem('token');
    }
    return Promise.reject(error);
  }
);

export default api;

import React, { useContext, useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
```

```
import { AuthContext } from '../../context/AuthContext';
import { CartContext } from '../../context/CartContext';
import SearchBar from '../common/SearchBar';
import logo from '../../assets/logo.svg';
import '../../styles/Header.css';

const Header = () => {
  const { user, isAuthenticated, isSeller, logout } = useContext(AuthContext);
  const { totalItems } = useContext(CartContext);
  const [isMenuOpen, setIsMenuOpen] = useState(false);
  const navigate = useNavigate();

  const toggleMenu = () => {
    setIsMenuOpen(!isMenuOpen);
  };

  const handleLogout = () => {
    logout();
    navigate('/');
  };

  const categories = [
    { id: 1, name: 'Процесори' },
    { id: 2, name: 'Відеокарти' },
    { id: 3, name: 'Материнські плати' },
    { id: 4, name: 'Оперативна пам\'ять' },
    { id: 5, name: 'Накопичувачі' },
    { id: 6, name: 'Блоки живлення' },
    { id: 7, name: 'Корпуси' },
    { id: 8, name: 'Охолодження' }
  ];

  return (
    <header className="header">
      <div className="header-top">
        <div className="logo">
          <Link to="/">
            <img src={logo} alt="TechMarket Logo" />
```

```
    <span>TechMarket</span>
  </Link>
</div>

<SearchBar />

<div className="header-actions">
  <Link to="/cart" className="cart-icon">
    <i className="fas fa-shopping-cart"></i>
    {totalItems > 0 && <span className="cart-count">{totalItems}</span>}
  </Link>

  {isAuthenticated ? (
    <div className="user-menu">
      <button className="user-button" onClick={toggleMenu}>
        <i className="fas fa-user"></i>
        <span>{user.name}</span>
      </button>

      {isMenuOpen && (
        <div className="dropdown-menu">
          <Link to="/profile">Мій профіль</Link>
          <Link to="/orders">Мої замовлення</Link>
          {isSeller ? (
            <Link to="/seller/dashboard">Панель продавця</Link>
          ) : (
            <Link to="/profile?tab=become-seller">Стати продавцем</Link>
          )}
          <button onClick={handleLogout}>Вийти</button>
        </div>
      )}
    </div>
  ) : (
    <div className="auth-buttons">
      <Link to="/login" className="login-btn">Увійти</Link>
      <Link to="/register" className="register-btn">Зареєструватися</Link>
    </div>
  )}
```

```
          </div>
        </div>

        <nav className="main-nav">
          <ul className="categories-menu">
            {categories.map(category => (
              <li key={category.id}>
                <Link to={`/category/${category.id}`}>{category.name}</Link>
              </li>
            ))}
          </ul>
        </nav>
      </header>
    );
};

export default Header;

import React from 'react';
import { Link } from 'react-router-dom';
import '../../styles/Footer.css';

const Footer = () => {
  return (
    <footer className="footer">
      <div className="footer-content">
        <div className="footer-section">
          <h3>TechMarket</h3>
          <p>Ваш надійний магазин комп'ютерних комплектуючих. Знайдіть все необхідне для вашого ПК в одному місці.</p>
        </div>

        <div className="footer-section">
          <h3>Категорії</h3>
          <ul>
            <li><Link to="/category/1">Процесори</Link></li>
            <li><Link to="/category/2">Відеокарти</Link></li>
            <li><Link to="/category/3">Материнські плати</Link></li>
```

```
            <li><Link to="/category/4">Оперативна пам'ять</Link></li>
            <li><Link to="/category/5">Накопичувачі</Link></li>
          </ul>
        </div>

