

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ВЕБСАЙТ ДЛЯ ОНЛАЙН-ПРОДАЖУ КОМП'ЮТЕРНИХ
КОМПЛЕКТУЮЧИХ
WEBSITE FOR ONLINE SALES OF COMPUTER
COMPONENTS

Виконав(ла): студент(ка) 4 курсу, групи КНТ-221
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерні науки

АПАНАСЕНКО І.П.

(ПРИЗВИЩЕ та ініціали)

Керівник ФЕДОРОНЧАК Т.В.

(ПРИЗВИЩЕ та ініціали)

Рецензент КОРОТКИЙ О.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

АПАНАСЕНКА Ігоря Павловича
(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Вебсайт для онлайн-продажу комп'ютерних комплектуючих .
Website for Online Sales of Computer Components

керівник проєкту (роботи) к.т.н., доцент, ФЕДОРОНЧАК Тетяна Василівна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “28”квітня 2025 року №209

2. Строк подання студентом проєкту (роботи) 2 червня 2025 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми 3. Основні рішення щодо реалізації компонентів системи. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	ФЕДОРОНЧАК Т.В., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання “8” квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

_____ Ігор АПАНАСЕНКО
(підпис) (Імя ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Тетяна ФЕДОРОНЧАК
(підпис) (Імя ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
116 с., 17 табл., 30 рис., 2 дод., 7 джерел.

ІНТЕРНЕТ-МАГАЗИН, КОМП'ЮТЕРНІ КОМПЛЕКТУЮЧІ, DJANGO,
PYTHON, WEB-РОЗРОБКА, E-COMMERCE.

Об'єкт дослідження – програмні засоби та вебтехнології для автоматизації процесу онлайн-торгівлі комп'ютерними комплектуючими.

Предмет дослідження – вебсистеми для онлайн-торгівлі комп'ютерними комплектуючими.

Мета роботи – створення системи онлайн-продажу комп'ютерних комплектуючих, що забезпечує зручний інтерфейс для користувачів та управління товарами для адміністраторів.

Матеріали, методи та технічні засоби: вебфреймворк Django, мова програмування Python, база даних PostgreSQL, HTML/CSS/JavaScript, Bootstrap для фронтенд-розробки, середовище розробки Visual Studio Code.

Результати. Розроблено вебсайт для онлайн-продажу комп'ютерних комплектуючих з модульною архітектурою, що включає систему автентифікації користувачів, каталог товарів з фільтрацією, кошик покупок, систему замовлень та адміністративну панель. Реалізовано REST API для динамічного оновлення даних без перезавантаження сторінки.

Висновки. Створено функціональний інтернет-магазин, який забезпечує зручний процес вибору та придбання комп'ютерних комплектуючих, а також управління товарами та замовленнями через адміністративну панель.

Галузь використання – онлайн-торгівля комп'ютерними комплектуючими, що може бути використана як окремими продавцями, так і компаніями, що спеціалізуються на продажі комп'ютерних компонентів.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 116 pages, 17 tables, 30 figures, 2 appendixes, 7 sources.

ONLINE STORE, COMPUTER COMPONENTS, DJANGO, PYTHON, WEB DEVELOPMENT, E-COMMERCE.

Object of research – software tools and web technologies to automate the process of online trading in computer components.

Subject of research – web-based systems for online trading in computer components.

Purpose – to create an online sales system for computer components that provides a user-friendly interface for users and product management for administrators.

Materials, methods and techniques: Django web framework, Python programming language, PostgreSQL database, HTML/CSS/JavaScript, Bootstrap for front-end development, Visual Studio Code development environment.

Results. A website for the online sale of computer components with a modular architecture, including a user authentication system, a product catalogue with filtering, a shopping cart, an ordering system, and an administrative panel. A REST API was implemented to dynamically update data without reloading the page.

Conclusions. A functional online store has been created that provides a convenient process for selecting and purchasing computer components, as well as managing goods and orders through the administrative panel.

The field of application is online trade in computer components, which can be used by both individual sellers and companies specialising in the sale of computer components.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ.....	9
1 Аналіз предметної області.....	10
1.1 Сучасні тенденції розвитку електронної комерції в сфері комп'ютерних комплектуючих.....	10
1.2 Аналіз існуючих рішень.....	10
1.2.1 Огляд compbest.com.ua	10
1.2.2 Огляд neon.ua	11
1.2.3 Огляд itplanet.zp.ua	12
1.2.4 Висновки за аналізом існуючих рішень.....	13
1.3 Порівняльна таблиця існуючих аналогів	13
1.4 Висновки за розділом 1	15
2 Розробка архітектури програми.....	17
2.1 Функціональні вимоги до системи.....	17
2.2 Вибір технологій та інструментальних засобів для реалізації системи	19
2.2.1 Вибір серверної частини.....	19
2.2.2 Вибір клієнтської частини.....	22
2.2.3 Вибір системи керування базами даних	26
2.3 Висновки за розділом 2	28
3 Основні рішення щодо реалізації компонентів системи.....	29
3.1 Опис структури та логіки бази даних	29
3.2 Функціонування програмного забезпечення.....	39
3.3 Опис структури та інтерфейсу вебсайту.....	41
3.3.1 Головна сторінка	41
3.3.2 Сторінка додавання/редагування товару	43
3.3.3 Сторінка перегляду товару.....	46
3.3.4 Сторінка адміністрування замовлень.....	47
3.3.5 Сторінка перегляду замовлення	49

3.3.6 Сторінка кошика.....	51
3.3.7 Сторінка оформлення замовлення.....	53
3.3.8 Сторінка профілю користувача	56
3.4 Висновки за розділом 3	57
4 Експлуатація, тестування та експериментальне дослідження програми	59
4.1 Опис застосування програми	59
4.2 Тестування та експериментальне дослідження	60
4.3 Висновки за розділом 4	66
Висновки	67
Перелік джерел посилання	69
Додаток А Технічне завдання	70
Додаток Б Текст програми	74
Додаток В Слайди презентації.....	108

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

СКБД	– Система Керування Базами Даних;
БД	– база даних;
CSS	– Cascading Style Sheets (каскадні таблиці стилів);
API	– Application Programming Interface (програмний інтерфейс застосування)
URL	– Uniform Resource Locator (уніфікований локатор ресурсу);
JSON	– JavaScript Object Notation (нотація об'єктів JavaScript);
UX	– User Experience (користувацький досвід);
UI	– User Interface (інтерфейс користувача).

ВСТУП

Сучасний ринок комп'ютерних комплектуючих характеризується високою динамікою розвитку та постійним зростанням попиту на високоякісні компоненти. З розвитком технологій та збільшенням кількості користувачів, які потребують потужних комп'ютерних систем, виникає нагальна потреба у створенні ефективних онлайн-платформ для продажу комп'ютерних комплектуючих.

Актуальність теми обумовлена наступними факторами:

- зростаюча популярність онлайн-шопінгу та необхідність адаптації традиційних магазинів до цифрового середовища;
- потреба у створенні зручного інтерфейсу для покупців комп'ютерних комплектуючих, що дозволить швидко знаходити потрібні компоненти та оформляти замовлення;
- необхідність автоматизації процесів управління товарами та замовленнями для продавців;
- розвиток електронної комерції та збільшення конкуренції на ринку онлайн-продажів комп'ютерних комплектуючих.

Мета роботи полягає у створенні системи онлайн-продажу комп'ютерних комплектуючих, що забезпечить зручний інтерфейс для користувачів та управління товарами для адміністраторів.

Для досягнення поставленої мети у роботі планується вирішити такі завдання:

- провести аналіз існуючих рішень та визначити оптимальні технології для розробки вебзастосунку;
- розробити архітектуру системи та спроектувати базу даних;
- реалізувати функціонал каталогу товарів з можливістю фільтрації та пошуку;
- створити систему управління кошиком та оформлення замовлень;
- провести комплексне тестування системи та перевірити її працездатність.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Сучасні тенденції розвитку електронної комерції в сфері комп'ютерних комплектуючих

Електронна комерція в сфері комп'ютерних комплектуючих переживає період активного розвитку та трансформації. Сучасні онлайн-магазини перетворилися з простих каталогів товарів на складні екосистеми, що об'єднують широкий спектр функціональних можливостей та сервісів.

Особливістю ринку комп'ютерних комплектуючих є його технічна складність та швидка зміна асортименту. Користувачі потребують детальної інформації про технічні характеристики товарів, їх сумісність та продуктивність. Це створює специфічні вимоги до структури онлайн-магазину, включаючи детальну категоризацію товарів, систему технічних специфікацій, динамічне оновлення інформації про наявність та ціни, а також можливість швидкого пошуку за технічними параметрами.

Сучасні технології дозволяють створювати інтерактивні вебзастосунки з адаптивними інтерфейсами, що забезпечують зручний доступ до магазину з різних пристроїв.

1.2 Аналіз існуючих рішень

Для аналізу сучасних підходів до розробки інтернет-магазинів комп'ютерних комплектуючих було розглянуто три українських онлайн-магазини: compbest.com.ua [1], itplanet.zp.ua [2], neon.ua [3]. Кожен із них має свої особливості реалізації функціоналу, дизайну та взаємодії з користувачем.

1.2.1 Огляд compbest.com.ua

Даний сайт вирізняється відносно сучасним дизайном, хоча й не повністю відповідає сучасним стандартам UI/UX. На головній сторінці (рис. 1.1) видно, що

мобільна версія реалізована лише частково: для її активації потрібно натиснути спеціальну кнопку, а за замовчуванням сайт відкривається у десктопній версії навіть на мобільних пристроях, що знижує зручність користування. Оформлення замовлення реалізовано через модальне вікно кошика, яке з'являється після додавання товару. Однак, відсутні сповіщення про успішне додавання товару, що може викликати у користувача сумніви щодо виконання дії. Сторінка з детальним описом товару має застарілий вигляд, інформація подана неструктуровано, що ускладнює ознайомлення з характеристиками.

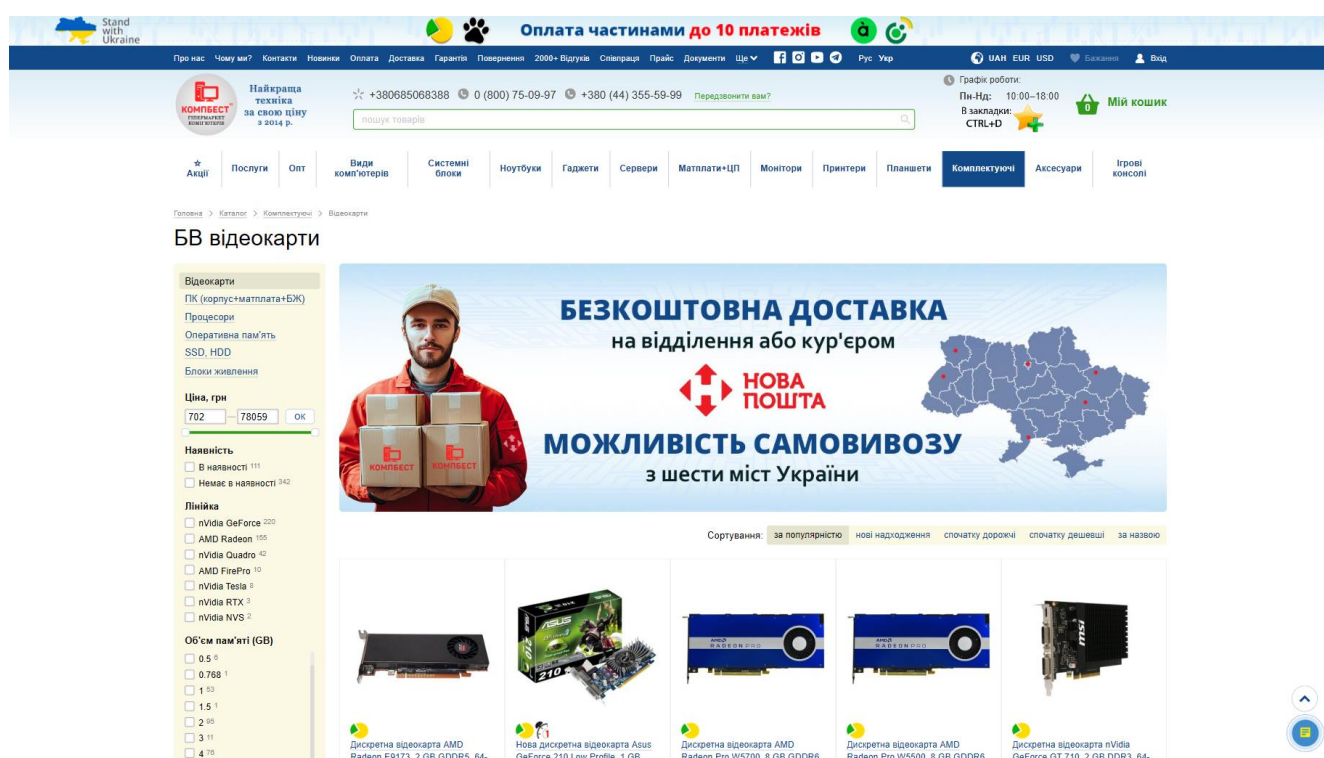


Рисунок 1.1 – Інтерфейс інтернет-магазину compbest

1.2.2 Огляд neop.ua

Як видно на рис. 1.2, сайт має застарілий дизайн та відсутність мобільної версії. Фільтрація товарів реалізована недостатньо зручно: неможливо обрати діапазон значень (наприклад, обрати відеокарти з об'ємом пам'яті від 2 до 4 ГБ), потрібно вручну відмічати кожне значення. Валідація при реєстрації відсутня, що може призвести до помилок при введенні даних. Сторінка з інформацією про товар

не є зручною для сприйняття: технічні характеристики подані у вигляді суцільного тексту, без структурування. Додавання товару до кошика не супроводжується жодним повідомленням, лише змінюється лічильник у кошику, що не завжди помітно для користувача.

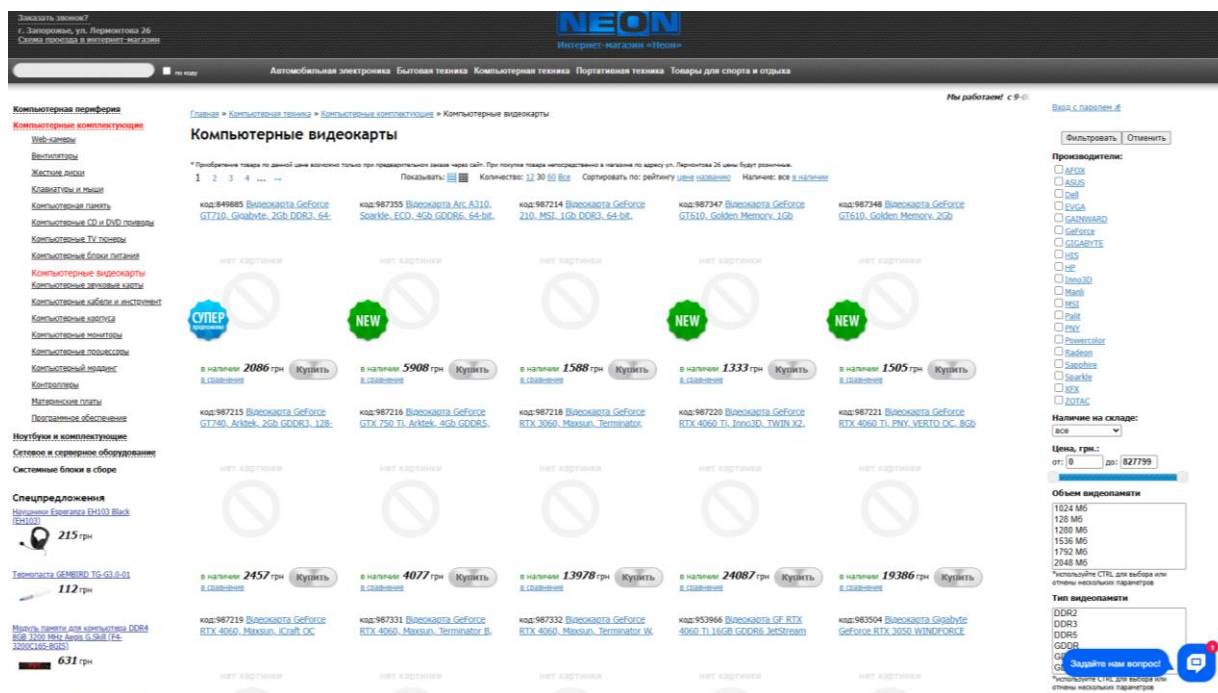


Рисунок 1.2 – Інтерфейс інтернет-магазину Неон

1.2.3 Огляд itplanet.zp.ua

На рис. 1.3 представлено інтерфейс сайту itplanet.zp.ua, який має дещо кращий дизайн у порівнянні з neon.ua, але все ще поступається сучасним рішенням. Мобільна версія відсутня, що робить користування з телефонів незручним. Функціонал фільтрації товарів не реалізований, що ускладнює пошук потрібних комплектуючих. Процес реєстрації містить валідацію, але вимагає введення великої кількості даних, що може відштовхнути потенційних клієнтів. Після реєстрації автоматична авторизація не відбувається, що також негативно впливає на користувацький досвід. Додавання товару до кошика одразу перенаправляє користувача на сторінку кошика, що не завжди зручно.



Рисунок 1.3 – Інтерфейс інтернет-магазину itPlanet

1.2.4 Висновки за аналізом існуючих рішень

Усі розглянуті рішення мають спільні недоліки: застарілий дизайн, відсутність повноцінної мобільної версії, незручна або недостатньо функціональна система фільтрації, а також недопрацьований процес взаємодії з кошиком та реєстрацією користувачів. Це свідчить про актуальність розробки нового інтернет-магазину, який би враховував сучасні вимоги до UI/UX, мав адаптивний дизайн, зручну систему фільтрації та сповіщення для користувачів, а також забезпечував простий і безпечний процес реєстрації та оформлення замовлень.

1.3 Порівняльна таблиця існуючих аналогів

Для більш наочного представлення результатів аналізу існуючих рішень було складено порівняльну таблицю (табл. 1.1), в якій відображено основні функціональні та якісні характеристики трьох популярних українських інтернет-магазинів комп'ютерних комплектуючих: compbest.com.ua, neon.ua, itplanet.zp.ua.

Також у таблиці наведено очікувані характеристики майбутнього власного вебсайту, який планується розробити в межах цієї роботи.

Таблиця 1.1 – Порівняльний аналіз функціональних можливостей існуючих інтернет-магазинів комп'ютерних комплектуючих

Критерій	compbest.com.ua	neon.ua	itplanet.zp.ua	Власний вебсайт
Адаптація для мобільних пристроїв	±	-	-	+
Сучасний дизайн	±	-	±	+
Зручний профіль користувача	±	-	-	+
Функціональна система фільтрації товарів	+	±	-	+
Структуроване відображення інформації про товар	+	-	+	+
Оптимізований процес роботи з кошиком та оформлення замовлення	±	-	-	+

У табл. 1.1 порівнюються такі ключові критерії, як адаптація для мобільних пристроїв, сучасність дизайну, інтуїтивність профілю користувача, функціональність системи фільтрації товарів, структурованість відображення інформації про товар, а також оптимізований процес роботи з кошиком та оформлення замовлення.

Оцінювання проводилося за принципом: "+" – реалізовано повністю, "±" – реалізовано частково або з недоліками, "-" – відсутнє або реалізовано незадовільно.

Проведений аналіз (див. табл. 1.1) показав, що жоден із розглянутих сайтів не відповідає повністю сучасним вимогам до електронної комерції: більшість із них мають застарілий дизайн, неадаптовані для мобільних пристроїв, мають обмежену або незручну систему фільтрації, а також недосконалий процес оформлення замовлення. Це підкреслює актуальність розробки нового вебсайту, який би враховував сучасні тенденції UI/UX, забезпечував би зручність користування, функціональність та безпеку для кінцевого користувача.

1.4 Висновки за розділом 1

У першому розділі було здійснено комплексний аналіз предметної області електронної комерції у сфері комп'ютерних комплектуючих. Визначено основні тенденції розвитку ринку, специфічні вимоги до функціоналу сучасних онлайн-магазинів, а також ключові проблеми, з якими стикаються користувачі під час взаємодії з існуючими платформами.

Проведений огляд та порівняльний аналіз трьох інтернет-магазинів показав, що більшість із них не відповідають сучасним стандартам зручності, адаптивності та функціональності. Основними недоліками є відсутність повноцінної мобільної версії, застарілий дизайн, обмежені можливості фільтрації товарів, а також неінтуїтивний процес оформлення замовлення. Це свідчить про наявність значного потенціалу для вдосконалення існуючих рішень та створення нового вебсайту, який би враховував сучасні вимоги до UI/UX, забезпечував би високий рівень зручності та функціональності для кінцевого користувача.

На основі проведеного аналізу сформульовано основні вимоги до майбутнього програмного продукту:

- адаптивний дизайн для різних типів пристроїв;
- сучасний, інтуїтивно зрозумілий інтерфейс;
- розширена система фільтрації та пошуку товарів;

- структуроване відображення інформації про продукцію;
- оптимізований процес роботи з кошиком та оформленням замовлення;
- забезпечення безпеки персональних даних користувачів.

Отже, розробка нового інтернет-магазину комп'ютерних комплектуючих є не лише доцільною, а й своєчасною відповіддю на сучасні виклики ринку. Реалізація такого проєкту дозволить підвищити якість онлайн-сервісу, задовольнити актуальні потреби користувачів та забезпечити конкурентоспроможність у сфері електронної комерції.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Функціональні вимоги до системи

Відповідно до проведеного аналізу предметної області, основною метою розроблюваної системи є створення сучасного вебзастосунку для онлайн-торгівлі комп'ютерними комплектуючими, який забезпечує зручний та безпечний процес купівлі-продажу для кінцевих користувачів і ефективне управління товарами для адміністраторів.

Функціональні вимоги до системи сформульовано з урахуванням потреб основних категорій користувачів: відвідувачів, зареєстрованих покупців та адміністраторів. Основні функціональні можливості системи включають:

- реєстрація та автентифікація користувачів: можливість створення облікового запису, вхід до системи;
- перегляд каталогу товарів: доступ до структурованого каталогу з категоріями, фільтрацією та пошуком за різними параметрами (виробник, ціна, характеристики тощо);
- перегляд детальної інформації про товар: відображення опису, технічних характеристик, фотографій, наявності на складі;
- робота з кошиком: додавання, видалення та редагування кількості товарів у кошику, перегляд загальної суми замовлення;
- оформлення замовлення: заповнення контактних даних, вибір способу доставки та оплати, підтвердження замовлення;
- управління замовленнями: перегляд історії замовлень для користувача, зміна статусу замовлення адміністратором, обробка нових замовлень.
- адміністративна панель: управління товарами (додавання, редагування, видалення), категоріями, користувачами;
- адаптивний інтерфейс: коректне відображення сайту на різних пристроях (ПК, планшет, смартфон).

Для наочного представлення функціональних вимог до системи доцільно використати діаграму прецедентів, яка відображає основні сценарії взаємодії користувачів із системою (рис. 2.1).



Рисунок 2.1 – Діаграма прецедентів системи

2.2 Вибір технологій та інструментальних засобів для реалізації системи

2.2.1 Вибір серверної частини

Серверна частина (бекенд) є фундаментом будь-якого сучасного вебзастосування, оскільки саме вона відповідає за обробку бізнес-логіки, взаємодію з базою даних, забезпечення безпеки, масштабованості та інтеграції з іншими сервісами. Від правильного вибору технології для серверної частини залежить не лише швидкість розробки, а й подальша підтримка, розширюваність та надійність системи.

На сучасному ринку існує низка популярних фреймворків для розробки серверної частини, серед яких найбільш поширеними є Django (Python), Express (Node.js), Ruby on Rails (Ruby), Laravel (PHP) та Spring (Java). Кожен із них має свої переваги та недоліки, які слід враховувати при виборі оптимального рішення для конкретного проєкту.

Для розробки онлайн-магазину комп'ютерних комплектуючих важливо, щоб серверна частина забезпечувала:

- високу швидкість розробки та легкість підтримки коду;
- вбудовані засоби безпеки;
- зручну роботу з базами даних та ORM;
- можливість швидкої інтеграції з REST API;
- масштабованість для подальшого розвитку проєкту;
- активну спільноту та якісну документацію.