        <div className="footer-section">
          <h3>Інформація</h3>
          <ul>
            <li><Link to="/about">Про нас</Link></li>
            <li><Link to="/delivery">Доставка і оплата</Link></li>
            <li><Link to="/returns">Повернення товару</Link></li>
            <li><Link to="/terms">Умови використання</Link></li>
            <li><Link to="/privacy">Політика конфіденційності</Link></li>
          </ul>
        </div>

        <div className="footer-section">
          <h3>Контакти</h3>
          <ul className="contact-info">
            <li><i className="fas fa-map-marker-alt"></i> м. Київ, вул. Технічна, 10</li>
            <li><i className="fas fa-phone"></i> +380 (44) 123-4567</li>
            <li><i className="fas fa-envelope"></i> support@techmarket.ua</li>
          </ul>

          <div className="social-media">
            <a href="https://facebook.com" target="_blank" rel="noreferrer"><i className="fab fa-facebook-f"></i></a>
            <a href="https://instagram.com" target="_blank" rel="noreferrer"><i className="fab fa-instagram"></i></a>
            <a href="https://twitter.com" target="_blank" rel="noreferrer"><i className="fab fa-twitter"></i></a>
            <a href="https://youtube.com" target="_blank" rel="noreferrer"><i className="fab fa-youtube"></i></a>
          </div>
        </div>
      </div>

      <div className="footer-bottom">
```

```
      <p>&copy; {new Date().getFullYear()} TechMarket. Всі права захищені.</p>
    </div>
  </footer>
 );
};

export default Footer;

import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import '../../styles/SearchBar.css';

const SearchBar = () => {
 const [searchTerm, setSearchTerm] = useState('');
 const navigate = useNavigate();

 const handleSubmit = (e) => {
  e.preventDefault();
  if (searchTerm.trim()) {
   navigate(`/search?q=${encodeURIComponent(searchTerm.trim())}`);
  }
 };

 return (
  <div className="search-bar">
   <form onSubmit={handleSubmit}>
    <input
     type="text"
     placeholder="Пошук комплектуючих..."
     value={searchTerm}
     onChange={(e) => setSearchTerm(e.target.value)}
    />
    <button type="submit">
     <i className="fas fa-search"></i>
    </button>
   </form>
  </div>
 );
```

```
};

export default SearchBar;

import React, { useContext } from 'react';
import { Link } from 'react-router-dom';
import { CartContext } from '../../context/CartContext';
import '../../styles/ProductCard.css';

const ProductCard = ({ product }) => {
 const { addToCart } = useContext(CartContext);

 const handleAddToCart = (e) => {
  e.preventDefault();
  e.stopPropagation();
  addToCart(product, 1);
 };

 return (
  <div className="product-card">
   <Link to={`/product/${product.id}`} className="product-link">
    <div className="product-image">
     <img src={product.imageUrl} alt={product.name} />
     {product.discount > 0 && (
      <span className="product-discount">-{product.discount}%</span>
     )}
    </div>

    <div className="product-info">
     <h3 className="product-name">{product.name}</h3>
     <div className="product-rating">
      <div className="stars">
       {[...Array(5)].map((_, i) => (
        <i key={i} className={`fas fa-star ${i < Math.round(product.rating) ? 'filled' : ''}`}></i>
       ))}
      </div>
      <span className="rating-count">({product.reviewCount})</span>
     </div>
```

```
            <div className="product-price">
              {product.discount > 0 ? (
                <>
                  <span    className="current-price">{(product.price    *    (1    -    product.discount    /
100)).toFixed(2)} ₴</span>
                  <span className="old-price">{product.price.toFixed(2)} ₴</span>
                </>
              ) : (
                <span className="current-price">{product.price.toFixed(2)} ₴</span>
              )}
            </div>

            <div className="product-availability">
              {product.stock > 0 ? (
                <span className="in-stock">В наявності</span>
              ) : (
                <span className="out-of-stock">Немає в наявності</span>
              )}
            </div>
          </div>
        </Link>

        <button
          className={`add-to-cart-btn ${product.stock <= 0 ? 'disabled' : ''}`}
          onClick={handleAddToCart}
          disabled={product.stock <= 0}
        >
          <i className="fas fa-shopping-cart"></i>
          Додати до кошика
        </button>
      </div>
    );
  };
```

Додаток Г – ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ІНТЕРАКТИВНОГО ВЕБДОДАТКУ КОНФІГУРАТОРА ПЕРСОНАЛЬНОГО КОМП’ЮТЕРА

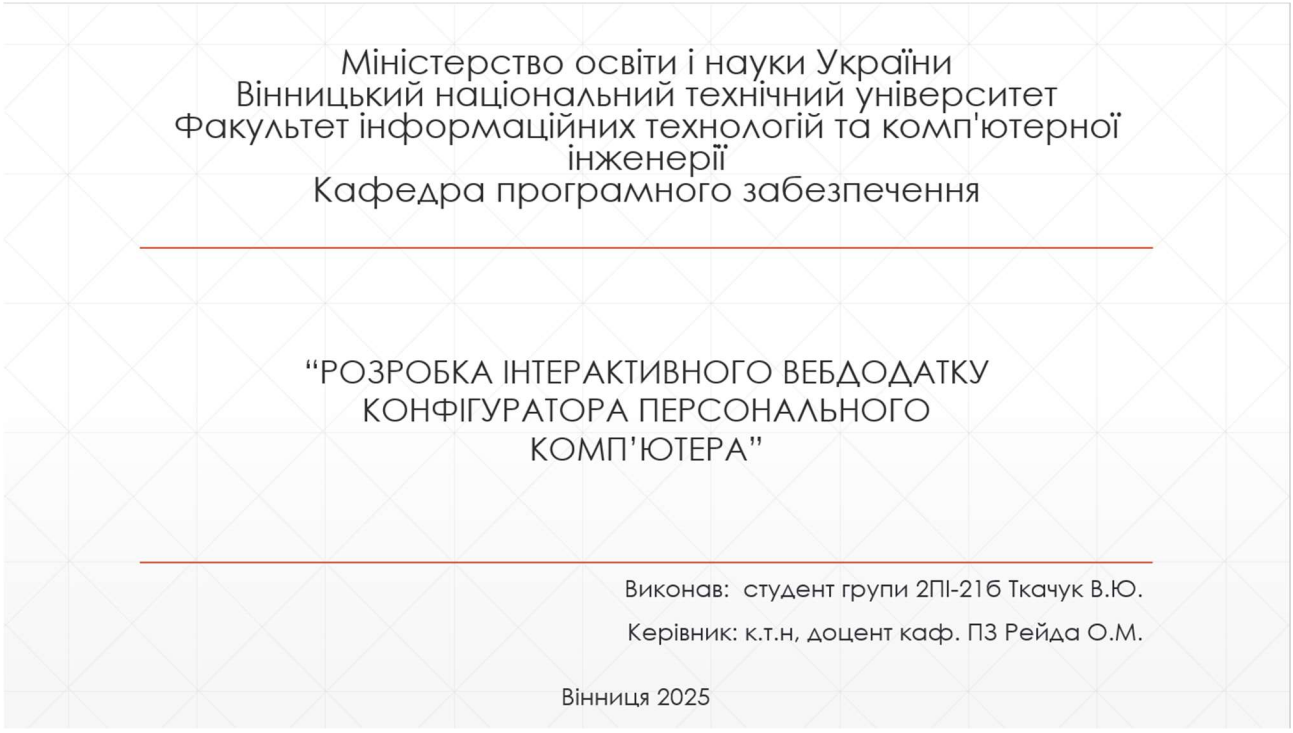

Рисунок Г.1 – Титульний слайд

Рисунок Г.2 – Актуальність

Рисунок Г.3 – Мета роботи, задачі дослідження, об'єкт та предмет дослідження