З урахуванням цих критеріїв було проведено порівняльний аналіз найпопулярніших фреймворків, результати якого наведено у табл. 2.1.

Таблиця 2.1 – Порівняльний аналіз фреймворків для серверної частини

Критерій	Django	Node.js	Ruby on Rails	Laravel	Spring
Мова програмування	Python – проста, популярна, універсальна	JavaScript – універсальна, популярна	Ruby – менш поширена, але зручна	PHP – поширена, але менш сучасна	Java – складніша, але потужна
Швидкість розробки	Висока: багато готових модулів, генераторів, DRY-принцип	Висока: багато npm-пакетів, але часто треба писати більше вручну	Висока: генератори scaffold, «конвенція понад конфігурацією»	Висока: багато пакетів, але часто потребує налаштувань	Середня: багато шаблонного коду, складна конфігурація
Безпека	Вбудовані захисти від XSS, CSRF, SQLi, clickjacking	Потрібно додавати вручну (helmet, csurf тощо)	Вбудовані захисти, але менше оновлень	Є базові захисти, але не всі за замовчуванням	Високий рівень безпеки, але складна конфігурація
ORM	Вбудований потужний ORM, підтримка міграцій	Є ORM (Sequelize, TypeORM), але не вбудовані	Вбудований ActiveRecord, простий у використанні	Вбудований Eloquent ORM, зручний	Вбудований Hibernate, але складний
REST API	Django REST Framework – стандарт де-факто, гнучкий, зручний	Express дозволяє швидко створювати API, але багато ручної роботи	Є gems для API, але не так зручно	Є пакети для API, але не так гнучко	Spring Boot дозволяє створювати API, але складніше

Продовження таблиці 2.1

Критерій	Django	Node.js	Ruby on Rails	Laravel	Spring
Масштабованість	Добре масштабується, використовується у великих проєктах (Instagram, Pinterest)	Добре масштабується, підходить для мікросервісів	Підходить для середніх проєктів	Підходить для середніх проєктів	Висока масштабованість, підходить для великих enterprise-рішень
Документація	Дуже детальна, багато прикладів, велика спільнота	Багато прикладів, велика спільнота	Хороша документація, але менша спільнота	Хороша документація, велика спільнота	Дуже детальна, але складна
Легкість навчання	Дуже проста для початківців, багато навчальних матеріалів	Проста для тих, хто знає JS, але є нюанси з асинхронністю	Відносно проста, але Ruby менш популярна	Середня, PHP знайомий багатьом, але сучасний Laravel має свої особливості	Складна для початківців, вимагає знань Java та Spring-екосистеми
Підтримка тестування	Вбудовані засоби для тестування, зручна інтеграція	Є багато npm-пакетів для тестування	Вбудовані засоби для тестування	Вбудовані засоби для тестування	Вбудовані засоби для тестування

Проведений аналіз показав, що Django є оптимальним вибором для реалізації серверної частини онлайн-магазину комп'ютерних комплектуючих з наступних причин:

- висока швидкість розробки: завдяки великій кількості вбудованих інструментів (адмін-панель, ORM, автентифікація, захист) можна швидко створити повноцінний функціонал без зайвого дублювання коду;

- безпека: Django автоматично захищає від більшості поширених вебуразливостей, що особливо важливо для проєктів, які працюють з персональними даними та фінансовими операціями;
- гнучкість та масштабованість: фреймворк підходить як для невеликих стартапів, так і для великих високонавантажених систем;
- зручна робота з базами даних: потужний ORM дозволяє легко змінювати структуру даних, підтримує міграції, працює з різними СУБД;
- підтримка REST API: завдяки Django REST Framework можна швидко реалізувати сучасний API для взаємодії з фронтендом або мобільними додатками;
- велика спільнота та документація: це гарантує швидке вирішення проблем, наявність великої кількості навчальних матеріалів та постійну підтримку фреймворку[4];
- легкість навчання: Python – одна з найпопулярніших мов програмування, що робить Django доступним для широкого кола розробників.

Таким чином, вибір Django як серверної частини для розробки онлайн-магазину є обґрунтованим з точки зору швидкості розробки, безпеки, масштабованості та підтримки сучасних стандартів веброботки [5].

2.2.2 Вибір клієнтської частини

У сучасних вебзастосунках клієнтська частина (front-end) відіграє ключову роль у забезпеченні якісної взаємодії користувача із системою. Саме від неї залежить перше враження відвідувача, зручність навігації, швидкість доступу до інформації, а також загальний рівень задоволеності користувача. З розвитком технологій та зростанням конкуренції на ринку електронної комерції, вимоги до інтерфейсів вебсайтів постійно зростають: користувачі очікують не лише привабливого дизайну, а й інтуїтивної логіки, адаптивності до різних пристроїв, швидкої реакції на дії та інтерактивності.

Вибір технологій для клієнтської частини визначає не лише зовнішній вигляд і функціональність сайту, а й впливає на ефективність розробки, можливість

масштабування, простоту підтримки та подальшого розвитку проєкту. Для інтернет-магазину комп'ютерних комплектуючих особливо важливо забезпечити:

- адаптивний дизайн для коректного відображення на ПК, планшетах і смартфонах;
- високу продуктивність та швидке завантаження сторінок навіть при великій кількості товарів;
- зручну систему фільтрації, пошуку та сортування;
- можливість інтеграції з серверною частиною через REST API;
- можливість швидкого внесення змін та розширення функціоналу.

На сучасному ринку існує декілька основних підходів до розробки клієнтської частини:

- використання класичних технологій (HTML, CSS, JavaScript) у поєднанні з CSS-фреймворками (Bootstrap, Bulma, Foundation тощо);
- використання сучасних JavaScript-фреймворків (React, Angular, Vue.js, Svelte) для створення односторінкових додатків (SPA);
- комбіновані підходи, які дозволяють поєднувати переваги класичних і сучасних технологій.

Кожен із підходів має свої переваги та недоліки, які слід враховувати при виборі оптимального рішення для конкретного проєкту. Вибір клієнтської частини має базуватися на аналізі вимог до функціоналу, очікуваного навантаження, досвіду команди розробників, а також можливостей подальшого масштабування та підтримки.

Порівняльний аналіз основних технологій для розробки клієнтської частини наведено у табл. 2.2. Це дозволяє обґрунтовано зробити вибір на користь оптимального стека для реалізації інтернет-магазину комп'ютерних комплектуючих.

Таблиця 2.2 – Порівняльний аналіз фреймворків для клієнтської частини

Критерій	HTML+CSS+JS+Bootstrap(наш вибір)	React/Vue/Angular	Пояснення/Переваги для нашого проєкту
Швидкість розробки	Висока: наявність великої кількості готових компонентів, простота впровадження	Висока для складних SPA, але потребує додаткової конфігурації	Bootstrap дозволяє швидко реалізувати сучасний інтерфейс
Адаптивність	Забезпечується вбудованими можливостями Bootstrap, не потребує додаткових налаштувань	Потребує підключення окремих бібліотек для адаптивності	Сайт коректно відображається на всіх пристроях
Інтеграція з Django	Пряма інтеграція через шаблони Django, мінімальні складнощі при обміні даними	Потребує створення окремого API, налаштування CORS	Зменшується кількість коду, спрощується підтримка
Продуктивність	Висока для класичних сайтів, сторінки швидко завантажуються	Висока, але для SPA потрібна додаткова оптимізація	Користувачі отримують швидкий доступ до контенту
Легкість підтримки	Висока для невеликих і середніх проєктів, легко знайти розробників	Потрібні знання сучасних фреймворків та JavaScript	Спрощується залучення нових учасників до проєкту
Готові рішення	Bootstrap містить широкий набір компонентів: модальні вікна, форми, меню, повідомлення, пагінація	Для SPA потрібне підключення сторонніх бібліотек	Можна швидко реалізувати будь-який елемент інтерфейсу
Можливість розширення	Дозволяє поступово додавати JavaScript-компоненти за потреби	SPA вимагає повної архітектури з самого початку	Гнучкість: можна почати з простого рішення, поступово ускладнювати функціонал

Продовження таблиці 2.2

Критерій	HTML+CSS+JS+Bootstrap(наш вибір)	React/Vue/Angular	Пояснення/Переваги для нашого проєкту
Вартість розробки	Менше часу на розробку, менше помилок, нижча вартість	Більше часу на налаштування, тестування	Важливо для студентського або малого проєкту
Документація/Спільнота	Велика кількість матеріалів, Bootstrap є стандартом у галузі	Велика спільнота, але більше орієнтована на складні SPA	Легко знайти відповіді на питання, швидко вирішувати проблеми

Як видно з табл. 2.2, використання класичних технологій HTML, CSS, JavaScript у поєднанні з фреймворком Bootstrap є оптимальним вибором для реалізації інтернет-магазину комп'ютерних комплектуючих, що розробляється у межах цієї роботи.

Основними перевагами такого підходу є:

- висока швидкість розробки завдяки наявності великої кількості готових компонентів у Bootstrap, що дозволяє швидко створювати сучасний та адаптивний інтерфейс [6];
- адаптивність забезпечується вбудованими можливостями Bootstrap, що гарантує коректне відображення сайту на різних пристроях без додаткових зусиль [6];
- пряма інтеграція з Django через шаблони, що спрощує обмін даними між клієнтською та серверною частинами, зменшує кількість коду та полегшує підтримку проєкту;
- легкість підтримки та залучення нових розробників завдяки поширеності базових технологій та великій кількості навчальних матеріалів;
- гнучкість: у разі необхідності можна поступово додавати сучасні JavaScript-компоненти для розширення функціоналу без повної перебудови архітектури клієнтської частини;

– низька вартість розробки та мінімізація кількості помилок, що особливо важливо для студентських та малих проєктів.

Варто зазначити, що сучасні JavaScript-фреймворки (React, Vue, Angular) мають свої переваги, зокрема для створення складних односторінкових додатків із високим рівнем інтерактивності. Проте для даного проєкту, де основний акцент робиться на швидкість розробки, простоту підтримки та інтеграцію з серверною частиною на Django, класичний підхід із використанням Bootstrap є найбільш доцільним.

Таким чином, вибір HTML, CSS, JavaScript та Bootstrap як основних технологій для клієнтської частини повністю відповідає функціональним вимогам системи, забезпечує ефективну реалізацію архітектури програми та сприяє досягненню поставлених цілей проєкту.

2.2.3 Вибір системи керування базами даних

У сучасних інформаційних системах, особливо у сфері електронної комерції, система керування базами даних (СКБД) відіграє одну з найважливіших ролей. Саме СКБД забезпечує надійне зберігання, організацію, швидкий пошук, обробку та захист даних, які є основою для функціонування будь-якого вебзастосунку. Від правильного вибору СКБД залежить не лише продуктивність і масштабованість системи, а й можливість реалізації складних бізнес-процесів, підтримка транзакцій, забезпечення цілісності та безпеки даних.

У процесі розробки інтернет-магазину комп'ютерних комплектуючих особливу увагу слід приділити тому, як саме будуть зберігатися дані про товари, категорії, користувачів, замовлення, історію покупок тощо. Важливо, щоб обрана СКБД дозволяла ефективно працювати з великими обсягами структурованої інформації, підтримувала складні зв'язки між сутностями, забезпечувала швидке виконання аналітичних запитів, а також відповідала сучасним вимогам до безпеки та надійності.

На ринку представлено широкий спектр рішень для зберігання даних, які можна умовно поділити на дві великі групи: реляційні (SQL) та нереляційні (NoSQL) бази даних. Кожен із підходів має свої переваги та недоліки, і вибір між ними залежить від специфіки проєкту, структури даних, очікуваного навантаження та вимог до масштабованості.

Для обґрунтованого вибору СКБД для даного проєкту було проведено порівняльний аналіз основних характеристик реляційних та нереляційних баз даних, результати якого наведено у табл. 2.3.

Таблиця 2.3 – Порівняльний аналіз реляційних та нереляційних систем керування базами даних

Критерій	Реляційні БД (SQL)	Нереляційні БД (NoSQL)
Структура даних	Таблична, чітко визначені схеми	Гнучка, динамічна, документи / ключ-значення / графи
Мова запитів	Стандартизована (SQL)	Залежить від конкретної СКБД, часто власна
Цілісність даних	Висока, підтримка транзакцій (ACID)	Зазвичай нижча, часто eventual consistency
Масштабованість	Вертикальна (масштабування серверу)	Горизонтальна (розподілені системи)
Продуктивність	Висока для складних запитів, аналітики	Висока для великих обсягів неструктурованих даних
Гнучкість структури	Менш гнучка, зміни схеми потребують міграцій	Дуже гнучка, можна зберігати різні типи даних
Приклади	PostgreSQL, MySQL, Oracle, SQLite	MongoDB, Cassandra, Redis, CouchDB
Типові задачі	Фінанси, e-commerce, CRM, ERP	Big Data, IoT, соціальні мережі, кешування

Аналіз порівняльних характеристик реляційних та нереляційних баз даних свідчить, що реляційні СКБД найбільш ефективно вирішують задачі, пов'язані зі

зберіганням структурованої інформації, підтримкою складних зв'язків між сутностями та забезпеченням транзакційної цілісності даних. Саме такі вимоги характерні для інтернет-магазину комп'ютерних комплектуючих, де необхідно забезпечити цілісність даних про товари, замовлення, користувачів і взаємозв'язки між ними [7].

2.3 Висновки за розділом 2

У цьому розділі було визначено та обґрунтовано основні функціональні вимоги до системи, а також здійснено вибір оптимальних технологій і інструментальних засобів для її реалізації.

На основі аналізу предметної області сформульовано ключові сценарії взаємодії користувачів із системою, що відображено у вигляді діаграми прецедентів. Це дозволило чітко визначити перелік функцій, які повинна реалізовувати система для забезпечення зручності, безпеки та ефективності роботи як для кінцевих користувачів, так і для адміністраторів.

Проведений порівняльний аналіз сучасних фреймворків для серверної частини показав доцільність використання Django, що забезпечує високу швидкість розробки, безпеку, масштабованість і зручну інтеграцію з базою даних. Для клієнтської частини обрано класичний стек HTML, CSS, JavaScript у поєднанні з Bootstrap, що дозволяє створити адаптивний, сучасний та легко підтримуваний інтерфейс.

Щодо системи керування базами даних, було обґрунтовано вибір реляційної СКБД (PostgreSQL), яка найкраще відповідає вимогам до структурованості, цілісності та надійності зберігання даних у межах електронної комерції.

Таким чином, у розділі було закладено фундамент для подальшого проектування архітектури системи, визначено оптимальний технологічний стек і сформовано вимоги до основних компонентів програмного продукту, що забезпечить ефективну реалізацію поставлених цілей проєкту.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Опис структури та логіки бази даних

База даних інтернет-магазину побудована з використанням Django ORM та складається з 11 основних таблиць, які забезпечують повний функціонал для управління товарами, користувачами, кошиками та замовленнями. Кожна таблиця має своє призначення та взаємопов'язана з іншими через зовнішні ключі, що забезпечує цілісність даних та ефективне управління ними.

Архітектура бази даних побудована з урахуванням принципів нормалізації та забезпечує ефективне зберігання та обробку даних. Основні компоненти системи розділені на логічні блоки:

- блок управління товарами включає таблиці для зберігання інформації про категорії, виробників та самі товари. Особливістю цього блоку є гнучка система характеристик товарів, яка дозволяє налаштовувати специфікації для різних категорій товарів;
- блок управління користувачами складається з таблиць для зберігання основних даних користувачів та їх профілів. Це забезпечує безпечне зберігання конфіденційної інформації та розділення даних автентифікації та профілю;
- блок обробки замовлень включає таблиці для роботи з кошиками та замовленнями. Особливістю цього блоку є збереження історичних даних про ціни товарів на момент створення замовлення.

Взаємозв'язки між таблицями реалізовані через зовнішні ключі, що забезпечує цілісність даних та дозволяє ефективно виконувати запити для отримання пов'язаної інформації. Наприклад, для отримання повної інформації про товар система може об'єднати дані з таблиць товарів, категорій та виробників.

Для кращого розуміння взаємозв'язків між таблицями бази даних, на рис. 3.1 представлена ER-діаграма, яка наочно демонструє структуру бази даних та зв'язки між її компонентами. На діаграмі відображені всі описані вище таблиці та їх взаємозв'язки, що дозволяє краще зрозуміти архітектуру бази даних інтернет-магазину.

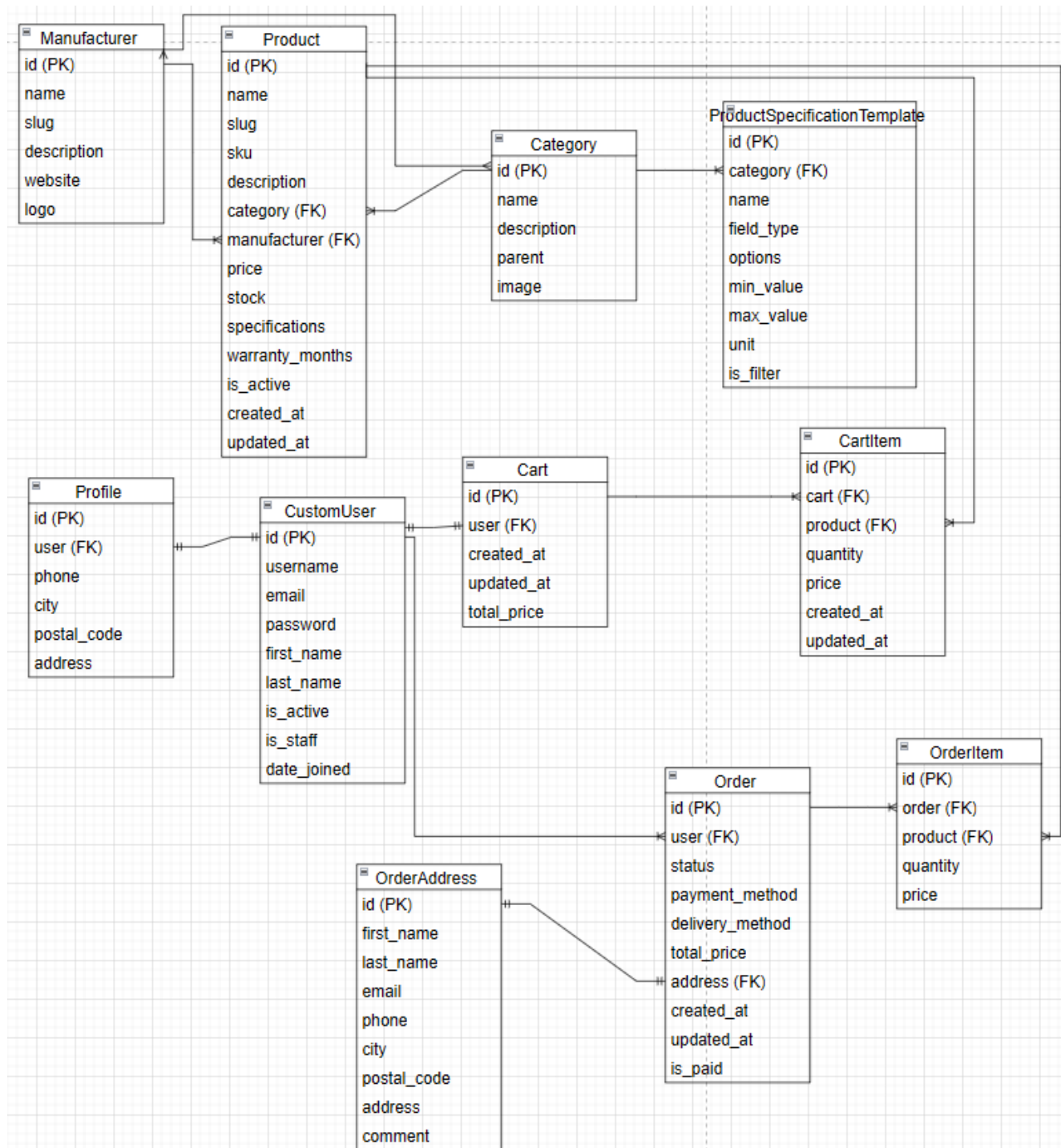


Рисунок 3.1 – Схема бази даних інтернет-магазину

Таблиця Категорії (табл. 3.1) призначена для зберігання інформації про категорії товарів. Вона дозволяє організувати товари в ієрархічну структуру завдяки полю `parent`, яке вказує на батьківську категорію. Кожна категорія має унікальний `slug` для URL-адреси та може містити опис та зображення.

Таблиця 3.1 – Структура таблиці "Категорії" (Category)

Назва атрибуту	Тип	Пояснення
id	Integer	Унікальний ідентифікатор категорії в системі
name	CharField	Назва категорії товарів
slug	SlugField	URL-адреса категорії
description	TextField	Опис категорії
parent	ForeignKey	Батьківська категорія
image	ImageField	Зображення категорії

Таблиця Виробники (табл. 3.2) зберігає інформацію про виробників товарів. Кожен виробник має назву, опис, вебсайт та логотип. Поле slug використовується для створення зручних URL-адрес для сторінок виробників.

Таблиця 3.2 – Структура таблиці "Виробники" (Manufacturer)

Назва атрибуту	Тип	Пояснення
id	Integer	Унікальний ідентифікатор виробника в системі
name	CharField	Назва виробника
slug	SlugField	URL-адреса виробника
description	TextField	Опис виробника
website	URLField	Вебсайт виробника
logo	ImageField	Логотип виробника

Таблиця Товари (табл. 3.3), містить інформацію про всі товари в магазині. Кожен товар має унікальний артикул (SKU), прив'язаний до категорії та виробника,

має ціну, кількість на складі та технічні характеристики у форматі JSON. Таблиця також відстежує статус активності товару та дати створення/оновлення.

Таблиця 3.3 – Структура таблиці "Товари" (Product)

Назва атрибуту	Тип	Пояснення
id	Integer	Унікальний ідентифікатор товару в системі
name	CharField	Назва товару
slug	SlugField	URL-адреса товару, створюється автоматично з назви
sku	CharField	Унікальний артикул товару, генерується автоматично за шаблоном
description	TextField	Опис товару
category	ForeignKey	Зв'язок з категорією товару для класифікації
manufacturer	ForeignKey	Зв'язок з виробником товару
price	DecimalField	Ціна товару
stock	PositiveInteger	Кількість одиниць товару на складі
specifications	JSONField	Технічні характеристики товару у форматі JSON
warranty_months	PositiveInteger	Гарантійний термін
is_active	Boolean	Прапорець активності товару (чи доступний для продажу)
created_at	DateTimeField	Дата та час додавання товару до каталогу
updated_at	DateTimeField	Дата та час останнього оновлення інформації про товар

Таблиця Шаблони характеристик (табл. 3.4) призначена для визначення структури технічних характеристик товарів для кожної категорії. Вона дозволяє створювати уніфіковані шаблони характеристик, які будуть застосовуватися до всіх товарів у конкретній категорії. Це забезпечує стандартизацію опису товарів та полегшує пошук та фільтрацію товарів за технічними характеристиками. Кожен шаблон може мати різний тип поля (текстове поле, вибір зі списку або діапазон), що дозволяє гнучко налаштовувати формат введення характеристик. Додатково, характеристики можуть бути позначені як фільтри, що дозволяє використовувати їх для пошуку та фільтрації товарів на сайті. Ця таблиця тісно пов'язана з таблицею "Товари" через поле specifications, яке зберігає значення характеристик у форматі JSON відповідно до шаблону категорії товару.

Таблиця 3.4 – Структура таблиці "Шаблони характеристик" (ProductSpecificationTemplate)

Назва атрибуту	Тип	Пояснення
id	Integer	Унікальний ідентифікатор шаблону в системі
category	ForeignKey	Зв'язок з категорією товарів
name	CharField	Назва характеристики
field_type	CharField	Тип поля (текстове поле, вибір зі списку, діапазон)
options	JSONField	Варіанти вибору для полів типу "вибір зі списку"
min_value	FloatField	Мінімальне значення для полів типу "діапазон"

Продовження таблиці 3.4

Назва атрибуту	Тип	Пояснення
max_value	FloatField	Максимальне значення для полів типу "діапазон"
unit	CharField	Одиниця виміру характеристики
is_filter	Boolean	Прапорець використання характеристики як фільтра

Таблиця Користувачі (табл. 3.5) використовується для зберігання інформації про користувачів системи. Розширює стандартну модель користувача Django, додаючи додаткові поля для імені та прізвища. Відповідає за автентифікацію та авторизацію користувачів.

Таблиця 3.5 – Структура таблиці "Користувачі" (CustomUser)

Назва атрибуту	Тип	Пояснення
id	Integer	Унікальний ідентифікатор користувача в системі
username	CharField	Логін користувача
email	EmailField	Email користувача
password	CharField	Пароль користувача
first_name	CharField	Ім'я користувача
last_name	CharField	Прізвище користувача
is_active	Boolean	Статус активності
is_staff	Boolean	Право доступу до адміністративної панелі
date_joined	DateTimeField	Дата реєстрації

Допоміжна таблиця Профілі (табл. 3.6), пов'язана з CustomUser через OneToOneField. Зберігає додаткову інформацію про користувача, таку як контактні дані та адресу доставки. Це дозволяє розділити основну інформацію для автентифікації та додаткові дані профілю.

Таблиця 3.6 – Структура таблиці "Профілі" (Profile)

Назва атрибуту	Тип	Пояснення
id	Integer	Унікальний ідентифікатор профілю в системі
user	OneToOneField	Зв'язок з основним обліковим записом користувач
phone	CharField	Телефон користувача
city	CharField	Місто проживання
postal_code	CharField	Поштовий індекс
address	TextField	Адреса доставки

Таблиця Кошки (табл. 3.7) використовується для зберігання кошиків користувачів. Кожен користувач має один кошик, який автоматично створюється при реєстрації. Відстежує загальну вартість товарів у кошику та дати створення/оновлення.

Таблиця 3.7 – Структура таблиці "Кошки" (Cart)

Назва атрибуту	Тип	Пояснення
id	Integer	Унікальний ідентифікатор кошика в системі
user	OneToOneField	Зв'язок з користувачем, який володіє кошиком

Продовження таблиці 3.7

Назва атрибуту	Тип	Пояснення
created_at	DateTimeField	Дата створення кошика
updated_at	DateTimeField	Дата останнього оновлення кошика
total_price	DecimalField	Загальна вартість усіх товарів у кошику

Таблиця Товари в кошику (табл. 3.8) призначена для зберігання товарів, доданих до кошика. Кожен запис містить посилання на кошик та товар, кількість одиниць товару та ціну на момент додавання до кошика. Забезпечує відсутність дублювання товарів у кошику користувача. Тобто, один і той самий товар не може бути доданий до кошика кілька разів як окремі записи - замість цього збільшується кількість існуючого товару в кошику.

Таблиця 3.8 – Структура таблиці "Товари в кошику" (CartItem)

Назва атрибуту	Тип	Пояснення
id	Integer	Унікальний ідентифікатор запису про товар у кошику
cart	ForeignKey	Зв'язок з кошиком, до якого додано товар
product	ForeignKey	Зв'язок з товаром, який додано до кошика
quantity	PositiveInteger	Кількість одиниць товару в кошику
price	DecimalField	Ціна товару на момент додавання до кошика
created_at	DateTimeField	Дата додавання товару до кошика
updated_at	DateTimeField	Дата останнього оновлення кількості товару

Таблиця Замовлення (табл. 3.8) призначена для зберігання замовлень користувачів. Містить інформацію про статус замовлення, спосіб оплати та доставки, загальну суму та зв'язок з адресою доставки. Відстежує процес обробки замовлення від створення до доставки.

Таблиця 3.9 – Структура таблиці "Замовлення" (Order)

Назва атрибуту	Тип	Пояснення
id	Integer	Унікальний ідентифікатор замовлення в системі
user	ForeignKey	Зв'язок з користувачем, який зробив замовлення
status	CharField	Статус замовлення (очікує обробки, в обробці, відправлено, доставлено, скасовано)
payment_method	CharField	Спосіб оплати (готівка, карта, банківський переказ)
delivery_method	CharField	Спосіб доставки (Нова Пошта, Укрпошта, самовивіз)
total_price	DecimalField	Загальна сума замовлення
address	ForeignKey	Зв'язок з OrderAddress, що містить детальну інформацію про адресу доставки конкретного замовлення
created_at	DateTimeField	Дата створення замовлення
updated_at	DateTimeField	Дата останнього оновлення замовлення
is_paid	Boolean	Прапорець оплати замовлення

Таблиця Товари замовлення (табл. 3.10) призначена для зберігання товарів у замовленні. Фіксує кількість та ціну кожного товару на момент створення замовлення. Це дозволяє зберегти історичну інформацію про ціни товарів.

Таблиця 3.10 - Структура таблиці "Товари замовлення" (OrderItem)

Назва атрибуту	Тип	Пояснення
id	Integer	Унікальний ідентифікатор запису про товар у замовленні
order	ForeignKey	Зв'язок з замовленням, до якого належить товар
product	ForeignKey	Зв'язок з товаром, який був замовлений
quantity	PositiveInteger	Кількість одиниць товару в замовленні
price	DecimalField	Ціна товару на момент створення замовлення

Таблиця Адреси замовлень (табл. 3.11) призначена для зберігання адрес доставки замовлень. Містить детальну інформацію про адресу, контактні дані отримувача та додаткові коментарі щодо доставки. Дозволяє зберігати різні адреси для різних замовлень одного користувача.

Таблиця 3.11 - Структура таблиці "Адреси замовлень" (OrderAddress)

Назва атрибуту	Тип	Пояснення
id	Integer	Унікальний ідентифікатор адреси в системі
first_name	CharField	Ім'я користувача
last_name	CharField	Прізвище користувача

Продовження таблиці 3.11

Назва атрибуту	Тип	Пояснення
email	EmailField	Email користувача
phone	CharField	Телефон користувача
city	CharField	Місто проживання
postal_code	CharField	Поштовий індекс
address	TextField	Адреса доставки
comment	TextField	Додаткові коментарі щодо доставки

Таким чином, запропонована структура бази даних забезпечує повний функціонал для роботи інтернет-магазину, включаючи:

- управління товарами та їх категоріями;
- обробку замовлень та управління кошиками;
- роботу з користувачами та їх профілями;
- зберігання історичних даних про ціни та замовлення.

Використання Django ORM дозволяє ефективно керувати даними та забезпечує їх цілісність. Розділення даних на логічні блоки (товари, користувачі, замовлення) сприяє зручності обслуговування та масштабування системи. Ієрархічна структура категорій та гнучка система характеристик товарів забезпечують зручну навігацію та пошук товарів для користувачів.

3.2 Функціонування програмного забезпечення

Функціонування програмного забезпечення онлайн-магазину комп'ютерних комплектуючих реалізовано на основі модульної архітектури з використанням фреймворку Django. Основні бізнес-процеси системи охоплюють роботу з каталогом товарів, управління кошиком, оформлення замовлень, а також адміністративне керування товарами та замовленнями. Взаємодія між

користувачем і системою здійснюється через вебінтерфейс, що забезпечує зручність та інтуїтивність використання.

Для наочного відображення логіки роботи програмного забезпечення розроблено структурну схему (рис. 3.2), яка демонструє основні етапи та сценарії взаємодії користувача з системою.

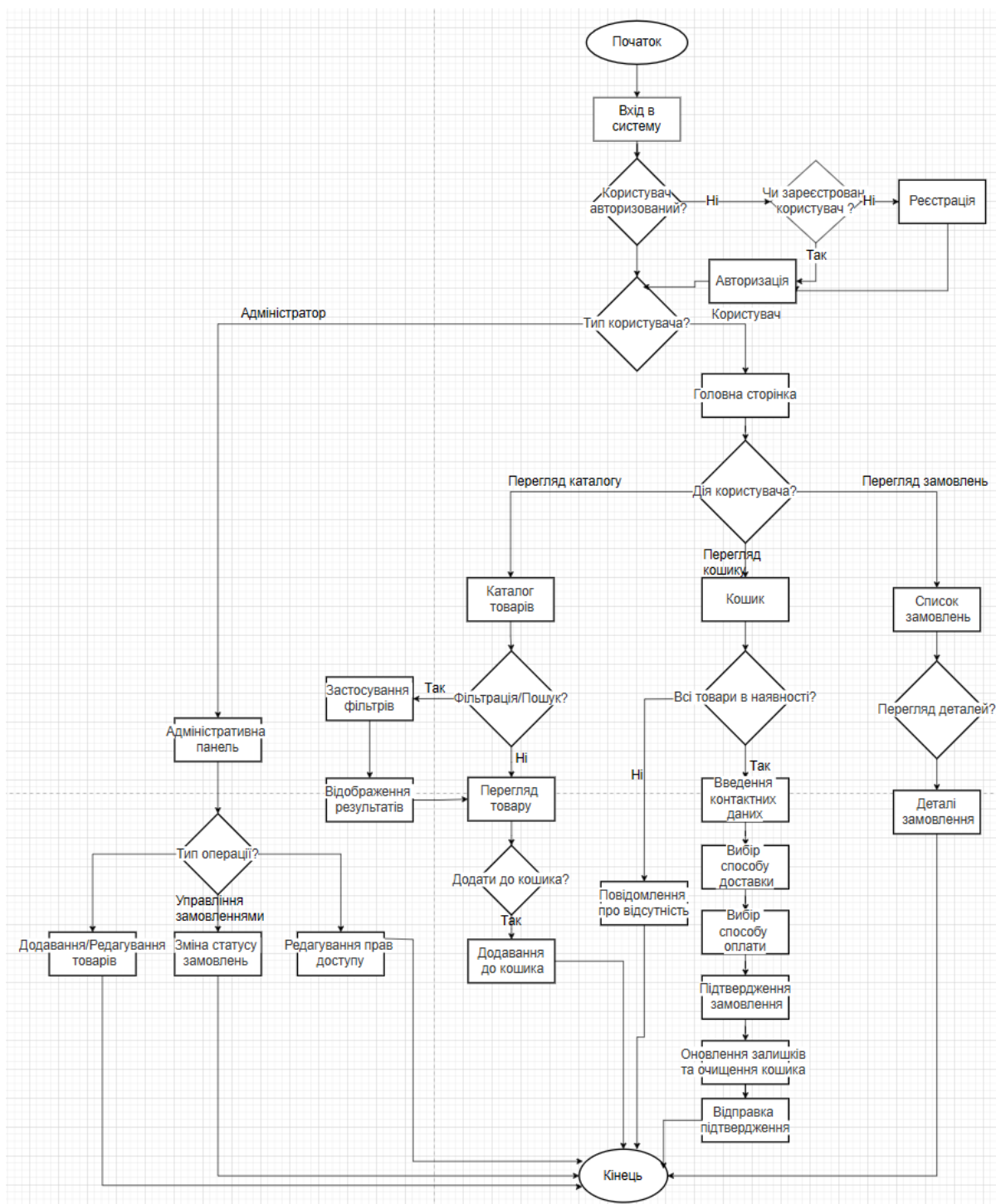


Рисунок 3.2 – Загальна схема функціонування програмного забезпечення онлайн-магазину

На схемі зображено основні процеси, що реалізовані у програмному забезпеченні онлайн-магазину:

- авторизація та реєстрація користувачів: система дозволяє новим користувачам реєструватися, а зареєстрованим – входити до особистого кабінету. Від типу користувача (адміністратор чи звичайний користувач) залежить доступний функціонал;
- робота з каталогом товарів: користувачі можуть переглядати каталог, здійснювати пошук і фільтрацію товарів за різними параметрами, переглядати детальну інформацію про товар;
- управління кошиком: додавання товарів до кошика, перевірка наявності, редагування кількості, видалення товарів;
- оформлення замовлення: перевірка наявності товарів, введення контактних даних, вибір способу доставки та оплати, підтвердження замовлення, оновлення залишків на складі;
- перегляд і керування замовленнями: користувачі можуть переглядати історію своїх замовлень, а адміністратори – змінювати статуси замовлень, керувати товарами та правами доступу інших користувачів.

Кожен з етапів, відображених на схемі, реалізовано у відповідних модулях проекту (accounts, products, cart, orders). Взаємодія між модулями забезпечує цілісність бізнес-логіки та зручність для кінцевого користувача.

3.3 Опис структури та інтерфейсу вебсайту

3.3.1 Головна сторінка

Головна сторінка онлайн-магазину (див. рис. 3.3) є центральним елементом взаємодії користувача із системою. Вона містить основне меню, пошук, фільтри за категоріями та виробниками, а також блоки для відображення товарів. Інтерфейс реалізовано на основі шаблону base.html, який забезпечує єдиний стиль для всіх сторінок сайту.

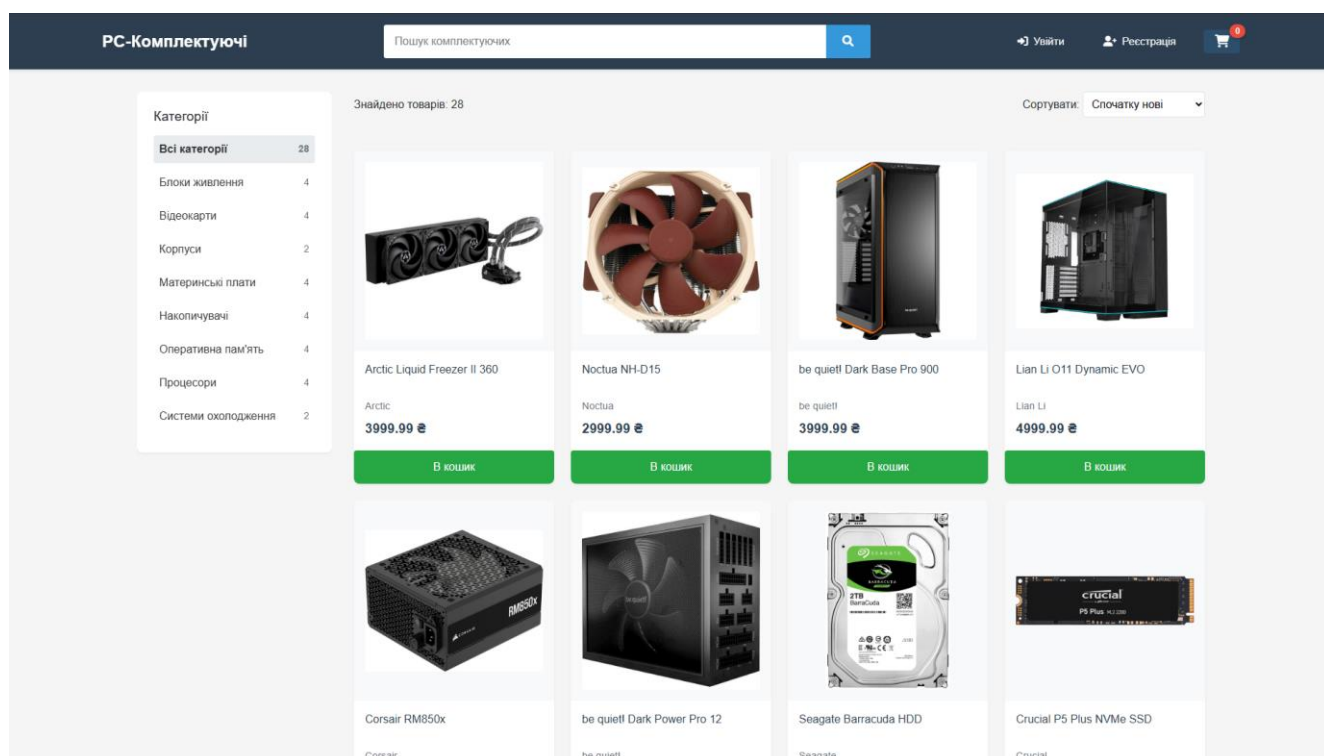


Рисунок 3. 3 – Головна сторінка вебсайту

На головній сторінці реалізовано такі ключові алгоритми.

Алгоритм 1. Пошук і фільтрація товарів.

Користувач може здійснювати пошук товарів за ключовими словами, а також фільтрувати їх за категоріями, виробниками, ціною та іншими характеристиками. Для цього у відповідному view (products/views.py, функція для виводу списку товарів) обробляються GET-параметри, формується запит до бази даних із урахуванням вибраних фільтрів, а результати відображаються на сторінці з пагінацією.

Алгоритм 2. Сортування товарів.

Реалізовано можливість сортувати товари за ціною, новизною або іншими параметрами. Сортування здійснюється шляхом зміни порядку елементів у queryset відповідно до вибраного критерію.

Алгоритм 3. Вибір категорії.

Меню категорій формується динамічно на основі даних із моделі категорій. При виборі категорії товари фільтруються відповідно до неї.

Алгоритм 4. Відображення повідомлень

Для інформування користувача про успішне додавання товару до кошика, помилки чи інші події використовується система повідомлень Django. Повідомлення відображаються у нижній правій частині сторінки.

Алгоритм 5. Перевірка профілю нового користувача.

Якщо користувач щойно зареєструвався і не заповнив профіль, на головній сторінці відображається відповідне повідомлення із закликом доповнити контактну інформацію. Це реалізовано через перевірку стану профілю у view-функції та передачу відповідного прапорця у шаблон.

Алгоритм 6. Додавання товару до кошика.

Після натискання кнопки “Додати до кошика” викликається відповідна функція, яка перевіряє наявність товару, додає його до кошика користувача або збільшує кількість, якщо товар вже є у кошику. Після цього користувач отримує повідомлення про успішне додавання.

Таким чином, головна сторінка поєднує в собі зручну навігацію, гнучкі інструменти пошуку та фільтрації, а також інтерактивні повідомлення, що забезпечують комфортну роботу користувача з онлайн-магазином.

3.3.2 Сторінка додавання/редагування товару

Сторінка додавання (див. рис. 3.4) та редагування товару призначена для адміністраторів магазину. Вони дозволяють створювати нові товари, редагувати існуючі, а також керувати категоріями, виробниками, фотографіями та характеристиками товарів. Доступ до цієї сторінки мають лише користувачі з відповідними правами.

Рисунок 3.4 – Сторінка додавання/редагування товару

На цій сторінці реалізовано такі основні алгоритми.

Алгоритм 1. Створення та редагування товару.

За додавання нового товару або редагування існуючого відповідають функції у файлі `products/views.py` (функції `add_product`, `edit_product`). При створенні товару адміністратор заповнює форму, яка містить основні поля: назва, опис, ціна,

категорія, виробник, кількість на складі, характеристики. При редагуванні дані товару автоматично підтягуються у форму для зручності зміни.

Алгоритм 2. Керування категоріями та виробниками.

На сторінці доступна можливість створювати нові категорії та виробників, а також редагувати або видаляти існуючі. Для цього використовуються окремі форми, які взаємодіють із моделями `Category` та `Manufacturer`. При створенні нової категорії можна вказати батьківську категорію, що дозволяє формувати ієрархічну структуру каталогу.

Алгоритм 3. Додавання та керування фотографіями товару.

Адміністратор може завантажити кілька фотографій для кожного товару, а також вибрати головне фото, яке буде відображатися у каталозі. Фотографії зберігаються у відповідній директорії (`media/products/`), а інформація про них – у моделі товару. Вибір головного фото реалізовано через відповідне поле у формі.

Алгоритм 4. Робота з характеристиками категорій.

Для кожної категорії можна визначити набір характеристик (наприклад, об'єм пам'яті, тип процесора тощо). Додавання, редагування та видалення характеристик здійснюється через окремі форми, які взаємодіють із моделлю `ProductSpecificationTemplate`. При створенні або редагуванні товару адміністратор може заповнити значення характеристик відповідно до шаблону категорії.

Алгоритм 5. Валідація даних.

Перед збереженням товару система перевіряє коректність введених даних (наприклад, унікальність назви, правильність формату ціни, наявність обов'язкових полів). Всі перевірки реалізовані у відповідних формах (`products/forms.py`).

Алгоритм 6. Відображення повідомлень.

Після успішного додавання, редагування або видалення товару, категорії чи виробника користувач отримує відповідне повідомлення про результат дії.

Таким чином, сторінка додавання/редагування товару забезпечує повний цикл керування асортиментом магазину, дозволяючи гнучко налаштовувати структуру каталогу, характеристики товарів та їх візуальне представлення.

3.3.3 Сторінка перегляду товару

Сторінка перегляду товару (див. рис. 3.5) призначена для ознайомлення користувача з детальною інформацією про обраний товар. Тут відображаються фотографії, опис, технічні характеристики, ціна, наявність на складі, а також кнопка для додавання товару до кошика. Дизайн сторінки забезпечує зручний доступ до всієї необхідної інформації для прийняття рішення про покупку.

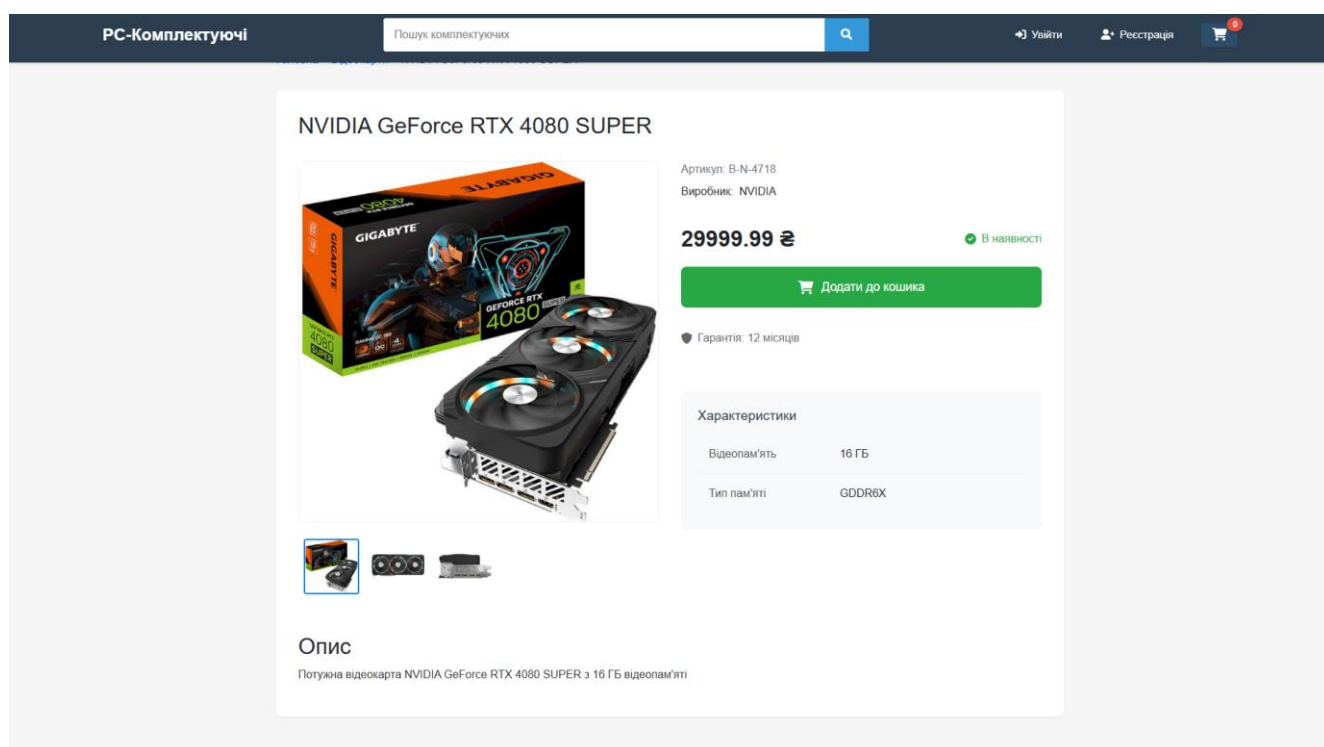


Рисунок 3.5 – Сторінка перегляду товару

На цій сторінці реалізовано такі основні алгоритми.

Алгоритм 1. Галерея зображень та збільшення фото.

Для кожного товару може бути основне зображення (`main_image`) і додаткові фото (`product.images.all`). Користувач може натискати на мініатюри додаткових фото, щоб змінити основне зображення (JS-функція `changeMainImage`). При натисканні на основне зображення відкривається модальне вікно з його збільшеною версією (`openImageModal`). Закриття модального вікна реалізовано як

по кнопці, так і по кліку поза зображенням. Весь функціонал реалізовано у JS-кодi, що міститься у шаблоні.

Алгоритм 2. Відображення характеристик товару.

Технічні характеристики товару зберігаються у полі `specifications` моделі `Product` і відображаються у вигляді таблиці через `product.get_specifications_display.items`.

Алгоритм 3. Адміністративні дії (для `staff`).