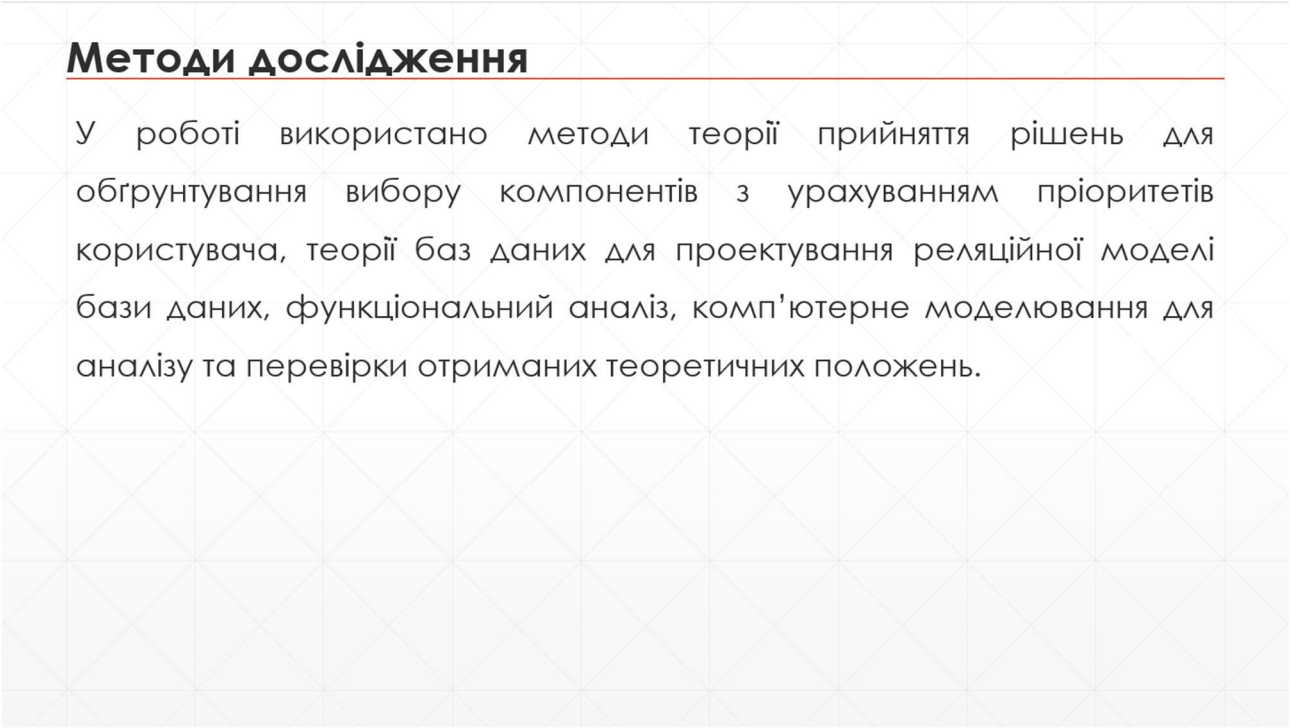

Рисунок Г.4 – Методи дослідження

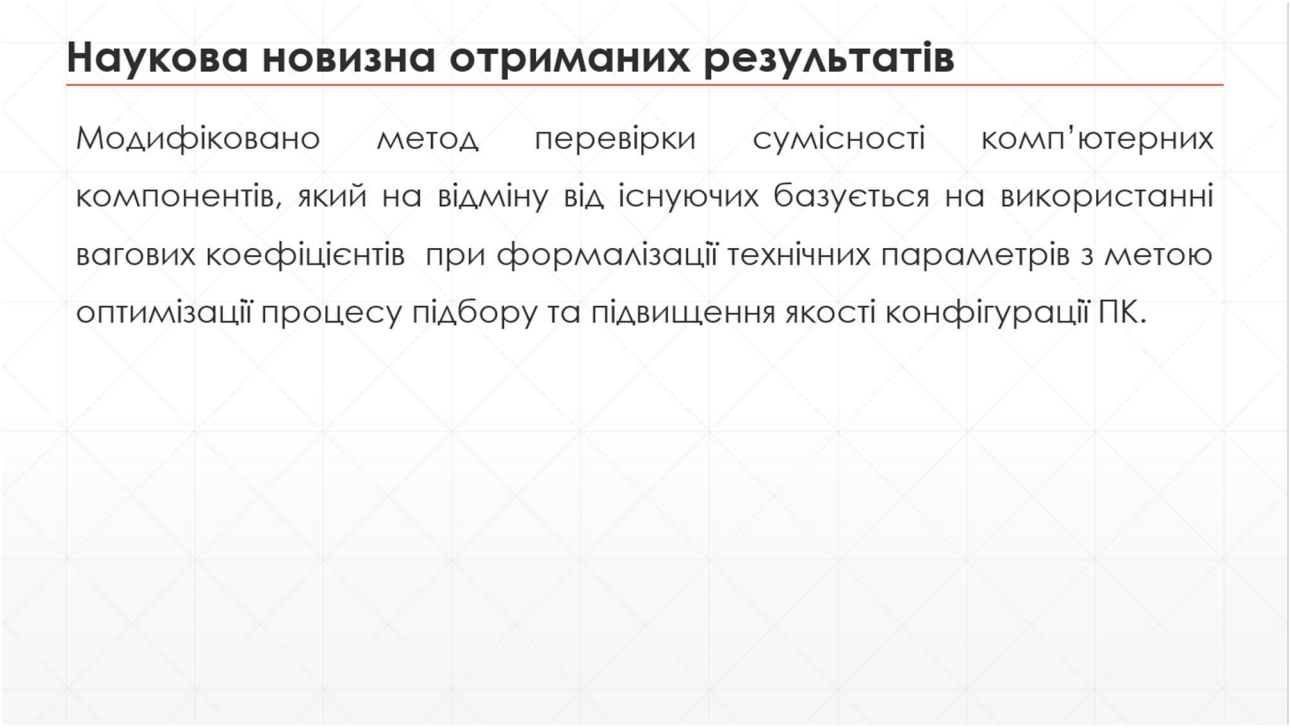

Рисунок Г.5 – Наукова новизна отриманих результатів

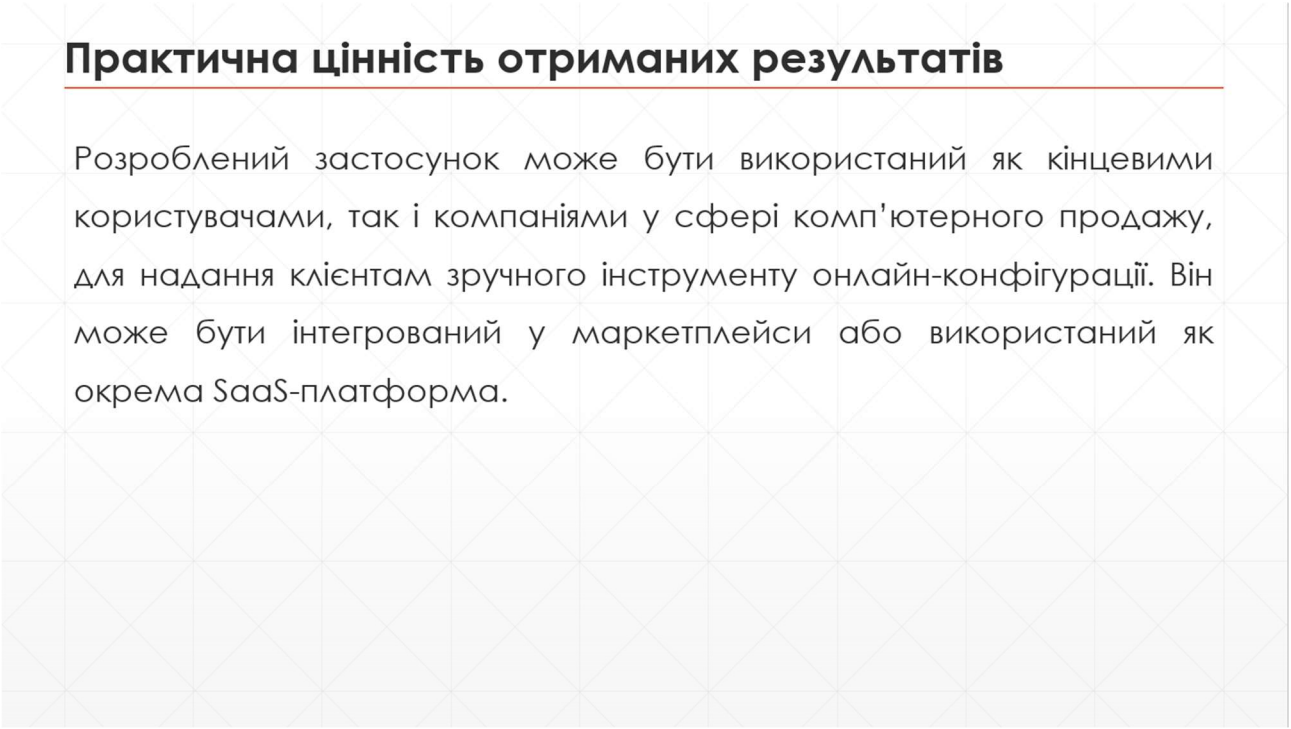