Якщо користувач має права адміністратора, на сторінці з'являються кнопки для редагування та видалення товару, а також для видалення окремих зображень.

Алгоритм 4. Відображення повідомлень.

Всі дії супроводжуються повідомленнями для користувача (наприклад, про успішне додавання до кошика чи видалення зображення).

Таким чином, сторінка перегляду товару забезпечує зручний перегляд усіх зображень із можливістю збільшення, динамічне перемикання основного фото, повний доступ до характеристик і опису, а також інтеграцію з кошиком та адміністративними функціями.

3.3.4 Сторінка адміністрування замовлень

Сторінка адміністрування замовлень (див. рис. 3.6) призначена для користувачів із правами адміністратора магазину. Вона дозволяє переглядати всі замовлення, переглядати деталі кожного замовлення, а також змінювати статус замовлення та відмічати факт оплати. Доступ до цієї сторінки мають лише авторизовані користувачі з відповідними правами.

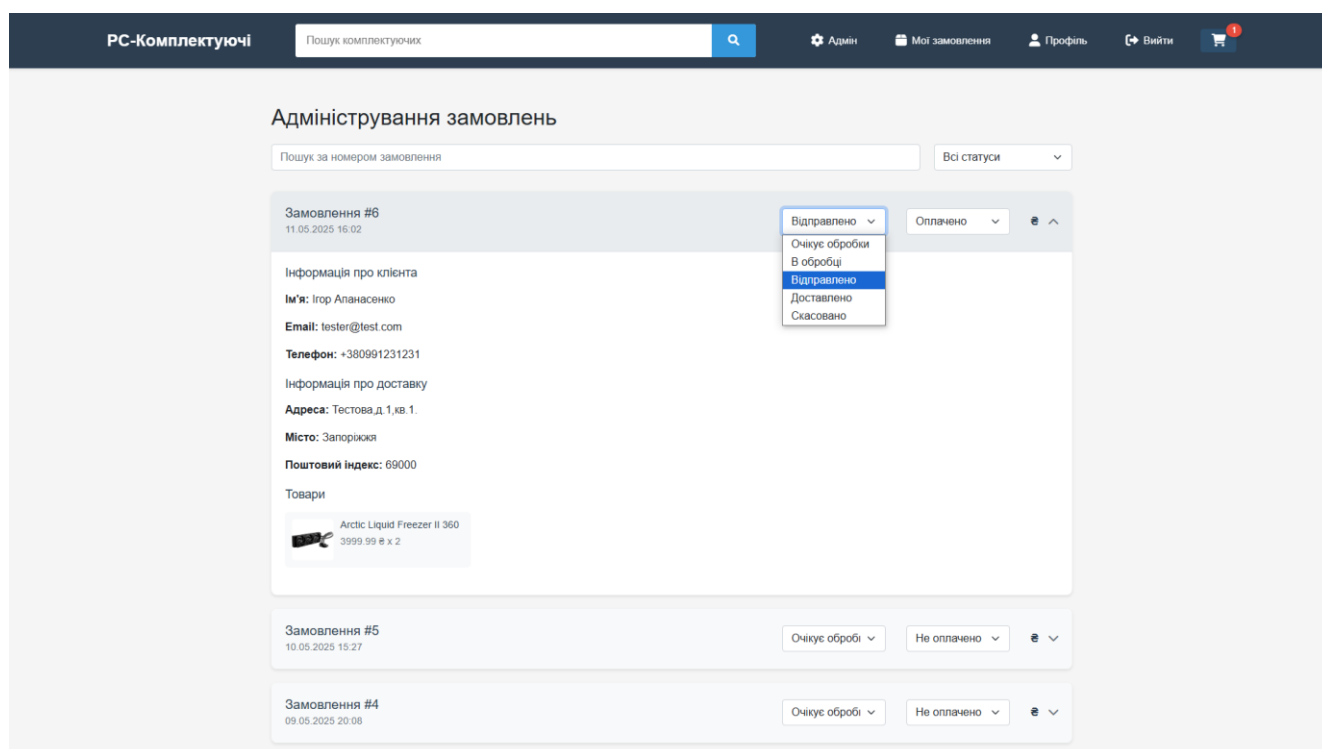


Рисунок 3.6 – Сторінка адміністрування замовлень

На цій сторінці реалізовано такі основні алгоритми.

Алгоритм 1. Відображення списку замовлень.

У файлі `orders/views.py` реалізовано функцію, яка формує список усіх замовлень (`admin_orders`). Замовлення відображаються у вигляді таблиці з основною інформацією: номер замовлення, користувач, дата створення, статус, спосіб оплати, чи оплачено. Для зручності реалізовано фільтрацію та сортування замовлень за статусом.

Алгоритм 2. Перегляд деталей замовлення.

При виборі конкретного замовлення адміністратор переходить на сторінку з детальною інформацією. Відображається повний склад замовлення: перелік товарів, кількість, ціна кожного товару, контактні дані покупця, адреса доставки, коментарі. Доступна інформація про історію змін статусу замовлення.

Алгоритм 3. Зміна статусу замовлення.

На сторінці деталей замовлення адміністратор може змінити статус замовлення (наприклад, “очікує обробки”, “в обробці”, “відправлено”, “доставлено”, “скасовано”). Для цього реалізовано випадаючий список, який

дозволяє вибрати новий статус. Після зміни статусу викликається відповідна функція у `orders/views.py`, яка оновлює поле `status` у моделі `Order` і зберігає зміни в базі даних. Зміна статусу супроводжується повідомленням для адміністратора про успішне оновлення.

Алгоритм 4. Відмітка про оплату.

Адміністратор може відзначити, що замовлення оплачено (поле `is_paid` у моделі `Order`). Після зміни статусу оплати інформація оновлюється у базі даних, а адміністратор отримує повідомлення про результат дії.

Таким чином, сторінка адміністрування замовлень забезпечує повний контроль над процесом обробки замовлень, дозволяє оперативно змінювати їх статус, відмічати оплату, переглядати деталі та історію кожного замовлення, що є важливою складовою ефективної роботи інтернет-магазину.

3.3.5 Сторінка перегляду замовлення

Сторінка перегляду замовлення (див. рис. 3.7) дозволяє користувачу ознайомитися з усією інформацією щодо конкретного замовлення: склад, статус, контактні та адресні дані, спосіб доставки й оплати, а також залишити коментар. Інтерфейс реалізовано на основі шаблону `order_detail`, який забезпечує зручний доступ до всіх деталей замовлення.

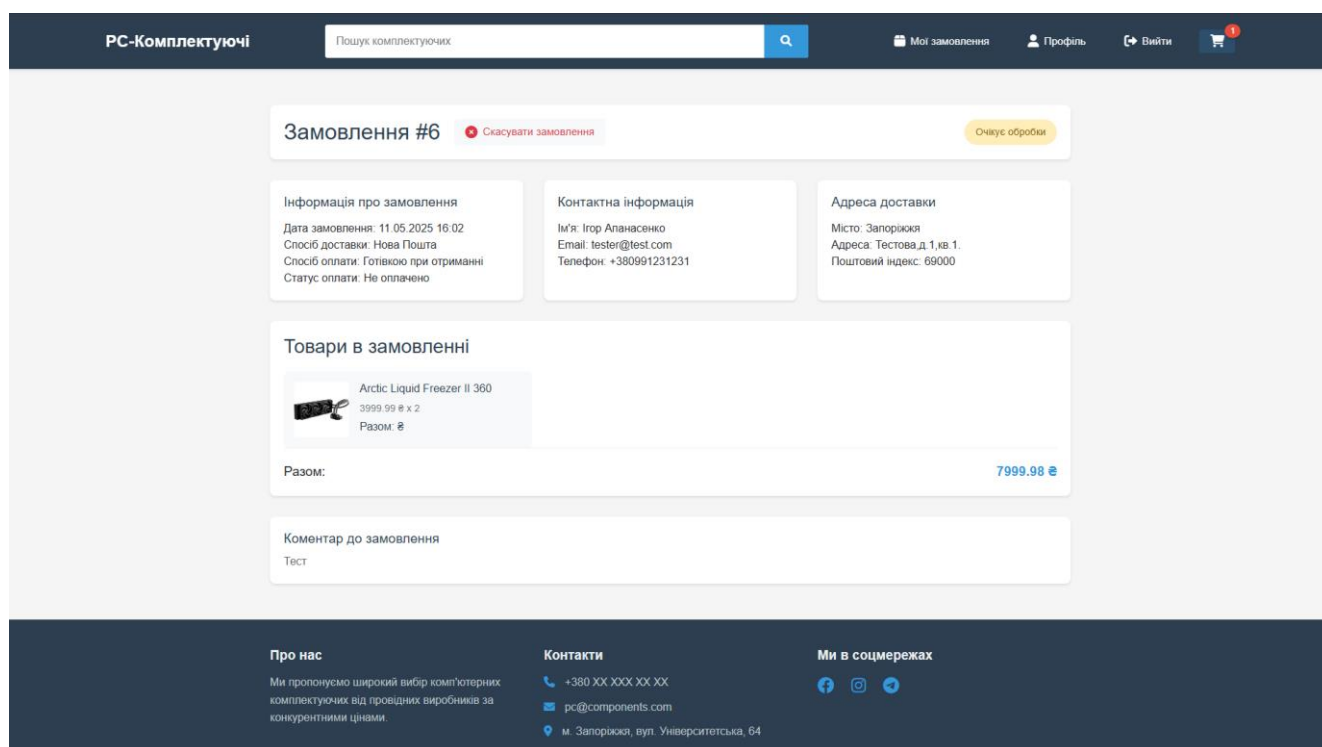


Рисунок 3.7 – Сторінка перегляду замовлення

На цій сторінці реалізовано такі основні алгоритми.

Алгоритм 1. Відображення детальної інформації про замовлення.

Вся інформація про замовлення (дата, статус, спосіб доставки, оплати, контактні дані, адреса, коментар) передається у шаблон через функцію `order_detail` у `orders/views.py`. Відображення статусу замовлення здійснюється з використанням кольорових бейджів для різних станів (очікує обробки, в обробці, відправлено, доставлено, скасовано).

Алгоритм 2. Відображення складу замовлення з переходом до товарів.

Для кожного товару у замовленні виводиться його назва, фото, ціна, кількість та підсумкова вартість. Назва та зображення кожного товару є посиланням, що веде на сторінку детального перегляду цього товару (`products:product_detail`). Це дозволяє користувачу швидко перейти до опису будь-якого товару з замовлення.

Алгоритм 3. Відображення загальної суми замовлення.

Під списком товарів виводиться підсумкова сума замовлення (`order.total_price`).

Алгоритм 4. Можливість скасування замовлення.

Якщо замовлення знаходиться у статусі “Очікує обробки” (pending), на сторінці відображається кнопка “Скасувати замовлення”. При натисканні на цю кнопку викликається JS-функція `cancelOrder`, яка надсилає POST-запит на адресу `/orders/cancel/<order_id>/`. У функції `cancel_order` у `orders/views.py` перевіряється, чи замовлення належить користувачу і чи має статус “Очікує обробки”. Якщо так – статус змінюється на “Скасовано”, і користувач отримує відповідне повідомлення.

Алгоритм 5. Відображення повідомлень.

Всі дії (наприклад, скасування замовлення) супроводжуються сповіщеннями для користувача, які відображаються у нижньому правому куті сторінки.

Алгоритм 6. Перевірка прав доступу.

Переглядати замовлення може лише користувач, якому воно належить.

Таким чином, сторінка перегляду замовлення забезпечує користувачу повний контроль над своїми замовленнями: дозволяє переглядати всі деталі, переходити до опису замовлених товарів, а також скасовувати замовлення, якщо воно ще не обробляється.

3.3.6 Сторінка кошика

Сторінка кошика (див. рис. 3.8) дозволяє користувачу переглядати всі додані товари, змінювати їхню кількість, видаляти товари, а також бачити підсумкову суму замовлення. Інтерфейс реалізовано на основі шаблону `cart_detail`, який забезпечує зручну взаємодію з кошиком.

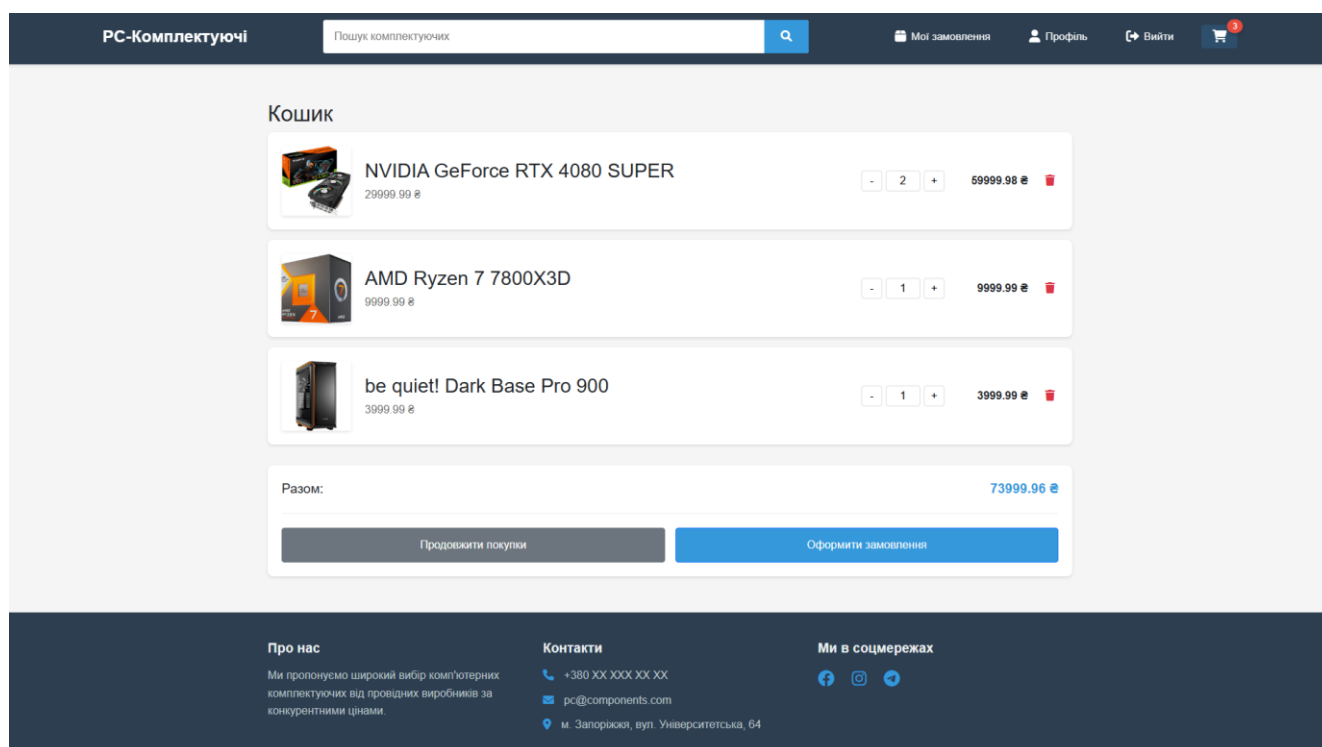


Рисунок 3.8 – Сторінка кошика

На цій сторінці реалізовано такі основні алгоритми.

Алгоритм 1. Відображення товарів у кошику.

Всі товари, які користувач додав до кошика, відображаються у вигляді списку з фото, назвою, ціною, кількістю та підсумковою вартістю для кожної позиції. Дані передаються у шаблон через функцію `cart_detail` у `cart/views.py`.

Алгоритм 2. Редагування кількості товару.

Користувач може змінювати кількість кожного товару за допомогою кнопок "+" і "-" або вручну у полі вводу. При зміні кількості відправляється POST-запит до функції `update_quantity` у `cart/views.py`. Система перевіряє, щоб нова кількість не перевищувала залишок на складі (`item.product.stock`). Якщо користувач вводить більше, ніж є на складі, кількість автоматично обмежується, а користувач отримує попередження. Після успішної зміни кількості оновлюється підсумкова сума для товару та всього кошика.

Алгоритм 3. Видалення товару з кошика.

Користувач може видалити будь-який товар із кошика. Для цього відправляється POST-запит до функції `remove_from_cart` у `cart/views.py`. Після

видалення товару оновлюється підсумкова сума, а якщо кошик порожній – відображається відповідне повідомлення.

Алгоритм 4. Перевірка залишків і обмеження на кількість.

Для кожного товару перевіряється, чи є він у наявності (`item.product.stock > 0`). Якщо товару немає на складі, відображається попередження, а кнопки зміни кількості та оформлення замовлення блокуються. Якщо кількість у кошику перевищує залишок, користувач бачить попередження і не може оформити замовлення, поки не зменшить кількість.

Алгоритм 5. Перевірка, чи товари вже замовлені.

Перед оформленням замовлення (у функції `orders/views.py:checkout`) додатково перевіряється, чи всі товари з кошика доступні у потрібній кількості. Якщо ні – користувач отримує повідомлення і повертається до кошика для коригування замовлення.

Алгоритм 6. Відображення підсумкової суми.

Під списком товарів виводиться загальна сума кошика (`total_price`), яка автоматично оновлюється при будь-яких змінах.

Алгоритм 7. Відображення повідомлень.

Всі дії (зміна кількості, видалення товару, помилки) супроводжуються інтерактивними повідомленнями для користувача.

Таким чином, сторінка кошика забезпечує повний контроль над складом замовлення, дозволяє гнучко керувати кількістю товарів, враховує залишки на складі та попереджає про всі можливі проблеми перед оформленням замовлення.

3.3.7 Сторінка оформлення замовлення

Сторінка оформлення замовлення (див. рис. 3.9) дозволяє користувачу ввести або підтвердити свої контактні дані, вибрати спосіб доставки та оплати, а також залишити коментар до замовлення. Інтерфейс реалізовано на основі шаблону `checkout`, який забезпечує зручне та швидке оформлення покупки.

PC-Комплектуючі

Пошук комплектуючих

Мои заповнення

Профіль

Вийти

1

Оформлення замовлення

Контактна інформація

Ім'я

Igor

Прізвище

Апанасенко

Електронна пошта

tester@test.com

Телефон

+380991231231

Спосіб доставки

Спосіб доставки

Нова Пошта

Адреса доставки

Місто

Запоріжжя

Поштовий індекс

69000

Тестова_д_1_кв_1

Спосіб оплати

Спосіб оплати

Готівкою при отриманні

Коментар до замовлення

Коментар

Введіть коментар до замовлення (необов'язково)

Оформити замовлення

Ваше замовлення

AMD Ryzen 7 7800X3D

1 x 9999.99 ₴

be quiet! Dark Base Pro 900

1 x 3999.99 ₴

NVIDIA GeForce RTX 4080 SUPER

2 x 29999.99 ₴

Разом:

73999.96 ₴

Про нас

Ми пропонуємо широкий вибір комп'ютерних комплектуючих від провідних виробників за конкурентними цінами.

Контакти

+380 XX XXX XX XX

pc@components.com

м. Запоріжжя, вул. Університетська, 64

Ми в соцмережах

Рисунок 3.9 – Сторінка оформлення замовлення

На цій сторінці реалізовано такі основні алгоритми.

Алгоритм 1. Автозаповнення форми контактної інформації.

При відкритті сторінки всі поля контактної інформації (ім'я, прізвище, email, телефон, місто, індекс, адреса) автоматично заповнюються даними з профілю користувача, якщо вони вже були вказані раніше. Це реалізовано у конструкторі

форми OrderAddressForm (orders/forms.py), де при ініціалізації форми з параметром user дані з моделі користувача та профілю підставляються у відповідні поля.

Алгоритм 2. Валідація номера телефону.

Для поля телефону встановлено шаблон і додаткову перевірку у методі clean_phone форми OrderAddressForm. Валідація здійснюється як на стороні клієнта (HTML-атрибут pattern), так і на сервері (перевірка формату +380XXXXXXXXXX, видалення нецифрових символів, повідомлення про помилку при некоректному номері).

Алгоритм 3. Підтвердження та створення замовлення.

Після натискання кнопки “Оформити замовлення” відбувається перевірка всіх полів. Якщо дані коректні: створюється новий запис адреси доставки (OrderAddress), оновлюється профіль користувача, якщо якісь поля були порожні, створюється новий запис замовлення (Order), до якого додаються всі товари з кошика (OrderItem), залишки товарів на складі зменшуються відповідно до замовленої кількості, кошик очищується, а користувач отримує повідомлення про успішне оформлення замовлення.

Алгоритм 4. Відображення підсумку замовлення.

У правій частині сторінки виводиться список усіх товарів, які будуть замовлені, із зазначенням кількості, ціни та загальної суми.

Алгоритм 5. Відображення повідомлень.

Усі помилки при заповненні форми, а також повідомлення про успішне оформлення замовлення, відображаються користувачу у вигляді інтерактивних сповіщень.

Таким чином, сторінка оформлення замовлення забезпечує максимально швидкий і зручний процес покупки, мінімізує кількість ручного введення даних, гарантує коректність контактної інформації та дозволяє користувачу контролювати всі етапи оформлення.

3.3.8 Сторінка профілю користувача

Сторінка профілю (див. рис. 3.10) користувача дозволяє переглядати та редагувати особисті дані, а також переглядати історію замовлень. Інтерфейс реалізовано на основі шаблону profile, що забезпечує зручний доступ до всіх основних функцій особистого кабінету.

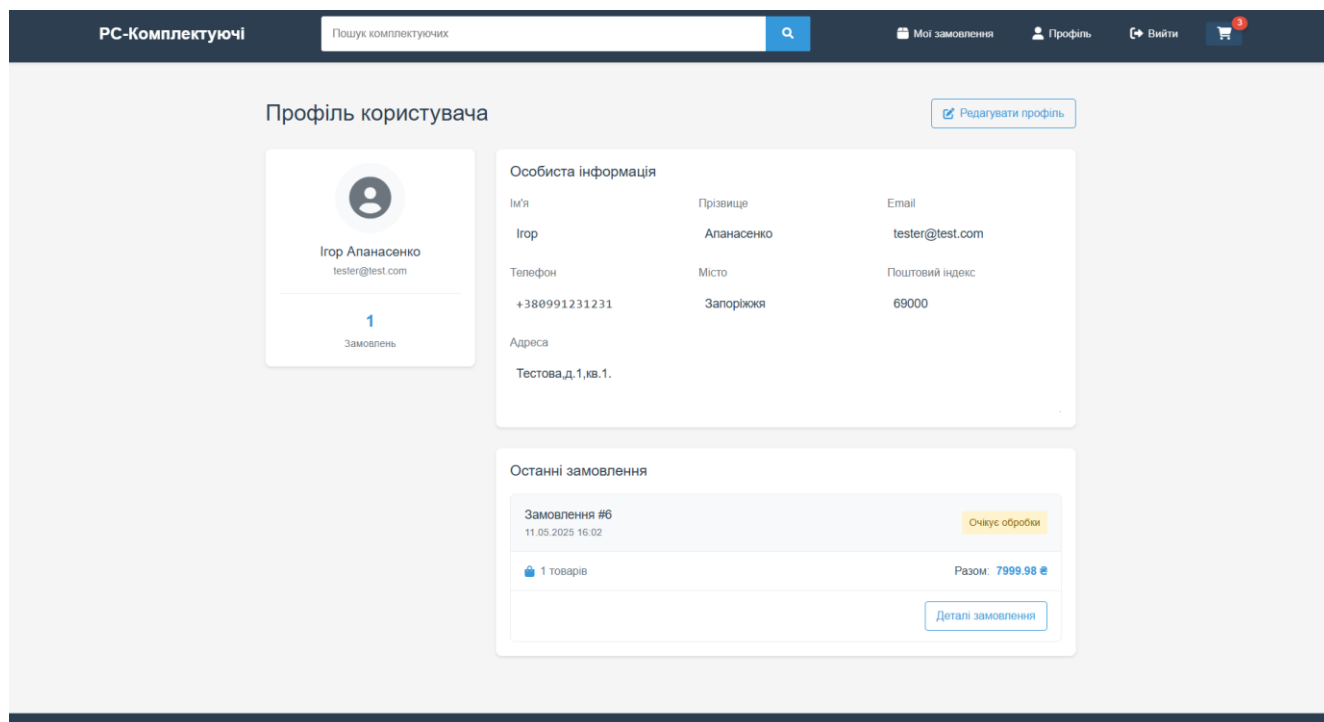


Рисунок 3.10 – Сторінка профілю користувача

На цій сторінці реалізовано такі основні алгоритми.

Алгоритм 1. Відображення та редагування особистих даних.

Користувач може переглядати свої основні дані (ім'я, прізвище, email, телефон, місто, індекс, адреса) та за потреби редагувати їх. Для цього передбачено режим редагування, який активується натисканням відповідної кнопки. Після внесення змін дані зберігаються через AJAX-запит до відповідної view-функції.

Алгоритм 2. Автоматичне підставлення даних у форму.

При відкритті сторінки всі поля форми автоматично заповнюються поточними даними користувача та його профілю.

Алгоритм 3. Валідація email та інших полів.

Email перевіряється на унікальність і коректність формату, інші поля – на заповнення та відповідність вимогам (довжина, відсутність заборонених символів).

Алгоритм 4. Збереження змін профілю.

Після натискання кнопки “Зберегти” дані відправляються через AJAX-запит. Якщо всі поля валідні, дані зберігаються у відповідних моделях (CustomUser та Profile). Користувач отримує повідомлення про успішне оновлення профілю.

Алгоритм 5. Перегляд історії замовлень.

На сторінці профілю відображається список усіх замовлень користувача з основною інформацією (номер, дата, сума, статус). Для кожного замовлення доступний перехід до детального перегляду.

Алгоритм 6. Відображення повідомлень.

Всі дії (успішне збереження, помилки валідації) супроводжуються інтерактивними повідомленнями для користувача.

Таким чином, сторінка профілю користувача забезпечує зручне керування особистими даними, гарантує коректність введення контактної інформації (зокрема, номера телефону), а також дозволяє контролювати історію замовлень.

3.4 Висновки за розділом 3

У розділі 3 було детально розглянуто основні рішення щодо реалізації компонентів системи інтернет-магазину комп'ютерних комплектуючих. Розроблена структура бази даних, що складається з 11 взаємопов'язаних таблиць, забезпечує повний функціонал для управління товарами, користувачами, кошиками та замовленнями. Використання Django ORM дозволило реалізувати гнучку та ефективну роботу з даними, забезпечуючи їх цілісність та безпеку. Модульна архітектура системи, побудована на основі Django, дозволила розділити функціонал на логічні блоки (accounts, products, cart, orders), що спрощує підтримку та розвиток системи. Розроблено 8 основних сторінок, кожна з яких має чітко визначені функції та алгоритми роботи: головна сторінка з пошуком та

фільтрацією, сторінка додавання/редагування товару для адміністраторів, сторінка перегляду товару з галереєю та характеристиками, сторінка адміністрування замовлень, сторінка перегляду замовлення, сторінка кошика, сторінка оформлення замовлення та сторінка профілю користувача. Кожна сторінка має інтуїтивно зрозумілий інтерфейс та реалізує необхідні перевірки та валідації. Реалізовано систему перевірок прав доступу для різних типів користувачів, впроваджено валідацію введених даних (номери телефонів, email, адреси), забезпечено контроль залишків товарів та перевірку доступності при оформленні замовлення. Система повідомлень інформує користувача про результати його дій, автозаповнення форм зменшує кількість ручного введення даних, а інтерактивні елементи (галерея, кошик, фільтри) покращують зручність використання. Таким чином, розроблена система інтернет-магазину комп'ютерних комплектуючих забезпечує повний цикл роботи з товарами та замовленнями, має зручний інтерфейс для користувачів та адміністраторів, а також готова до подальшого розвитку та масштабування.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Опис застосування програми

Розроблений вебсайт інтернет-магазину є сучасним програмним рішенням для організації онлайн-торгівлі комп'ютерними комплектуючими. Програма призначена для автоматизації процесів купівлі-продажу комп'ютерних комплектуючих через інтернет, забезпечуючи зручний інтерфейс як для покупців, так і для адміністраторів магазину.

Областю застосування програми є сфера електронної комерції, зокрема сегмент ринку комп'ютерних комплектуючих. Система дозволяє організувати повноцінний процес онлайн-торгівлі, починаючи від каталогізації товарів і закінчуючи оформленням замовлень та їх обробкою.

У розробці програми застосовані сучасні методи та технології веброзробки. Основний стек технологій включає фреймворк Django для бекенду, PostgreSQL як систему управління базами даних, та сучасні вебтехнології (HTML, CSS, JavaScript) для фронтенду. Для забезпечення безпеки використовуються вбудовані механізми Django, такі як CSRF-захист, система аутентифікації та авторизації.

Програма вирішує широкий спектр завдань, включаючи управління каталогом товарів, обробку замовлень, управління користувачькими акаунтами, роботу з кошиком покупок, а також адміністрування магазину. Система підтримує різні ролі користувачів (адміністратор, звичайний користувач), що дозволяє розділити функціональність відповідно до прав доступу.

До обмежень застосування програми можна віднести необхідність наявності стабільного інтернет-з'єднання для роботи з системою, а також вимоги до технічних характеристик серверного обладнання для забезпечення оптимальної продуктивності.

Мінімальна конфігурація технічних засобів для розгортання програми включає:

- сервер з операційною системою Linux або Windows Server;

- Python версії 3.8 або вище;
- PostgreSQL версії 12 або вище;
- мінімум 2 ГБ оперативної пам'яті;
- 20 ГБ вільного місця на диску;
- Стабільне інтернет-з'єднання зі швидкістю не менше 10 Мбіт/с.

Для користувачів вебінтерфейсу достатньо сучасного веббраузера (Chrome, Firefox, Safari або Edge останньої версії) та стабільного інтернет-з'єднання.

4.2 Тестування та експериментальне дослідження

Для перевірки коректності роботи вебсайту інтернет-магазину було проведено комплексне тестування, яке охопило всі основні функціональні модулі системи. Тестування проводилось шляхом ручної перевірки функціональності та експериментального дослідження роботи системи.

Методика тестування включала наступні етапи:

- функціональне тестування - перевірка роботи всіх основних функцій системи;
- тестування інтерфейсу - перевірка коректності відображення та взаємодії з користувачем.

Результати функціонального тестування представлені в табл. 4.1.

Таблиця 4.1 - Результати функціонального тестування

Модуль	Функціональність	Тестований сценарій	Результат	Примітки
Модуль облікових записів (accounts)	Реєстрація	Створення нового акаунту	Успішно	Перевірка валідації email, паролю
	Вхід в систему	Авторизація існуючого користувача	Успішно	Перевірка валідації логіну та паролю

Продовження таблиці 4.1

Модуль	Функціональність	Тестований сценарій	Результат	Примітки
	Профіль користувача	Перегляд та редагування профілю	Успішно	Перевірка оновлення контактних даних, адреси
	Перевірка профілю	Валідація повноти профілю	Успішно	Перевірка наявності всіх обов'язкових полів
Модуль товарів (products)	Каталог товарів	Перегляд списку товарів	Успішно	Перевірка пагінації, фільтрації, сортування
	Пошук товарів	Пошук за назвою, виробником, описом	Успішно	Перевірка пошуку за різними параметрами
	Фільтрація	Фільтрація за категорією, виробником, ціною	Успішно	Перевірка роботи всіх фільтрів
	Деталі товару	Перегляд інформації про товар	Успішно	Перевірка відображення всіх характеристик, фото
	Адміністрування	Додавання нового товару	Успішно	Перевірка завантаження фото, валідації даних
		Редагування товару	Успішно	Перевірка оновлення всіх полів
		Видалення товару	Успішно	Перевірка видалення з бази даних
Модуль кошика (cart)	Додавання товару	Додавання товару до кошика	Успішно	Перевірка контролю кількості, наявності на складі

Продовження таблиці 4.1

Модуль	Функціональність	Тестований сценарій	Результат	Примітки
	Оновлення кількості	Зміна кількості товару	Успішно	Перевірка обмежень по наявності
	Видалення товару	Видалення товару з кошика	Успішно	Перевірка оновлення загальної суми
	Перегляд кошика	Відображення товарів у кошику	Успішно	Перевірка відображення цін, кількості
Модуль замовлень (orders)	Оформлення замовлення	Створення нового замовлення	Успішно	Перевірка валідації номеру, зміни способу доставки
	Перегляд замовлення	Деталі конкретного замовлення	Успішно	Перевірка відображення всіх даних
	Список замовлень	Перегляд історії замовлень	Успішно	Перевірка сортування, фільтрації
	Адміністрування	Зміна статусу замовлення	Успішно	Перевірка всіх можливих статусів
		Відмітка про оплату	Успішно	Перевірка оновлення статусу оплати

Окрім функціонального тестування, було проведено детальне тестування інтерфейсу користувача. Особливу увагу було приділено адаптивності дизайну та зручності використання на різних пристроях. Тестування інтерфейсу включало перевірку всіх ключових елементів системи, їх коректного відображення та взаємодії з користувачем.

Результати тестування інтерфейсу представлені в табл. 4.2.

Таблиця 4.2 - Результати тестування інтерфейсу

Елемент інтерфейсу	Тестований сценарій	Результат	Примітки
Головне меню	Відображення на різних розмірах екрану	Успішно	Адаптивне меню на мобільних пристроях
	Навігація по пунктах меню	Успішно	Коректні переходи на всі сторінки
	Іконки та підписи	Успішно	Чітке відображення всіх елементів
Пошукова строка	Адаптивність на мобільних пристроях	Успішно	Повноекранний пошук на малих екранах
	Відображення результатів	Успішно	Зручне форматування результатів
Кошик	Міні-кошик при наведенні	Успішно	Плавна анімація появи
	Відображення кількості товарів	Успішно	Оновлення в реальному часі
	Перехід до повного кошика	Успішно	Коректні посилання
Каталог товарів	Відображення сітки товарів	Успішно	Адаптивна сітка
	Фільтри та сортування	Успішно	Зручний інтерфейс вибору
	Пагінація	Успішно	Коректна робота на всіх сторінках

Продовження таблиці 4.2

Елемент інтерфейсу	Тестований сценарій	Результат	Примітки
Картки товарів	Відображення фото	Успішно	Оптимізовані зображення
	Інформація про товар	Успішно	Чітка структура даних
	Кнопки дій	Успішно	Зручне розташування
Форми	Валідація полів	Успішно	Миттєвий зворотній зв'язок
	Адаптивність форм	Успішно	Коректне відображення на всіх пристроях
	Повідомлення про помилки	Успішно	Зрозумілі повідомлення
Сповіщення	Системні повідомлення	Успішно	Анімовані сповіщення
	Помилки валідації	Успішно	Чітке відображення помилок

За результатами тестування інтерфейсу можна зробити висновок, що всі елементи системи коректно відображаються та функціонують на різних пристроях. Особливу увагу було приділено мобільній версії сайту, оскільки вагома кількість користувачів використовують саме мобільні пристрої для шопінгу.

На рис. 4.1 наведено приклад мобільної версії головної сторінки сайту, яка демонструє адаптивний дизайн та оптимізований інтерфейс для користувачів мобільних пристроїв.

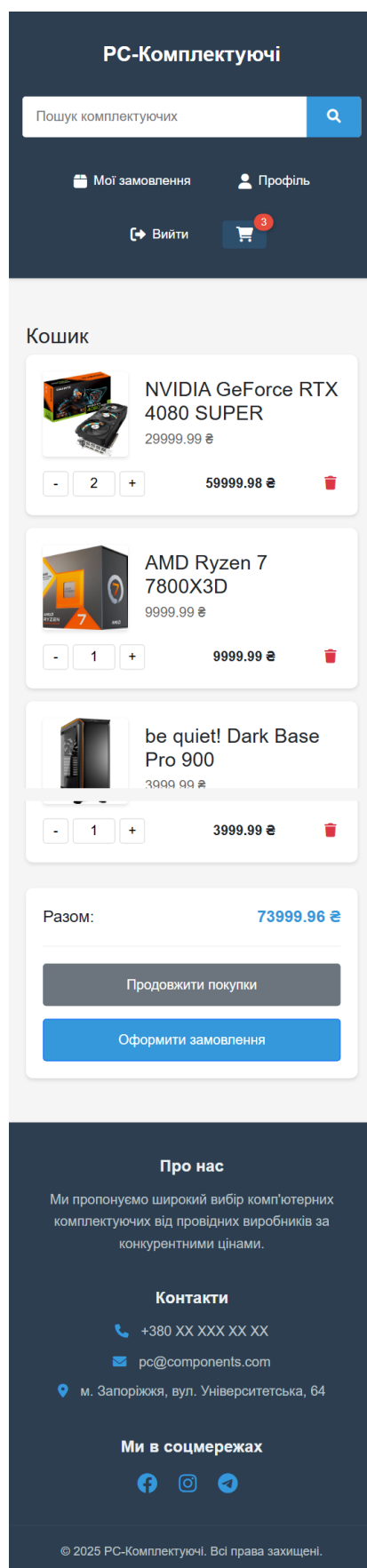


Рисунок 4.1 - Мобільна версія сторінки кошику інтернет-магазину

Як видно зі скріншоту, мобільна версія сайту забезпечує зручний доступ до всіх функцій інтернет-магазину. Інтерфейс оптимізований для використання на маленьких екранах.

За результатами проведеного тестування можна зробити висновок, що вебсайт інтернет-магазину успішно пройшов всі етапи перевірки. Система демонструє стабільну роботу, високу продуктивність та надійність. Всі основні функції працюють коректно, а інтерфейс забезпечує зручне використання на будь-яких пристроях. Система готова до впровадження та експлуатації.

4.3 Висновки за розділом 4

У ході розробки та тестування вебсайту інтернет-магазину було створено сучасне програмне рішення, яке повністю відповідає поставленим завданням. Система успішно реалізує всі необхідні функції для організації онлайн-торгівлі комп'ютерними комплектуючими, включаючи управління каталогом товарів, обробку замовлень, роботу з користувацькими акаунтами та адміністрування магазину. Комплексне тестування підтвердило стабільну роботу всіх модулів системи, високу продуктивність та надійність. Особливу увагу було приділено адаптивності інтерфейсу, що забезпечує зручне використання сайту на різних пристроях, включаючи мобільні. Система демонструє хорошу масштабованість та готова до впровадження в реальних умовах експлуатації. Всі виявлені під час тестування проблеми були успішно вирішені, що підтверджує готовність системи до промислового використання.

ВИСНОВКИ

З метою створення ефективної системи онлайн-продажу комп'ютерних комплектуючих у роботі вирішено актуальну проблему розробки та впровадження сучасного інтернет-магазину з використанням вебтехнологій. Проведено комплексний аналіз предметної області електронної комерції у сфері комп'ютерних комплектуючих, який виявив значні недоліки у сучасних інтернет-магазинах, зокрема застарілий дизайн, відсутність адаптивності, обмежені можливості фільтрації та незручний процес оформлення замовлень. Це підтвердило актуальність розробки нового рішення, яке б враховувало сучасні вимоги до UI/UX та функціональності.

Розроблено архітектуру системи та обґрунтовано вибір технологічного стеку. Для серверної частини обрано Django, що забезпечує високу швидкість розробки та масштабованість. Клієнтська частина реалізована з використанням HTML, CSS, JavaScript та Bootstrap, що дозволило створити сучасний адаптивний інтерфейс. Як СКБД обрано PostgreSQL, яка найкраще відповідає вимогам до структурованості та надійності даних.

Реалізовано повноцінну систему інтернет-магазину, що включає модульну архітектуру з чітким розділенням функціоналу, систему управління товарами з розширеними можливостями фільтрації, ефективну роботу з кошиком та оформленням замовлень, систему управління користувачами та їх правами, а також адміністративну панель для управління магазином. Система забезпечує зручний процес вибору та придбання комп'ютерних комплектуючих для користувачів та ефективне управління магазином для адміністраторів.

Проведено комплексне тестування системи, яке підтвердило коректну роботу всіх функціональних модулів, високу продуктивність та надійність системи, адаптивність інтерфейсу для різних пристроїв. Всі виявлені під час тестування проблеми були успішно вирішені, що підтверджує готовність системи до промислового використання.

Розроблена система має значний потенціал для подальшого розвитку. Перспективними напрямками є інтеграція з платіжними системами, розширення функціоналу для роботи з оптовими замовленнями, додавання системи рейтингів та відгуків, впровадження системи рекомендацій товарів, а також розвиток API для інтеграції з іншими системами. Система має модульну архітектуру, що дозволяє легко адаптувати її під конкретні потреби бізнесу та розширювати функціонал у міру розвитку.

Результати роботи демонструють, що розроблена система повністю відповідає сучасним вимогам до електронної комерції та готова до впровадження в реальних умовах. Розроблене рішення може бути використане як окремими продавцями, так і компаніями, що спеціалізуються на продажі комп'ютерних компонентів. Система забезпечує зручний процес вибору та придбання комп'ютерних комплектуючих, а також ефективне управління товарами та замовленнями через адміністративну панель.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Вебсайт інтернет-магазину комп'ютерних комплектуючих «CompBest» [Електрон. ресурс] . – Режим доступу: <https://compbest.com.ua/>.
2. Вебсайт інтернет-магазину комп'ютерних комплектуючих «NEON» [Електрон. ресурс] . – Режим доступу: <https://neon.ua/>.
3. Вебсайт інтернет-магазину комп'ютерних комплектуючих «itPlanet» [Електрон. ресурс] . – Режим доступу: <https://www.itplanet.zp.ua/>.
4. Django Підручник [Електрон. ресурс] . – Режим доступу: <https://w3schoolsua.github.io/django/>.
5. Django for eCommerce: Key Things to Consider in 2023 [Electronic resource] . – Access mode: <https://learnwoo.com/django-ecommerce/>.
6. Get started with Bootstrap [Electronic resource] . – Access mode: <https://getbootstrap.com/docs/5.3/getting-started/introduction/>.
7. Major Companies Using PostgreSQL: Purposes & Examples [Electronic resource] . – Access mode: <https://learnsql.com/blog/companies-that-use-postgresql-in-business/>.

ДОДАТОК А
Технічне завдання

Вступ

Дане технічне завдання стосується розробки вебзастосунку для онлайн-торгівлі комп'ютерними комплектуючими, який призначений для автоматизації процесу продажу та обліку товарів у сфері електронної комерції.

А.1 Підстави для розробки

Дипломна кваліфікаційна робота виконувалась у відповідності та на підставі Наказу № 209 від 28 квітня 2025 р. за Національним університетом «Запорізька політехніка», яким визначено тему роботи – «Вебсайт для онлайн-продажу комп'ютерних комплектуючих».

А.2 Призначення розробки

Розроблюваний програмний продукт призначений для забезпечення користувачів можливістю переглядати каталог товарів, здійснювати пошук, фільтрацію, оформлення замовлень та керування особистим профілем. Також система автоматизує облік товарів, замовлень і клієнтів для адміністратора магазину, що сприяє підвищенню ефективності процесу продажу та обслуговування клієнтів.

А.3 Вимоги до програми

А.3.1 Вимоги до функціональних характеристик

Розроблюваний програмний продукт має такі функціональні характеристики:

- реєстрація та автентифікація користувачів: можливість створення облікового запису, вхід до системи;

- перегляд каталогу товарів: доступ до структурованого каталогу з категоріями, фільтрацією та пошуком за різними параметрами (виробник, ціна, характеристики тощо);
- перегляд детальної інформації про товар: відображення опису, технічних характеристик, фотографій, наявності на складі;
- робота з кошиком: додавання, видалення та редагування кількості товарів у кошику, перегляд загальної суми замовлення;
- оформлення замовлення: заповнення контактних даних, вибір способу доставки та оплати, підтвердження замовлення;
- управління замовленнями: перегляд історії замовлень для користувача, зміна статусу замовлення адміністратором, обробка нових замовлень;
- адміністративна панель: управління товарами (додавання, редагування, видалення), категоріями, користувачами;
- адаптивний інтерфейс: коректне відображення сайту на різних пристроях (ПК, планшет, смартфон).

A.3.2 Вимоги до надійності

Забезпечення цілісності даних при одночасній роботі декількох користувачів. Відновлення роботи після збоїв (автоматичне збереження даних, резервне копіювання). Контроль коректності введення даних (валідація форм).

A.3.3 Умови експлуатації

Програма повинна працювати на сучасних персональних комп'ютерах з ОС Windows, Linux або macOS. Для користувача потрібен сучасний браузер (Google Chrome, Mozilla Firefox, Microsoft Edge тощо). Серверна частина розміщується на хостингу з підтримкою Python 3.8+, PostgreSQL.

A.3.4 Вимоги до складу та параметрів технічних засобів

Сервер: CPU 2+ ядра, RAM 2+ ГБ, HDD/SSD 20+ ГБ, ОС Linux/Windows.

Клієнт: ПК/ноутбук/смартфон з браузером.

A.4 Порядок контролю та приймання

Виконання роботи здійснюється відповідно до затвердженого технічного завдання під контролем керівника. Проміжні та кінцеві результати розробки оформлюються у вигляді пояснювальної записки. Остаточний контроль і приймання результатів виконаної бакалаврської кваліфікаційної роботи здійснюється під час її захисту перед екзаменаційною комісією.