Рисунок Г.6 – Практична цінність отриманих результатів

Рисунок Г.7 – Аналіз відомих рішень

Рисунок Г.8 – Методи та засоби розробки

Рисунок Г.9 – Архітектура застосунку

Рисунок Г.10 – Алгоритми роботи застосунку

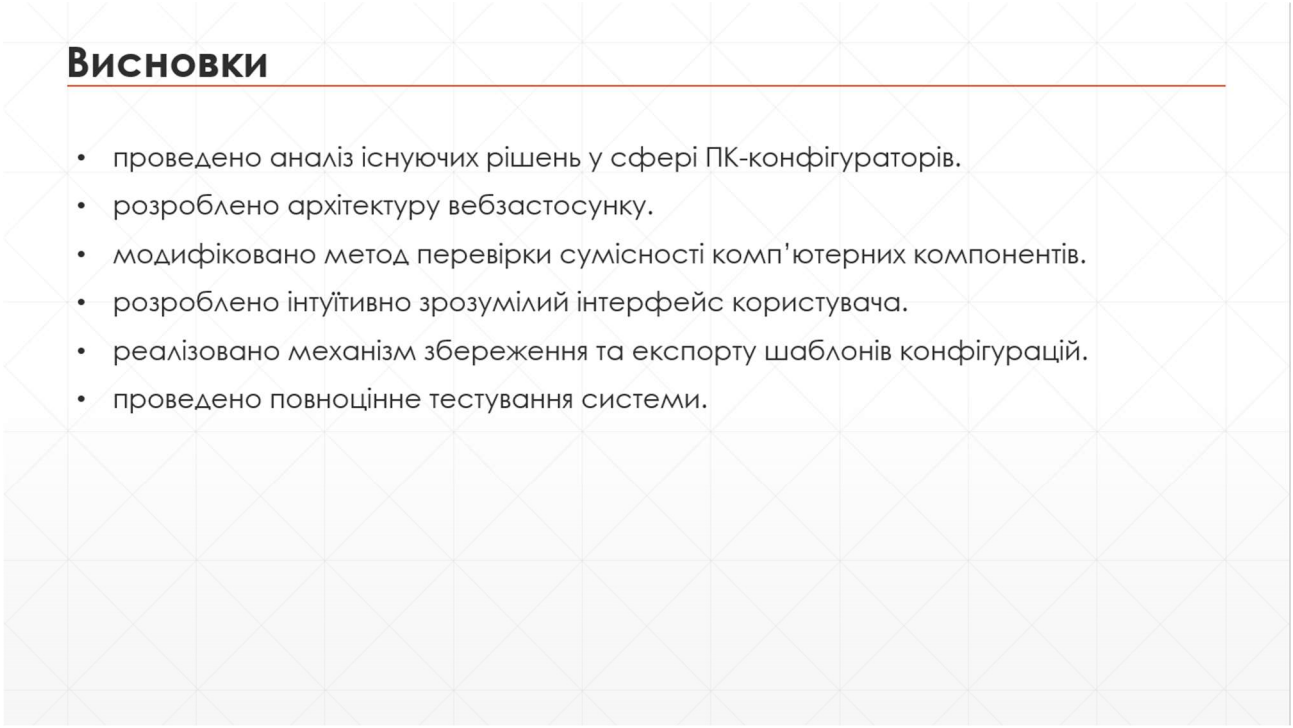

Рисунок Г.11 – Висновки

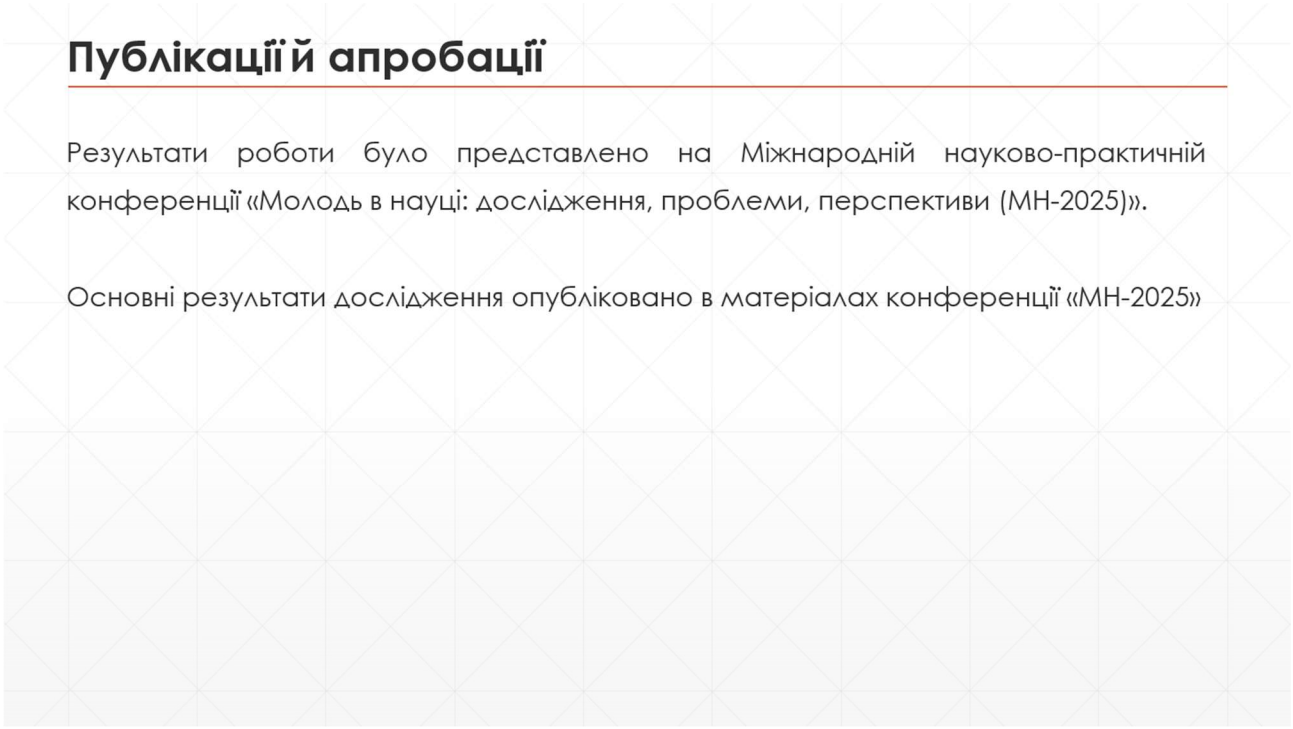

Рисунок Г.12 – Публікації й апробації

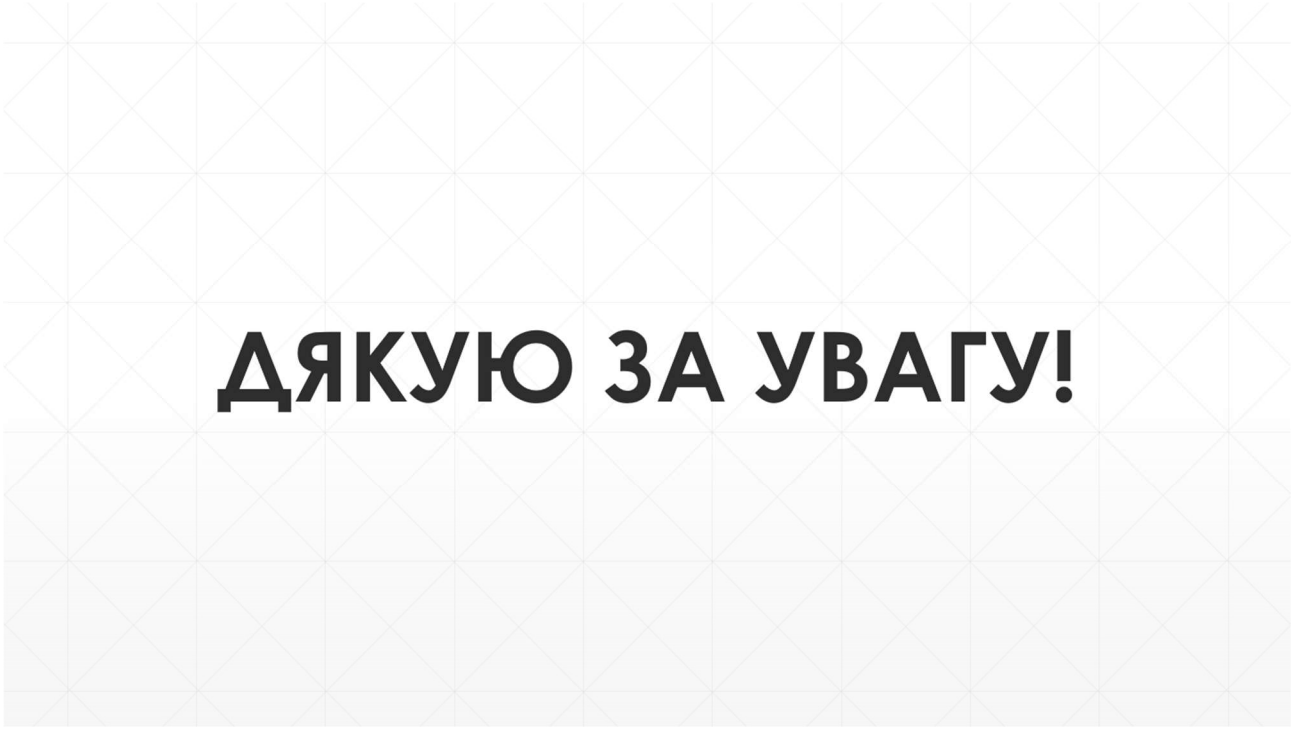

Рисунок Г.13 – Завершальний слайд