ДОДАТОК Б
Текст програми

Б.1 Текст файлу products/views_api.py

```

from django.http import JsonResponse
from django.views.decorators.http import require_http_methods, require_POST,
require_GET
from django.contrib.auth.decorators import login_required, user_passes_test
from django.shortcuts import get_object_or_404
from django.views.decorators.csrf import ensure_csrf_cookie
from django.utils.text import slugify
from .models import Category, ProductSpecificationTemplate, Manufacturer, Product,
ProductImage
from orders.models import OrderItem
import json

@require_http_methods(["GET"])
def get_specification_template(request, category_id):
    try:
        category = Category.objects.get(id=category_id)
        template =
        ProductSpecificationTemplate.objects.filter(category=category).first()

        if template:
            return JsonResponse({
                'success': True,
                'specifications': template.specifications
            })
        else:
            return JsonResponse({
                'success': True,
                'specifications': {}
            })
    except Category.DoesNotExist:
        return JsonResponse({
            'success': False,
            'error': 'Категорію не знайдено'
        }, status=404)
    except Exception as e:
        return JsonResponse({
            'success': False,
            'error': str(e)
        }, status=500)

@ensure_csrf_cookie
@login_required
@require_http_methods(["POST"])
def add_category(request):
    try:
        data = json.loads(request.body)
        name = data.get('name')
        if not name:
            return JsonResponse({'success': False, 'error': 'Назва категорії
обов\язкова'})

        if Category.objects.filter(name=name).exists():

```

```

        return JsonResponse({'success': False, 'error': 'Категорія з такою
назвою вже існує'})

    category = Category(name=name)

    category.save()

    return JsonResponse({
        'success': True,
        'category': {
            'id': category.id,
            'name': category.name,
            'slug': category.slug
        }
    })
except json.JSONDecodeError:
    return JsonResponse({'success': False, 'error': 'Неправильний формат
даних'})
except Exception as e:
    return JsonResponse({'success': False, 'error': str(e)})

@login_required
@require_http_methods(["POST"])
def delete_category(request, category_id):
    try:
        category = Category.objects.get(id=category_id)

        if Product.objects.filter(category=category).exists():
            return JsonResponse({
                'success': False,
                'error': 'Неможливо видалити категорію, оскільки є товари, що
належать до неї'
            })
        category.delete()
        return JsonResponse({'success': True})
    except Category.DoesNotExist:
        return JsonResponse({'success': False, 'error': 'Категорію не знайдено'})
    except Exception as e:
        return JsonResponse({'success': False, 'error': str(e)})

@login_required
@require_http_methods(["POST"])
def add_manufacturer(request):
    try:
        data = json.loads(request.body)
        name = data.get('name')
        category_ids = data.get('categories', [])

        if not name:
            return JsonResponse({'success': False, 'error': 'Назва виробника
обов'язкова'})

        if not category_ids:
            return JsonResponse({'success': False, 'error': 'Оберіть хоча б одну
категорію'})

```

```

        if Manufacturer.objects.filter(name=name).exists():
            return JsonResponse({'success': False, 'error': 'Виробник з такою
назвою вже існує'})

        manufacturer = Manufacturer.objects.create(name=name)

        categories = Category.objects.filter(id__in=category_ids)
        manufacturer.categories.set(categories)

        return JsonResponse({
            'success': True,
            'manufacturer': {
                'id': manufacturer.id,
                'name': manufacturer.name,
                'categories': list(categories.values('id', 'name'))
            }
        })
    except json.JSONDecodeError:
        return JsonResponse({'success': False, 'error': 'Неправильний формат
даних'})
    except Exception as e:
        return JsonResponse({'success': False, 'error': str(e)})

@login_required
@require_http_methods(["POST"])
def delete_manufacturer(request, manufacturer_id):
    try:
        manufacturer = Manufacturer.objects.get(id=manufacturer_id)

        if Product.objects.filter(manufacturer=manufacturer).exists():
            return JsonResponse({
                'success': False,
                'error': 'Неможливо видалити виробника, оскільки є товари, що
належать до нього'
            })
        manufacturer.delete()
        return JsonResponse({'success': True})
    except Manufacturer.DoesNotExist:
        return JsonResponse({'success': False, 'error': 'Виробника не знайдено'})
    except Exception as e:
        return JsonResponse({'success': False, 'error': str(e)})

@login_required
@require_http_methods(["POST"])
def delete_spec_option(request, template_id, option):
    try:
        template = ProductSpecificationTemplate.objects.get(id=template_id)
        specifications = template.specifications

        for spec_name, spec_data in specifications.items():
            if 'options' in spec_data and option in spec_data['options']:

```

```

        if
template.product_set.filter(specifications__contains={spec_name:
option}).exists():
            return JsonResponse({
                'success': False,
                'error': 'Неможливо видалити опцію, оскільки вона
використовується в товарах'
            })

            spec_data['options'].remove(option)
            template.save()
            return JsonResponse({'success': True})

        return JsonResponse({'success': False, 'error': 'Опцію не знайдено'})
    except ProductSpecificationTemplate.DoesNotExist:
        return JsonResponse({'success': False, 'error': 'Шаблон специфікації не
знайдено'})
    except Exception as e:
        return JsonResponse({'success': False, 'error': str(e)})

def get_category_manufacturers(request, category_id):
    """Отримати список виробників для обраної категорії"""
    category = get_object_or_404(Category, id=category_id)
    manufacturers = category.manufacturers.all()
    return JsonResponse({
        'manufacturers': [
            {'id': m.id, 'name': m.name}
            for m in manufacturers
        ]
    })

@login_required
@require_http_methods(['GET'])
def get_manufacturer_categories(request, manufacturer_id):
    try:
        manufacturer = Manufacturer.objects.get(id=manufacturer_id)
        categories = manufacturer.categories.all()
        all_categories = Category.objects.all()

        return JsonResponse({
            'selected_categories': [{'id': cat.id, 'name': cat.name} for cat in
categories],
            'all_categories': [{'id': cat.id, 'name': cat.name} for cat in
all_categories]
        })
    except Manufacturer.DoesNotExist:
        return JsonResponse({'error': 'Виробника не знайдено'}, status=404)
    except Exception as e:
        return JsonResponse({'error': str(e)}, status=500)

@login_required
@require_http_methods(['POST'])
def update_manufacturer_categories(request, manufacturer_id):
    try:
        manufacturer = Manufacturer.objects.get(id=manufacturer_id)
        data = json.loads(request.body)

```

```

category_ids = data.get('category_ids', [])

categories = Category.objects.filter(id__in=category_ids)
if len(categories) != len(category_ids):
    return JsonResponse({'error': 'Деякі категорії не знайдено'},
status=400)

manufacturer.categories.set(categories)

return JsonResponse({
    'message': 'Категорії успішно оновлено',
    'categories': [{ 'id': cat.id, 'name': cat.name } for cat in categories]
})
except Manufacturer.DoesNotExist:
    return JsonResponse({'error': 'Производитель не найден'}, status=404)
except json.JSONDecodeError:
    return JsonResponse({'error': 'Неверный формат данных'}, status=400)
except Exception as e:
    return JsonResponse({'error': str(e)}, status=500)

@require_POST
def update_category(request, category_id):
    try:

        category = get_object_or_404(Category, id=category_id)

        data = json.loads(request.body)
        new_name = data.get('name', '').strip()

        if not new_name:
            return JsonResponse({
                'success': False,
                'error': 'Назва категорії не може бути порожньою'
            })

        if
Category.objects.exclude(id=category_id).filter(name=new_name).exists():
            return JsonResponse({
                'success': False,
                'error': 'Категорія з такою назвою вже існує'
            })

        category.name = new_name
        category.save()

        return JsonResponse({
            'success': True,
            'message': 'Категорію успішно оновлено'
        })

```

```

except json.JSONDecodeError:
    return JsonResponse({
        'success': False,
        'error': 'Неправильный формат данных'
    })
except Exception as e:
    return JsonResponse({
        'success': False,
        'error': str(e)
    })

@login_required
@require_GET
def get_spec_template(request, template_id):
    try:
        template = ProductSpecificationTemplate.objects.get(id=template_id)
        return JsonResponse({
            'success': True,
            'template': {
                'id': template.id,
                'name': template.name,
                'field_type': template.field_type,
                'unit': template.unit or '',
                'is_filter': template.is_filter,
                'options': template.options if template.field_type == 'select' else
[],
                'min_value': template.min_value,
                'max_value': template.max_value
            }
        })
    except ProductSpecificationTemplate.DoesNotExist:
        return JsonResponse({'success': False, 'error': 'Шаблон характеристики не
найден'}, status=404)
    except Exception as e:
        return JsonResponse({'success': False, 'error': str(e)}, status=500)

@login_required
@require_POST
def add_spec_template(request, category_id):
    try:
        category = Category.objects.get(id=category_id)
        data = json.loads(request.body)

        name = data.get('name', '').strip()
        field_type = data.get('field_type')
        options = data.get('options', [])
        unit = data.get('unit', '')
        is_filter = data.get('is_filter', False)
        min_value = data.get('min_value')
        max_value = data.get('max_value')

        if not name:
            return JsonResponse({
                'success': False,
                'error': 'Назва характеристики обов\'язкова'
            })

```

```

        if not field_type or field_type not in ['text', 'select', 'range']:
            return JsonResponse({
                'success': False,
                'error': 'Неправильний тип поля'
            })

        if ProductSpecificationTemplate.objects.filter(category=category,
name=name).exists():
            return JsonResponse({
                'success': False,
                'error': 'Характеристика з такою назвою вже існує в цій категорії'
            })

        template = ProductSpecificationTemplate.objects.create(
            category=category,
            name=name,
            field_type=field_type,
            options=options if field_type == 'select' else None,
            unit=unit,
            is_filter=is_filter,
            min_value=min_value if field_type == 'range' else None,
            max_value=max_value if field_type == 'range' else None
        )

        return JsonResponse({
            'success': True,
            'template': {
                'id': template.id,
                'name': template.name,
                'field_type': template.field_type,
                'options': template.options if template.field_type == 'select' else
[],
                'unit': template.unit or '',
                'is_filter': template.is_filter,
                'min_value': template.min_value,
                'max_value': template.max_value
            }
        })

    except json.JSONDecodeError:
        return JsonResponse({
            'success': False,
            'error': 'Неправильний формат даних'
        })

    except Exception as e:
        return JsonResponse({
            'success': False,
            'error': str(e)
        })

@login_required
@require_POST
def update_spec_template(request, template_id):
    try:
        template = get_object_or_404(ProductSpecificationTemplate, id=template_id)

```

```

data = json.loads(request.body)

name = data.get('name', '').strip()
field_type = data.get('field_type')
unit = data.get('unit', '')
is_filter = data.get('is_filter', False)
min_value = data.get('min_value')
max_value = data.get('max_value')
options = data.get('options', [])

if not name:
    return JsonResponse({
        'success': False,
        'error': 'Назва характеристики обов\'язкова'
    })

if ProductSpecificationTemplate.objects.filter(
    category=template.category,
    name=name
).exclude(id=template_id).exists():
    return JsonResponse({
        'success': False,
        'error': 'Характеристика з такою назвою вже існує в цій категорії'
    })

template.name = name
template.unit = unit
template.is_filter = is_filter
template.field_type = field_type

if field_type == 'range':
    template.min_value = min_value
    template.max_value = max_value
elif field_type == 'select':
    template.options = options

template.save()

return JsonResponse({
    'success': True,
    'template': {
        'id': template.id,
        'name': template.name,
        'field_type': template.field_type,
        'unit': template.unit,
        'is_filter': template.is_filter,
        'min_value': template.min_value,
        'max_value': template.max_value,
        'options': template.options if template.field_type == 'select' else
[]
    }
})

except json.JSONDecodeError:
    return JsonResponse({
        'success': False,

```

```

        'error': 'Неправильний формат даних'
    })
except Exception as e:
    return JsonResponse({
        'success': False,
        'error': str(e)
    })

@login_required
@require_POST
def update_spec_option(request, template_id, option):
    try:
        template = get_object_or_404(ProductSpecificationTemplate, id=template_id)
        data = json.loads(request.body)
        new_option = data.get('new_option', '').strip()

        if not new_option:
            return JsonResponse({
                'success': False,
                'error': 'Нове значення не може бути порожнім'
            })

        if template.field_type != 'select':
            return JsonResponse({
                'success': False,
                'error': 'Ця характеристика не підтримує опції'
            })

        if option not in template.options:
            return JsonResponse({
                'success': False,
                'error': 'Опцію не знайдено'
            })

        if new_option in template.options:
            return JsonResponse({
                'success': False,
                'error': 'Така опція вже існує'
            })

        options = template.options
        index = options.index(option)
        options[index] = new_option
        template.options = options
        template.save()

        products = Product.objects.filter(category=template.category)
        for product in products:
            specs = product.specifications
            if template.name in specs and specs[template.name] == option:
                specs[template.name] = new_option
                product.specifications = specs
                product.save()

    return JsonResponse({
        'success': True,

```

```

        'message': 'Опцію успішно оновлено'
    })

except json.JSONDecodeError:
    return JsonResponse({
        'success': False,
        'error': 'Неправильний формат даних'
    })
except Exception as e:
    return JsonResponse({
        'success': False,
        'error': str(e)
    })

@login_required
@require_POST
def delete_spec_template(request, template_id):
    try:
        template = get_object_or_404(ProductSpecificationTemplate, id=template_id)

        if Product.objects.filter(category=template.category,
specifications__has_key=template.name).exists():
            return JsonResponse({
                'success': False,
                'error': 'Неможливо видалити характеристику, оскільки вона
використовується в товарах'
            })

        template.delete()
        return JsonResponse({
            'success': True,
            'message': 'Характеристику успішно видалено'
        })

    except Exception as e:
        return JsonResponse({
            'success': False,
            'error': str(e)
        })

@require_GET
def get_category_specs(request, category_id):
    try:
        print(f"Отримання характеристик для категорії {category_id}")
        category = Category.objects.get(id=category_id)
        specs = ProductSpecificationTemplate.objects.filter(category=category)
        specifications = []
        for spec in specs:
            print(f"Обробка характеристики: {spec.name}")
            spec_data = {
                'id': spec.id,
                'name': spec.name,
                'field_type': spec.field_type,
                'unit': spec.unit or '',
                'is_filter': spec.is_filter,
            }

```

```

        'options': spec.options if spec.field_type == 'select' and
spec.options else [],
        'min_value': spec.min_value if spec.field_type == 'range' else
None,
        'max_value': spec.max_value if spec.field_type == 'range' else
None,
    }
    specifications.append(spec_data)
    print(f"Успішно отримано {len(specifications)} характеристик")
    return JsonResponse({'success': True, 'specifications': specifications})
except Category.DoesNotExist:
    print(f"Категорію {category_id} не знайдено")
    return JsonResponse({'success': False, 'error': 'Категорію не знайдено'},
status=404)
except Exception as e:
    print(f"Помилка при отриманні характеристик: {str(e)}")
    return JsonResponse({'success': False, 'error': str(e)}, status=500)

@require_POST
@user_passes_test(lambda u: u.is_staff)
def delete_products(request):
    try:
        data = json.loads(request.body)
        product_ids = data.get('product_ids', [])

        products_in_orders = []
        for product_id in product_ids:
            product = Product.objects.get(id=product_id)
            if OrderItem.objects.filter(product=product).exists():
                products_in_orders.append(product.name)

        if products_in_orders:
            return JsonResponse({
                'success': False,
                'error': f'Неможливо видалити наступні товари, оскільки вони є в
заказах: {", ".join(products_in_orders)}'
            })

        Product.objects.filter(id__in=product_ids).delete()

        return JsonResponse({'success': True})
    except Exception as e:
        return JsonResponse({'success': False, 'error': str(e)})

@login_required
@user_passes_test(lambda u: u.is_staff)
def delete_image(request, image_id):
    if request.method != 'DELETE':
        return JsonResponse({'error': 'Метод не дозволено'}, status=405)

    image = get_object_or_404(ProductImage, id=image_id)
    image.delete()

    return JsonResponse({'success': True})

```

Б.2 Текст файла products/views.py

```

from django.contrib.auth.decorators import user_passes_test
from django.shortcuts import render, get_object_or_404, redirect
from django.http import HttpResponse, JsonResponse
from django.db.models import Q, Count
from .models import Product, Category, Manufacturer, ProductSpecificationTemplate,
ProductImage
from .forms import ProductForm, CategoryForm
from django.views.decorators.http import require_GET, require_POST
import json
from django.urls import reverse
from django.contrib.auth.decorators import login_required
from django.contrib import messages
from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger

def is_admin(user):
    return user.is_staff

def product_list(request, category_slug=None):
    products = Product.objects.filter(is_active=True)
    current_category = None
    categories = Category.objects.filter(parent=None)

    search_query = request.GET.get('search')
    if search_query:
        products = products.filter(
            Q(name__icontains=search_query) |
            Q(manufacturer__name__icontains=search_query) |
            Q(description__icontains=search_query) |
            Q(specifications__icontains=search_query)
        )

    if category_slug:
        current_category = get_object_or_404(Category, slug=category_slug)
        products = products.filter(category=current_category)

        manufacturers =
Manufacturer.objects.filter(products__category=current_category).distinct()
        spec_templates =
ProductSpecificationTemplate.objects.filter(category=current_category)
    else:
        manufacturers = None
        spec_templates = None

    selected_manufacturers = []
    manufacturer_ids = request.GET.getlist('manufacturer')
    if manufacturer_ids:
        selected_manufacturers = [int(m_id) for m_id in manufacturer_ids]
        products = products.filter(manufacturer_id__in=selected_manufacturers)

    min_price = request.GET.get('min_price')

```

```

max_price = request.GET.get('max_price')
if min_price:
    products = products.filter(price__gte=min_price)
if max_price:
    products = products.filter(price__lte=max_price)

selected_specs = {}
if spec_templates:
    for spec in spec_templates:
        if spec.field_type == 'select':
            values = request.GET.getlist(f'spec_{spec.id}')
            if values:
                selected_specs[spec.id] = values
                q_objects = Q()
                for value in values:
                    q_objects |= Q(specifications__contains={spec.name:
value})
                products = products.filter(q_objects)
            elif spec.field_type == 'range':
                min_val = request.GET.get(f'spec_{spec.id}_min')
                max_val = request.GET.get(f'spec_{spec.id}_max')
                if min_val or max_val:
                    selected_specs[spec.id] = {'min': min_val, 'max': max_val}

        products_with_spec =
products.filter(specifications__has_key=spec.name)

        if min_val:
            min_val = float(min_val)
            products_with_spec = products_with_spec.filter(**{
                f'specifications__{spec.name}__gte': min_val
            })

        if max_val:
            max_val = float(max_val)
            products_with_spec = products_with_spec.filter(**{
                f'specifications__{spec.name}__lte': max_val
            })

        products = products_with_spec

    sort = request.GET.get('sort', '-created_at')
    if sort:
        products = products.order_by(sort)

    page = request.GET.get('page', 1)
    paginator = Paginator(products, 20)
    try:
        page_obj = paginator.page(page)
    except PageNotAnInteger:
        page_obj = paginator.page(1)

```

```

except EmptyPage:
    page_obj = paginator.page(paginator.num_pages)

total_products = Product.objects.filter(is_active=True).count()

for category in categories:
    category.product_count = Product.objects.filter(category=category,
is_active=True).count()

if manufacturers:
    for manufacturer in manufacturers:
        manufacturer.product_count =
products.filter(manufacturer=manufacturer).count()

if spec_templates:
    for spec in spec_templates:
        if spec.field_type == 'select':
            spec.value_counts = {}
            for option in spec.options:
                count = products.filter(specifications__contains={spec.name:
option}).count()
                if count > 0:
                    spec.value_counts[option] = count

context = {
    'products': page_obj.object_list,
    'categories': categories,
    'current_category': current_category,
    'manufacturers': manufacturers,
    'selected_manufacturers': selected_manufacturers,
    'spec_templates': spec_templates,
    'selected_specs': selected_specs,
    'total_products': total_products,
    'search_query': search_query,
    'page_obj': page_obj,
    'paginator': paginator,
}

return render(request, 'products/product_list.html', context)

def product_detail(request, id, slug):
    product = get_object_or_404(Product, id=id, slug=slug)
    return render(request, 'products/product_detail.html', {'product': product})

@login_required
def add_product(request):
    if request.method == 'POST':
        product_form = ProductForm(request.POST)
        if product_form.is_valid():
            product = product_form.save(commit=False)

```

```

specifications = {}
for key, value in request.POST.items():
    if key.startswith('spec_'):
        spec_name = key.replace('spec_', '')
        if value:
            specifications[spec_name] = value

if specifications:
    product.specifications = specifications

images = request.FILES.getlist('images')
main_image_index = request.POST.get('main_image_index', 0)

if images:

    product.save()

    for i, image in enumerate(images):
        ProductImage.objects.create(
            product=product,
            image=image,
            is_main=(str(i) == str(main_image_index)),
            order=i
        )

    return JsonResponse({
        'success': True,
        'redirect_url': reverse('products:product_detail',
args=[product.id, product.slug])
    })
else:
    return JsonResponse({
        'success': False,
        'error': 'Необхідно завантажити хоча б одне зображення'
    })
else:
    return JsonResponse({
        'success': False,
        'error': 'Форма містить помилки'
    })
else:
    product_form = ProductForm()

    return render(request, 'products/add_product.html', {
        'product_form': product_form
    })

@user_passes_test(is_admin)
def edit_product(request, product_id):
    product = get_object_or_404(Product, id=product_id)

    if request.method == 'POST':

```

```

try:
    name = request.POST.get('name')
    description = request.POST.get('description')
    price = request.POST.get('price')
    stock = request.POST.get('stock')
    category_id = request.POST.get('category')
    manufacturer_id = request.POST.get('manufacturer')

    product.name = name
    product.description = description
    product.price = price
    product.stock = stock
    product.category_id = category_id
    product.manufacturer_id = manufacturer_id

try:
    specifications = json.loads(request.POST.get('specifications',
'{}'))
    if isinstance(specifications, dict):
        product.specifications = specifications
except json.JSONDecodeError:
    if request.headers.get('Accept') == 'application/json':
        return JsonResponse({
            'success': False,
            'error': 'Неправильный формат характеристик'
        })
    else:
        messages.error(request, 'Неправильный формат характеристик')
        return redirect('products:edit_product',
product_id=product_id)

images = request.FILES.getlist('images')
delete_images = request.POST.getlist('delete_images[]')
main_image = request.POST.get('main_image')

if delete_images:
    ProductImage.objects.filter(id__in=delete_images).delete()

if main_image:
    ProductImage.objects.filter(product=product).update(is_main=False)
    ProductImage.objects.filter(id=main_image).update(is_main=True)

for image in images:
    ProductImage.objects.create(
        product=product,
        image=image,
        is_main=False
    )

```

```

product.save()

    redirect_url = reverse('products:product_detail', args=[product.id,
product.slug])
    if request.headers.get('Accept') == 'application/json':
        return JsonResponse({
            'success': True,
            'redirect_url': redirect_url
        })
    else:
        messages.success(request, 'Товар успішно оновлено')
        return redirect(redirect_url)

except Exception as e:
    if request.headers.get('Accept') == 'application/json':
        return JsonResponse({
            'success': False,
            'error': str(e)
        })
    else:
        messages.error(request, f'Помилка при збереженні товару:
{str(e)}')
        return redirect('products:edit_product', product_id=product_id)

categories = Category.objects.all()
manufacturers = Manufacturer.objects.all()

spec_templates =
ProductSpecificationTemplate.objects.filter(category=product.category)

product_specs = []
if product.specifications:
    for name, value in product.specifications.items():

        spec_template = spec_templates.filter(name=name).first()
        if spec_template:
            spec_data = {
                'name': name,
                'value': value,
                'field_type': spec_template.field_type,
                'unit': spec_template.unit,
                'options': spec_template.options,
                'min_value': spec_template.min_value,
                'max_value': spec_template.max_value,
                'is_filter': spec_template.is_filter,
                'template_id': spec_template.id
            }
        else:

            spec_data = {
                'name': name,
                'value': value,

```

```

        'field_type': 'text',
        'unit': '',
        'options': [],
        'min_value': None,
        'max_value': None,
        'is_filter': False,
        'template_id': None
    }
    product_specs.append(spec_data)

existing_specs = {spec['name'] for spec in product_specs}
for template in spec_templates:
    if template.name not in existing_specs:
        spec_data = {
            'name': template.name,
            'value': '',
            'field_type': template.field_type,
            'unit': template.unit,
            'options': template.options,
            'min_value': template.min_value,
            'max_value': template.max_value,
            'is_filter': template.is_filter,
            'template_id': template.id
        }
        product_specs.append(spec_data)

context = {
    'product': product,
    'categories': categories,
    'manufacturers': manufacturers,
    'product_specs': product_specs,
}

return render(request, 'products/edit_product.html', context)

@user_passes_test(is_admin)
def delete_product(request, product_id):
    product = get_object_or_404(Product, id=product_id)
    product.delete()
    return redirect('products:product_list')

@require_GET
def get_category_specs(request, category_id):
    try:
        category = Category.objects.get(id=category_id)
        specs = ProductSpecificationTemplate.objects.filter(category=category)
        specs_data = []

        for spec in specs:
            spec_data = {
                'id': spec.id,
                'name': spec.name,
                'type': spec.field_type,
                'unit': spec.unit
            }

```

```

    }

    if spec.field_type == 'select':
        spec_data['options'] = spec.options if isinstance(spec.options,
list) else []
    elif spec.field_type == 'range':
        spec_data['min_value'] = spec.min_value
        spec_data['max_value'] = spec.max_value

    specs_data.append(spec_data)

    return JsonResponse({'specifications': specs_data})
except Category.DoesNotExist:
    return JsonResponse({'error': 'Категорія не знайдена'}, status=404)
except Exception as e:
    return JsonResponse({'error': str(e)}, status=500)

@user_passes_test(is_admin)
@require_POST
def add_spec_option(request, template_id):
    template = get_object_or_404(ProductSpecificationTemplate, id=template_id)
    data = json.loads(request.body)
    new_option = data.get('option')

    if not new_option:
        return JsonResponse({'error': 'Опція не може бути порожньою'}, status=400)

    current_options = template.options or []
    if new_option in current_options:
        return JsonResponse({'error': 'Така опція вже існує'}, status=400)

    current_options.append(new_option)
    template.options = current_options
    template.save()

    return JsonResponse({
        'success': True,
        'options': template.options
    })

@require_GET
def search_spec_options(request, template_id):
    template = get_object_or_404(ProductSpecificationTemplate, id=template_id)
    query = request.GET.get('q', '').lower()

    if not template.options:
        return JsonResponse({'options': []})

    filtered_options = [
        option for option in template.options
        if query in option.lower()
    ]

    return JsonResponse({'options': filtered_options})

```

Б.3 Текст файлу cart/views.py

```

from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth.decorators import login_required
from django.http import JsonResponse
from django.template.loader import render_to_string
from products.models import Product
from .models import CartItem, Cart
from django.db import transaction
from django.views.decorators.http import require_POST
import json

@login_required
def cart_detail(request):
    cart = request.user.cart
    cart.update_prices()
    cart_items = cart.items.all().select_related('product')
    total_price = cart.total_price

    has_stock_issues = any(
        item.product.stock == 0 or item.product.stock < item.quantity
        for item in cart_items
    )

    return render(request, 'cart/cart_detail.html', {
        'cart_items': cart_items,
        'total_price': total_price,
        'has_stock_issues': has_stock_issues
    })

def add_to_cart(request, product_id):
    if not request.user.is_authenticated:
        return JsonResponse({
            'status': 'error',
            'message': 'Будь ласка, увійдіть в систему або зареєструйтесь, щоб додати товар до кошика'
        })

    if request.method == 'POST':
        try:
            with transaction.atomic():
                product = get_object_or_404(Product, id=product_id)
                quantity = int(request.POST.get('quantity', 1))

                if product.stock < quantity:
                    return JsonResponse({
                        'status': 'error',
                        'message': f'На складі доступно лише {product.stock} шт.'
                    })

                cart = Cart.objects.get_or_create(user=request.user)[0]
                cart_item, created = CartItem.objects.get_or_create(
                    cart=cart,
                    product=product,
                    defaults={'quantity': quantity, 'price': product.price}
                )

```

```

    )

    if not created:
        new_quantity = cart_item.quantity + quantity
        if new_quantity > product.stock:
            return JsonResponse({
                'status': 'error',
                'message': f'На складі доступно лише {product.stock}
шт.'
            })
        cart_item.quantity = new_quantity
        cart_item.save()

    cart.update_total_price()

    return JsonResponse({
        'status': 'success',
        'message': 'Товар додано до кошика',
        'cart_total': cart.total_price
    })
except Exception as e:
    return JsonResponse({
        'status': 'error',
        'message': str(e)
    })

return JsonResponse({'status': 'error', 'message': 'Невірний метод запиту'})

@login_required
def remove_from_cart(request, item_id):
    if request.method == 'POST':
        try:
            cart = request.user.cart
            cart_item = get_object_or_404(CartItem, id=item_id, cart=cart)
            cart_item.delete()
            cart.update_total_price()

            return JsonResponse({
                'status': 'success',
                'message': 'Товар видалено з кошика',
                'cart_total': cart.total_price
            })
        except Exception as e:
            return JsonResponse({
                'status': 'error',
                'message': str(e)
            })

    return JsonResponse({'status': 'error', 'message': 'Невірний метод запиту'})

@login_required
def update_quantity(request, item_id):
    if request.method == 'POST':
        try:
            with transaction.atomic():

```

```

        cart_item = get_object_or_404(CartItem, id=item_id,
cart__user=request.user)
        data = json.loads(request.body)
        new_quantity = int(data.get('quantity', 1))

        if new_quantity > cart_item.product.stock:
            return JsonResponse({
                'status': 'error',
                'message': f'На складі доступно лише
{cart_item.product.stock} шт.'
            })

        cart_item.quantity = new_quantity
        cart_item.save()
        cart_item.cart.update_total_price()

        return JsonResponse({
            'status': 'success',
            'message': 'Кількість оновлено',
            'cart_total': cart_item.cart.total_price
        })
    except Exception as e:
        return JsonResponse({
            'status': 'error',
            'message': str(e)
        })

    return JsonResponse({'status': 'error', 'message': 'Невірний метод запиту'})

@login_required
def get_cart_items(request):
    cart = request.user.cart
    cart.update_prices()
    cart_items = cart.items.all().select_related(
        'product',
        'product__manufacturer',
        'product__category'
    ).prefetch_related('product__images')
    total_price = cart.total_price

    items_html = render_to_string('cart/mini_cart_items.html', {
        'cart_items': cart_items
    })

    return JsonResponse({
        'status': 'success',
        'items_html': items_html,
        'total_price': total_price,
        'cart_total': cart.items.count()
    })

@login_required
def cart_count(request):
    try:
        if request.user.is_authenticated:
            count = request.user.cart.items.count()

```



```

    if not user.last_name:
        user.last_name = address.last_name
    if not user.email:
        user.email = address.email
    user.save()

    if not profile.phone:
        profile.phone = address.phone
    if not profile.city:
        profile.city = address.city
    if not profile.postal_code:
        profile.postal_code = address.postal_code
    if not profile.address:
        profile.address = address.address
    profile.save()

    order = order_form.save(commit=False)
    order.user = user
    order.address = address
    order.cart = cart
    order.total_price = cart.total_price
    order.save()

    for cart_item in cart_items:
        OrderItem.objects.create(
            order=order,
            product=cart_item.product,
            quantity=cart_item.quantity,
            price=cart_item.product.price
        )

        product = cart_item.product
        product.stock -= cart_item.quantity
        product.save()

    cart.items.all().delete()
    cart.update_total_price()

    messages.success(request, 'Замовлення успішно оформлено!')
    return redirect('orders:order_detail', order_id=order.id)

except Exception as e:
    messages.error(request, f'Помилка при оформленні замовлення:
{str(e)}')
else:
    for field, errors in address_form.errors.items():
        for error in errors:
            messages.error(request, f'Помилка в полі
{address_form.fields[field].label}: {error}')
    for field, errors in order_form.errors.items():

```

```

        for error in errors:
            messages.error(request, f'Помилка в полі
{order_form.fields[field].label}: {error}')
        else:
            address_form = OrderAddressForm(user=request.user)
            order_form = OrderForm()

    return render(request, 'orders/checkout.html', {
        'address_form': address_form,
        'order_form': order_form,
        'cart_items': cart_items,
        'total': cart.total_price
    })

@login_required
def order_detail(request, order_id):
    order = get_object_or_404(Order, id=order_id, user=request.user)
    return render(request, 'orders/order_detail.html', {'order': order})

@login_required
def order_list(request):
    orders = Order.objects.filter(user=request.user).order_by('-created_at')
    return render(request, 'orders/order_list.html', {'orders': orders})

@login_required
@user_passes_test(lambda u: u.is_staff)
def admin_orders(request):

    orders = Order.objects.all().order_by('-created_at')
    return render(request, 'orders/admin_orders.html', {'orders': orders})

@login_required
@user_passes_test(lambda u: u.is_staff)
def update_order_status(request, order_id):
    if request.method == 'POST':
        try:
            data = json.loads(request.body)
            new_status = data.get('status')

            if new_status not in dict(Order.STATUS_CHOICES):
                return JsonResponse({
                    'status': 'error',
                    'message': 'Невірний статус замовлення'
                }, status=400)

            order = Order.objects.get(id=order_id)
            order.status = new_status
            order.save()

            return JsonResponse({
                'status': 'success',

```

```

        'message': 'Статус замовлення оновлено'
    })
except Order.DoesNotExist:
    return JsonResponse({
        'status': 'error',
        'message': 'Замовлення не знайдено'
    }, status=404)
except json.JSONDecodeError:
    return JsonResponse({
        'status': 'error',
        'message': 'Невірний формат даних'
    }, status=400)
return JsonResponse({
    'status': 'error',
    'message': 'Невірний метод запиту'
}, status=405)

```

```

@login_required
@user_passes_test(lambda u: u.is_staff)
def update_payment_status(request, order_id):
    if request.method == 'POST':
        try:
            order = Order.objects.get(id=order_id)
            data = json.loads(request.body)
            is_paid = data.get('is_paid')

            if isinstance(is_paid, bool):
                order.is_paid = is_paid
                order.save()
                return JsonResponse({'status': 'success'})
            else:
                return JsonResponse({
                    'status': 'error',
                    'message': 'Невірний формат даних'
                })
        except Order.DoesNotExist:
            return JsonResponse({
                'status': 'error',
                'message': 'Замовлення не знайдено'
            })
    return JsonResponse({
        'status': 'error',
        'message': 'Невірний метод запиту'
    })

```

```

@login_required
def cancel_order(request, order_id):
    try:
        order = Order.objects.get(id=order_id, user=request.user)
        if order.status != 'pending':
            return JsonResponse({
                'status': 'error',
                'message': 'Можна скасувати тільки замовлення в статусі "Очікує
обробки"'
            })

```

```

        order.status = 'cancelled'
        order.save()

        return JsonResponse({
            'status': 'success',
            'message': 'Замовлення успішно скасовано'
        })

    except Order.DoesNotExist:
        return JsonResponse({
            'status': 'error',
            'message': 'Замовлення не знайдено'
        })
    except Exception as e:
        return JsonResponse({
            'status': 'error',
            'message': str(e)
        })

```

Б.5 Текст файлу accounts\views.py

```

from django.contrib.auth.forms import AuthenticationForm
from django.contrib.auth import login
from django.contrib import messages
from django.shortcuts import render, redirect
from django.contrib.auth.decorators import login_required
from django.http import JsonResponse
from .forms import CustomUserCreationForm, ProfileEditForm
from orders.models import Order

def login_view(request):
    if request.method == 'POST':
        form = AuthenticationForm(request, data=request.POST)
        if form.is_valid():
            user = form.get_user()
            login(request, user)
            messages.success(request, 'Ви успішно увійшли!')
            return redirect('products:product_list')
        else:
            messages.error(request, 'Невірні дані, спробуйте ще раз.')
    else:
        form = AuthenticationForm()
    return render(request, 'accounts/login.html', {'form': form})

def register_view(request):
    if request.method == 'POST':
        form = CustomUserCreationForm(request.POST)
        if form.is_valid():
            user = form.save()
            login(request, user)
            messages.success(request, 'Ваш акаунт успішно створено!')
            return redirect('products:product_list')
    else:
        form = CustomUserCreationForm()

```

```

        return render(request, 'accounts/register.html', {'form': form})

def is_profile_complete(user):

    if not user.is_authenticated:
        return True

    if not (user.first_name and user.last_name and user.email):
        return False

    profile = user.profile
    if not (profile.phone and profile.city and profile.postal_code and
profile.address):
        return False

    return True

@login_required
def profile_view(request):
    user_orders = Order.objects.filter(user=request.user).order_by('-created_at')

    orders_count = user_orders.count()

    context = {
        'orders': user_orders,
        'orders_count': orders_count,
        'is_profile_complete': is_profile_complete(request.user)
    }

    return render(request, 'accounts/profile.html', context)

@login_required
def edit_profile_view(request):
    if request.method == 'POST':
        form = ProfileEditForm(request.POST, instance=request.user)
        if form.is_valid():
            try:
                user = form.save()
                if request.headers.get('X-Requested-With') == 'XMLHttpRequest':
                    return JsonResponse({
                        'status': 'success',
                        'message': 'Профіль успішно оновлено',
                        'data': {
                            'full_name': user.get_full_name() or user.username,
                            'email': user.email,
                            'phone': user.profile.phone if hasattr(user,
'profile') else ''
                        }
                    })
                messages.success(request, 'Профіль успішно оновлено!')
                return redirect('accounts:profile')
            except Exception as e:
                if request.headers.get('X-Requested-With') == 'XMLHttpRequest':
                    return JsonResponse({
                        'status': 'error',
                        'message': str(e)

```

```

    })
    messages.error(request, f'Помилка при збереженні: {str(e)}')
else:
    if request.headers.get('X-Requested-With') == 'XMLHttpRequest':
        return JsonResponse({
            'status': 'error',
            'message': 'Помилка валідації даних',
            'errors': form.errors
        })
    messages.error(request, 'Помилка при збереженні даних')
else:
    form = ProfileEditForm(instance=request.user)

return render(request, 'accounts/edit_profile.html', {'form': form})

```

Б.6 Текст файлу cart.js

```

document.addEventListener('DOMContentLoaded', function() {
    document.querySelectorAll('.add-to-cart').forEach(button => {
        button.addEventListener('click', async function(e) {
            e.preventDefault();
            const productId = this.dataset.productId;

            try {
                const response = await fetch(`/cart/add/${productId}/`, {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json',
                        'X-CSRFToken':
document.querySelector('[name=csrfmiddlewaretoken]').value
                    }
                });

                const data = await response.json();

                showNotification(data.message, data.status);

                if (data.status === 'success') {
                    updateCartCount(data.cart_total);

                    updateMiniCart();
                }
            } catch (error) {
                console.error('Error:', error);
                showNotification('Помилка при додаванні товару в кошик', 'error');
            }
        });
    });

    function showNotification(message, type = 'success') {
        const toastContainer = document.querySelector('.toast-container');
        const toast = document.createElement('div');
        toast.className = `toast align-items-center text-white bg-${type ===
'success' ? 'success' : 'danger'} border-0`;
        toast.setAttribute('role', 'alert');
        toast.setAttribute('aria-live', 'assertive');
    }

```

```

toast.setAttribute('aria-atomic', 'true');

toast.innerHTML = `
  <div class="d-flex">
    <div class="toast-body">
      <i class="fas fa-${type === 'success' ? 'check-circle' :
'exclamation-circle'} me-2"></i>
      ${message}
    </div>
    <button type="button" class="btn-close btn-close-white me-2 m-
auto" data-bs-dismiss="toast"></button>
  </div>
`;

toastContainer.appendChild(toast);
const bsToast = new bootstrap.Toast(toast, {
  animation: true,
  autohide: true,
  delay: 3000
});
bsToast.show();

toast.addEventListener('hidden.bs.toast', function() {
  toast.remove();
});
}

async function updateMiniCart() {
  try {
    const response = await fetch('/cart/get-items/');
    const data = await response.json();

    if (data.status === 'success') {
      const miniCartItems = document.querySelector('.mini-cart-items');
      if (miniCartItems) {
        miniCartItems.innerHTML = data.items_html;
        addRemoveItemHandlers();
      }
    }
  } catch (error) {
    console.error('Error updating mini cart:', error);
  }
}

function addRemoveItemHandlers() {
  document.querySelectorAll('.mini-cart-remove').forEach(button => {
    button.addEventListener('click', async function(e) {
      e.preventDefault();
      e.stopPropagation();
      const itemId = this.dataset.itemId;

      try {
        const response = await fetch(`/cart/remove/${itemId}/`, {
          method: 'POST',
          headers: {

```

```

        'Content-Type': 'application/json',
        'X-CSRFToken':
document.querySelector('[name=csrfmiddlewaretoken]').value
    }
    });

    const data = await response.json();

    showNotification(data.message, data.status);

    if (data.status === 'success') {
        updateCartCount(data.cart_total);

        updateMiniCart();
    }
    } catch (error) {
        console.error('Error:', error);
        showNotification('Помилка при видаленні товару з кошика',
'error');
    }
    });
});

document.querySelectorAll('.mini-cart-item').forEach(item => {
    item.addEventListener('click', function(e) {
        if (!e.target.closest('.mini-cart-remove')) {
            const productId = this.dataset.productId;
            const productSlug = this.dataset.productSlug;
            window.location.href =
`/products/${productId}/${productSlug}/`;
        }
    });
});

addRemoveItemHandlers();
updateMiniCart();

function updateCartCount() {
    const cartCount = document.getElementById('cart-count');
    if (cartCount) {
        const csrfToken =
document.querySelector('[name=csrfmiddlewaretoken]').value;
        fetch('/cart/count/', {
            method: 'GET',
            headers: {
                'X-CSRFToken': csrfToken,
                'Accept': 'application/json'
            }
        })
        .then(response => {
            if (!response.ok) {
                throw new Error(`HTTP error! status: ${response.status}`);
            }
            return response.json();
        })
    }
}

```

```

        .then(data => {
            cartCount.textContent = data.count;
        })
        .catch(error => {
            console.error('Error updating cart count:', error);
            cartCount.textContent = '0';
        });
    }
}

updateCartCount();

async function updateQuantity(itemId, quantity) {
    try {
        const response = await fetch(`/cart/update/${itemId}/`, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'X-CSRFToken':
document.querySelector('[name=csrfmiddlewaretoken]').value
            },
            body: JSON.stringify({ quantity: quantity })
        });

        const data = await response.json();

        if (data.status === 'success') {
            updateCartCount();
            updateMiniCart();
            if (window.location.pathname === '/cart/') {
                window.location.reload();
            }
        } else {
            showNotification(data.message, 'error');
        }
    } catch (error) {
        console.error('Error:', error);
        showNotification(Помилка при оновлені кількості, 'error');
    }
}

async function removeItem(itemId) {
    try {
        const response = await fetch(`/cart/remove/${itemId}/`, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'X-CSRFToken':
document.querySelector('[name=csrfmiddlewaretoken]').value
            },
        });

        const data = await response.json();

        if (data.status === 'success') {

```

```

        updateCartCount();
        updateMiniCart();
        if (window.location.pathname === '/cart/') {
            window.location.reload();
        }
    } else {
        showNotification(data.message, 'error');
    }
} catch (error) {
    console.error('Error:', error);
    showNotification('Ошибка при удалении товара', 'error');
}
}

document.addEventListener('DOMContentLoaded', function() {
    document.querySelectorAll('.quantity-btn').forEach(button => {
        button.addEventListener('click', function() {
            const form = this.closest('.quantity-form');
            const input = form.querySelector('.quantity-input');
            const itemId = form.dataset.itemId;
            let quantity = parseInt(input.value);

            if (this.classList.contains('plus')) {
                quantity++;
            } else if (this.classList.contains('minus') && quantity > 1) {
                quantity--;
            }

            updateQuantity(itemId, quantity);
        });
    });

    document.querySelectorAll('.quantity-input').forEach(input => {
        input.addEventListener('change', function() {
            const form = this.closest('.quantity-form');
            const itemId = form.dataset.itemId;
            let quantity = parseInt(this.value);

            if (quantity < 1) {
                quantity = 1;
                this.value = 1;
            }

            updateQuantity(itemId, quantity);
        });
    });

    document.querySelectorAll('.remove-btn').forEach(button => {
        button.addEventListener('click', function(e) {
            e.preventDefault();
            const itemId = this.dataset.itemId;
            removeItem(itemId);
        });
    });
});
});
});

```

ДОДАТОК В
Слайди презентації

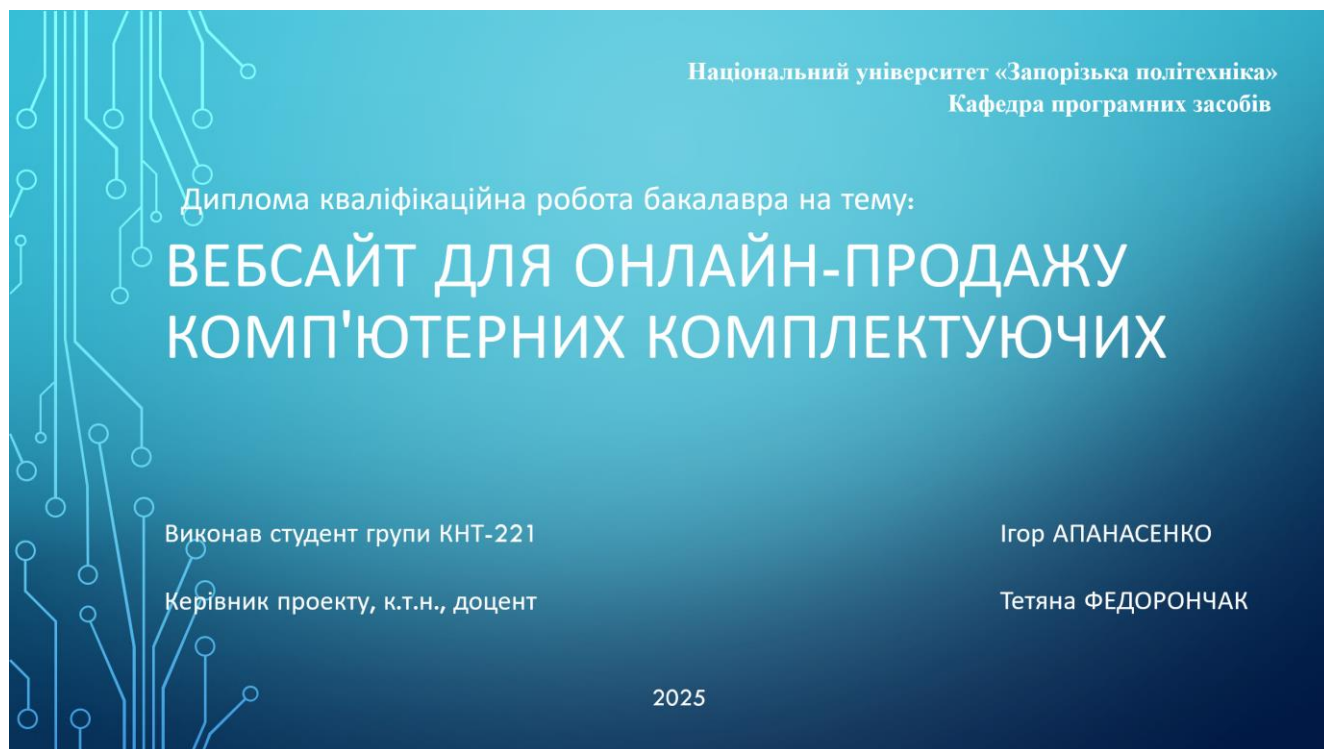


Рисунок В.1 – Слайд 1

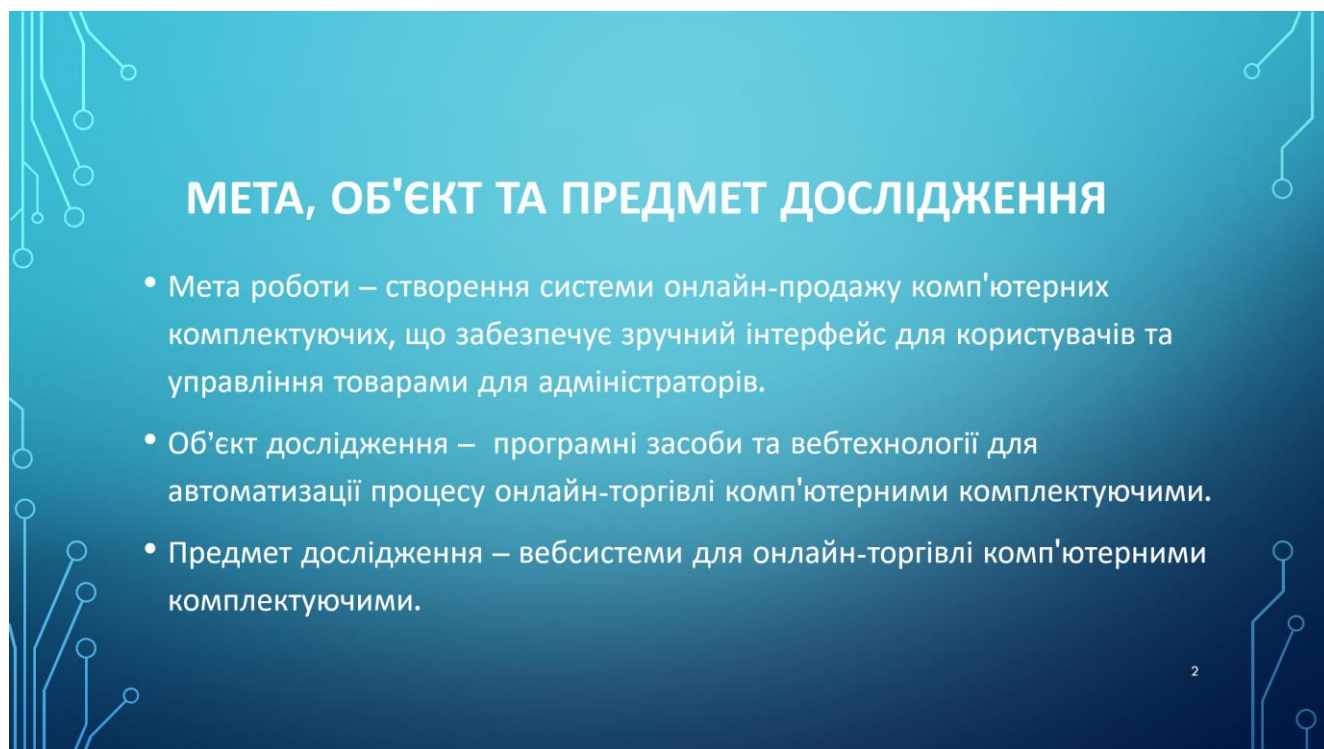


Рисунок В.2 – Слайд 2

ЗАВДАННЯ РОБОТИ

1. Провести аналіз існуючих рішень та визначити оптимальні технології для розробки веб-додатку;
2. Розробити архітектуру системи та спроектувати базу даних;
3. Реалізувати функціонал каталогу товарів з можливістю фільтрації та пошуку;
4. Створити систему управління кошиком та оформлення замовлень;
5. Провести комплексне тестування системи та перевірити її працездатність.

3

Рисунок В.3 – Слайд 3

Порівняльна таблиця існуючих аналогів

Критерій	compbest.com.ua	neon.ua	itplanet.zp.ua	Власний вебсайт
Адаптація для мобільних пристроїв	±	-	-	+
Сучасний дизайн	±	-	±	+
Зручний профіль користувача	±	-	-	+
Функціональна система фільтрації товарів	+	±	-	+
Структуроване відображення інформації про товар	+	-	+	+
Оптимізований процес роботи з кошиком та оформлення замовлення	±	-	-	+

4

Рисунок В.4 – Слайд 4

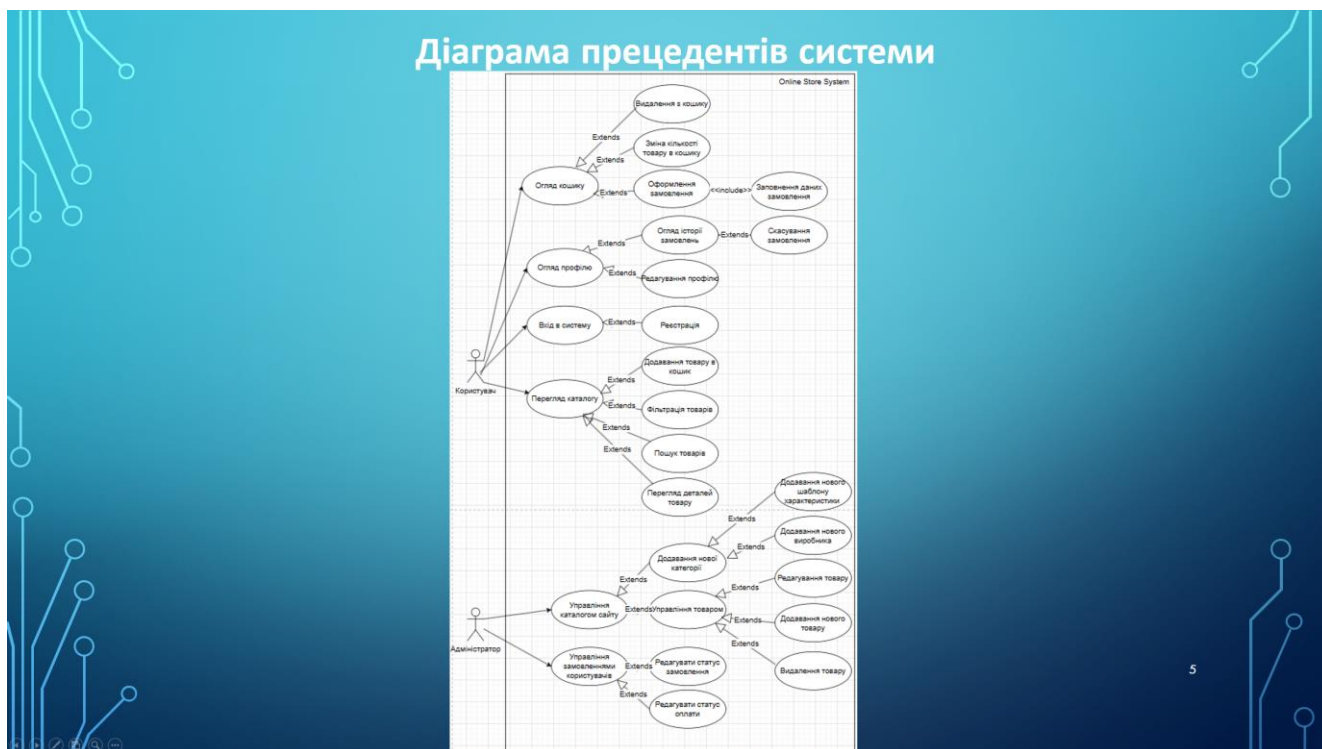


Рисунок В.5 – Слайд 5

Порівняльний аналіз фреймворків для серверної частини					
Критерій	Django	Node.js	Ruby on Rails	Laravel	Spring
Мова програмування	Python – проста, популярна, універсальна	JavaScript – універсальна, популярна	Ruby – менш поширена, але зручна	PHP – поширена, але менш сучасна	Java – складніша, але потужна
Швидкість розробки	Висока: багато готових модулів, генераторів, DRY-принцип	Висока: багато npm-пакетів, але часто треба писати більше вручну	Висока: генератори scaffold, «конвенція понад конфігурацію»	Висока: багато пакетів, але часто потребує налаштувань	Середня: багато шаблонного коду, складна конфігурація
Безпека	Вбудовані захисти від XSS, CSRF, SQLi, clickjacking	Потрібно додавати вручну (helmet, csrf тощо)	Вбудовані захисти, але менше оновлень	Є базові захисти, але не всі за замовчуванням	Високий рівень безпеки, але складна конфігурація
ORM	Вбудований потужний ORM, підтримка міграцій	Є ORM (Sequelize, TypeORM), але не вбудований	Вбудований ActiveRecord, простий у використанні	Вбудований Eloquent ORM, зручний	Вбудований Hibernate, але складний
REST API	Django REST Framework – стандарт де-факто, гнучкий, зручний	Express дозволяє швидко створювати API, але багато ручної роботи	Є gems для API, але не так зручно	Є пакети для API, але не так гнучко	Spring Boot дозволяє створювати API, але складніше
Масштабованість	Добре масштабується, використовується у великих проєктах (Instagram, Pinterest)	Добре масштабується, підходить для мікросервісів	Підходить для середніх проєктів	Підходить для середніх проєктів	Висока масштабованість, підходить для великих enterprise-рішень
Документація	Дуже детальна, багато прикладів, велика спільнота	Багато прикладів, велика спільнота	Хороша документація, але менша спільнота	Хороша документація, велика спільнота	Дуже детальна, але складна
Легкість навчання	Дуже проста для початківців, багато навчальних матеріалів	Проста для тих, хто знає JS, але є нюанси з асинхронністю	Відносно проста, але Ruby менш популярна	Середня, PHP знайомий багатьом, але сучасний Laravel має свої особливості	Складна для початківців, вимагає знань Java та Spring-екосистеми

6

Рисунок В.6 – Слайд 6

Порівняльний аналіз фреймворків для клієнтської частини

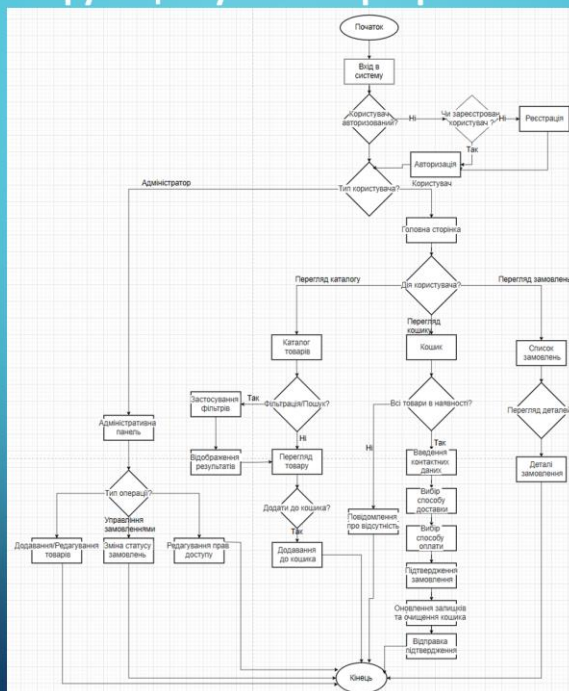
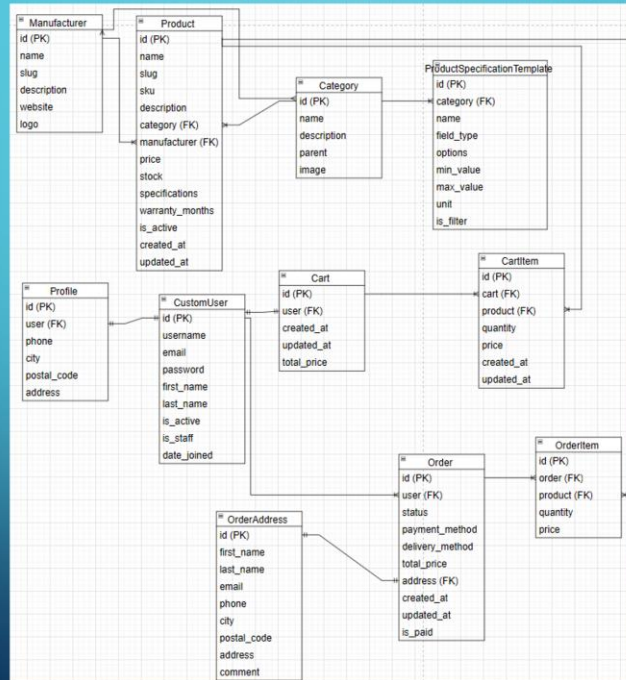
Критерій	HTML+CSS+JS+Bootstrap(наш вибір)	React/Vue/Angular	Переваги для нашого проєкту
Швидкість розробки	Висока: наявність великої кількості готових компонентів, простота впровадження	Висока для складних SPA, але потребує додаткової конфігурації	Bootstrap дозволяє швидко реалізувати сучасний інтерфейс
Адаптивність	Забезпечується вбудованими можливостями Bootstrap, не потребує додаткових налаштувань	Потребує підключення окремих бібліотек для адаптивності	Сайт коректно відображається на всіх пристроях
Інтеграція з Django	Пряма інтеграція через шаблони Django, мінімальні складнощі при обміні даними	Потребує створення окремого API, налаштування CORS	Зменшується кількість коду, спрощується підтримка
Продуктивність	Висока для класичних сайтів, сторінки швидко завантажуються	Висока, але для SPA потрібна додаткова оптимізація	Користувачі отримують швидкий доступ до контенту
Легкість підтримки	Висока для невеликих і середніх проєктів, легко знайти розробників	Потрібні знання сучасних фреймворків та JavaScript	Спрощується залучення нових учасників до проєкту
Готові рішення	Bootstrap містить широкий набір компонентів: модальні вікна, форми, меню, повідомлення, пагінація	Для SPA потрібне підключення сторонніх бібліотек	Можна швидко реалізувати будь-який елемент інтерфейсу
Можливість розширення	Дозволяє поступово додавати JavaScript-компоненти за потреби	SPA вимагає повної архітектури з самого початку	Гнучкість: можна почати з простого рішення, поступово ускладнювати функціонал
Вартість розробки	Менше часу на розробку, менше помилок, нижча вартість	Більше часу на налаштування, тестування	Важливо для студентського або малого проєкту
Документація/Спільнота	Велика кількість матеріалів, Bootstrap є стандартом у галузі	Велика спільнота, але більше орієнтована на складні SPA	Легко знайти відповіді на питання, швидко вирішувати проблеми

Рисунок В.7 – Слайд 7

Порівняльний аналіз реляційних та нереляційних систем керування базами даних

Критерій	Реляційні БД (SQL)	Нереляційні БД (NoSQL)
Структура даних	Таблична, чітко визначені схеми	Гнучка, динамічна, документи / ключ-значення / граfi
Мова запитів	Стандартизована (SQL)	Залежить від конкретної СКБД, часто власна
Цілісність даних	Висока, підтримка транзакцій (ACID)	Зазвичай нижча, часто eventual consistency
Масштабованість	Вертикальна (масштабування серверу)	Горизонтальна (розподілені системи)
Продуктивність	Висока для складних запитів, аналітики	Висока для великих обсягів неструктурованих даних
Гнучкість структури	Менш гнучка, зміни схеми потребують міграцій	Дуже гнучка, можна зберігати різні типи даних
Приклади	PostgreSQL, MySQL, Oracle, SQLite	MongoDB, Cassandra, Redis, CouchDB
Типові задачі	Фінанси, e-commerce, CRM, ERP	Big Data, IoT, соціальні мережі, кешування

Рисунок В.8 – Слайд 8



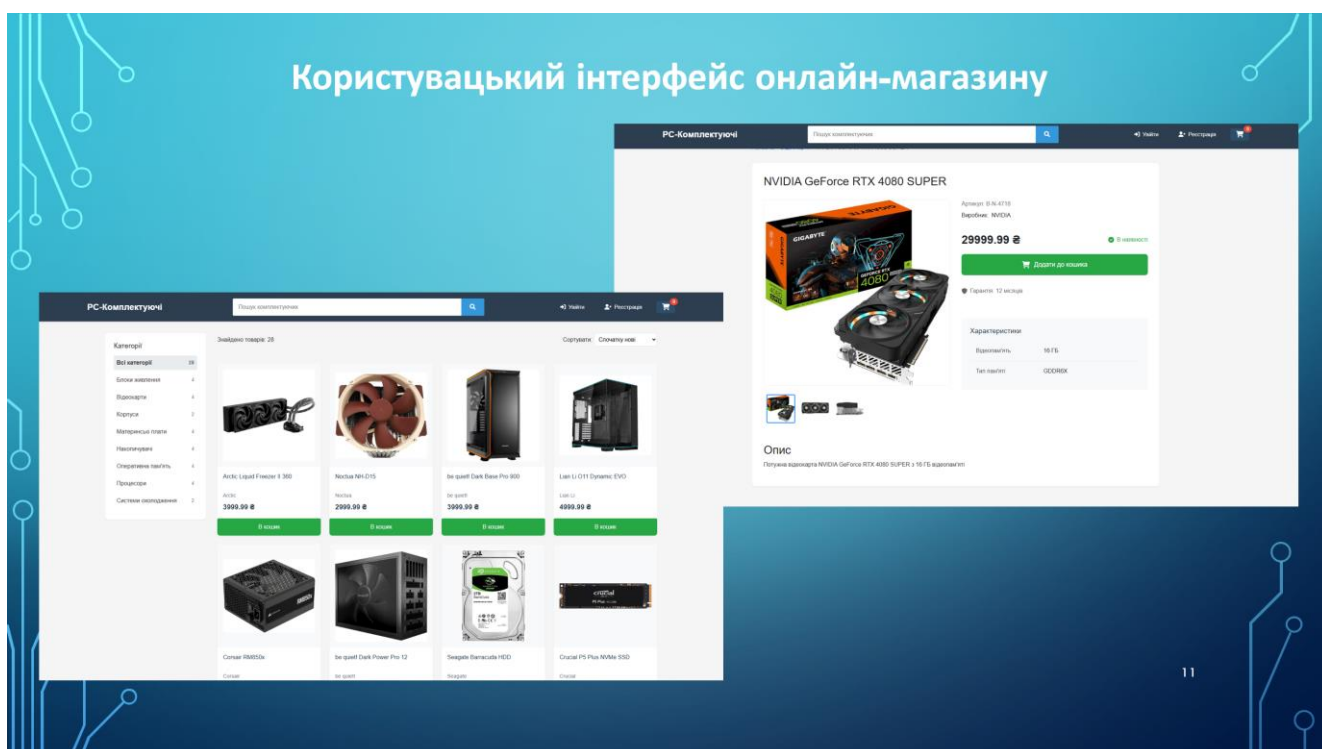


Рисунок В.11 – Слайд 11

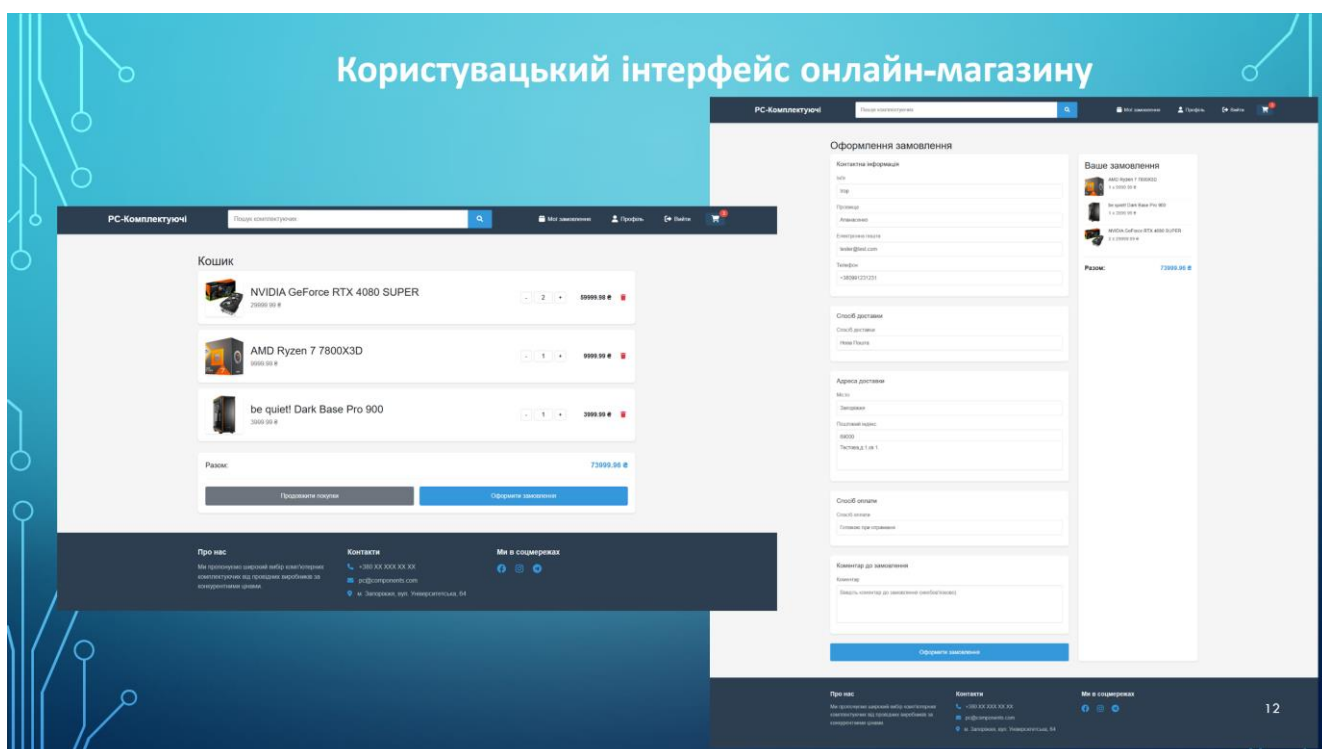


Рисунок В.12 – Слайд 12

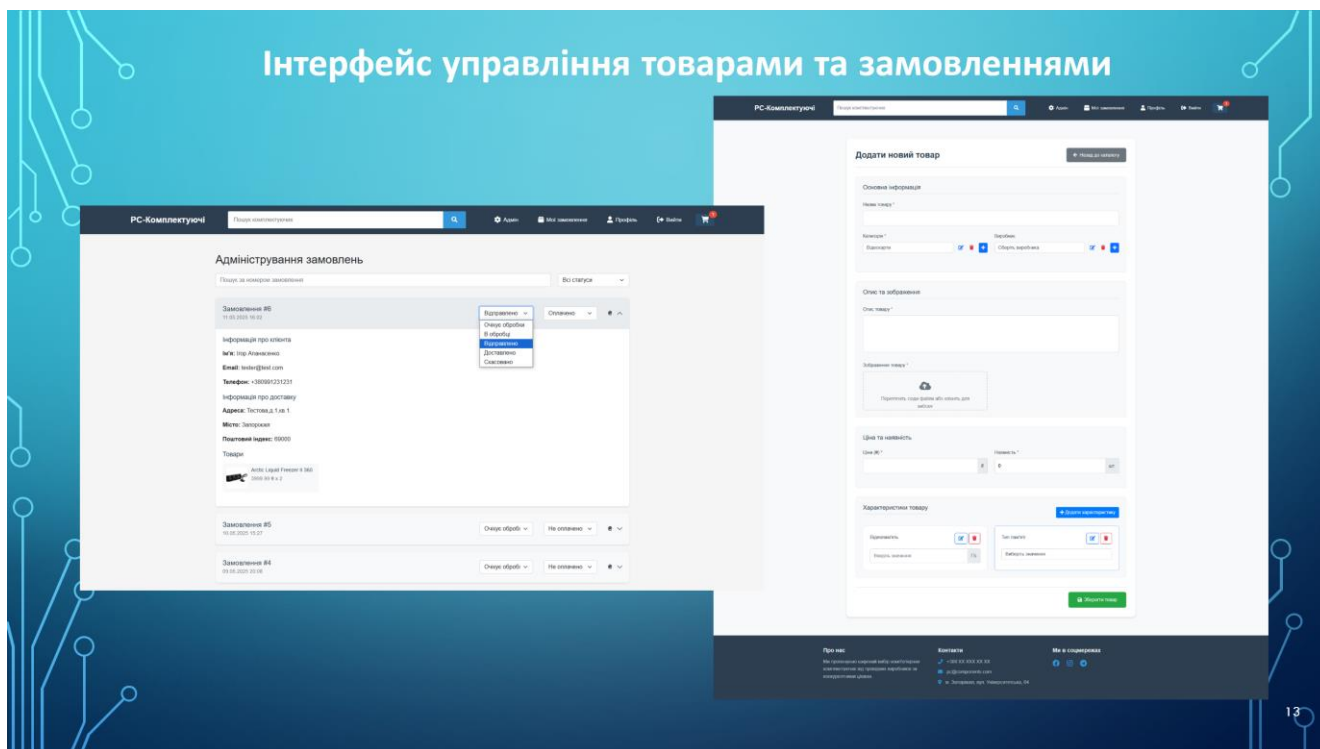


Рисунок В.13 – Слайд 13

Результати тестування інтерфейсу та функціональності				
Елемент інтерфейсу	Тестовий сценарій	Результат	Примітки	
Головне меню	Відображення на різних розмірах екрану	Успішно	Адаптивне меню на мобільних пристроях	
	Навігація по пунктах меню	Успішно	Коректні переходи на всі сторінки	
	Іконки та підписи	Успішно	Чітке відображення всіх елементів	
Пошукова строка	Адаптивність на мобільних пристроях	Успішно	Повноекранний пошук на малих екранах	
	Відображення результатів	Успішно	Зручне форматування результатів	
Кошик	Міні-кошик при наведенні	Успішно	Плавна анімація появи	
	Відображення кількості товарів	Успішно	Оновлення в реальному часі	
	Перехід до повного кошика	Успішно	Коректні посилання	
Каталог товарів	Відображення сітки товарів	Успішно	Адаптивна сітка	
	Фільтри та сортування	Успішно	Зручний інтерфейс вибору	
	Пагінація	Успішно	Коректна робота на всіх сторінках	
Картки товарів	Відображення фото	Успішно	Оптимізовані зображення	
	Інформація про товар	Успішно	Чітка структура даних	
Форми	Кнопки дій	Успішно	Зручне розташування	
	Валідація полів	Успішно	Миттєвий зворотній зв'язок	
	Адаптивність форм	Успішно	Коректне відображення на всіх пристроях	
Сповіщення	Повідомлення про помилки	Успішно	Зрозумілі повідомлення	
	Системні повідомлення	Успішно	Анімовані сповіщення	
	Помилки валідації	Успішно	Чітке відображення помилок	

Модуль	Функціональність	Тестовий сценарій	Результат	Примітки
Модуль облікових записів (accounts)	Реєстрація	Створення нового акаунту	Успішно	Перевірка валідації email, паролю
	Вхід в систему	Авторизація існуючого користувача	Успішно	Перевірка валідації логіну та паролю
	Профіль користувача	Перегляд та редагування профілю	Успішно	Перевірка оновлення контактних даних, адреси
Модуль товарів (products)	Перевірка профілю	Валідація повноти профілю	Успішно	Перевірка наявності всіх обов'язкових полів
	Каталог товарів	Перегляд списку товарів	Успішно	Перевірка пагінації, фільтрації, сортування
	Пошук товарів	Пошук за назвою, виробником, описом	Успішно	Перевірка пошуку за різними параметрами
	Фільтрація	Фільтрація за категорією, виробником, ціною	Успішно	Перевірка роботи всіх фільтрів
	Деталі товару	Перегляд інформації про товар	Успішно	Перевірка відображення всіх характеристик, фото
	Адміністрування	Додавання нового товару	Успішно	Перевірка завантаження фото, валідації даних
		Редагування товару	Успішно	Перевірка оновлення всіх полів
		Видалення товару	Успішно	Перевірка видалення з бази даних
Модуль кошика (cart)	Додавання товару	Додавання товару до кошика	Успішно	Перевірка контролю кількості, наявності на складі
	Оновлення кількості	Зміна кількості товару	Успішно	Перевірка обмежень по наявності
	Видалення товару	Видалення товару з кошика	Успішно	Перевірка оновлення загальної суми
Модуль замовлень (orders)	Перегляд кошика	Відображення товарів у кошику	Успішно	Перевірка відображення цін, кількості
	Оформлення замовлення	Створення нового замовлення	Успішно	Перевірка валідації номеру, зміни способу доставки
	Перегляд замовлення	Деталі конкретного замовлення	Успішно	Перевірка відображення всіх даних
	Список замовлень	Перегляд історії замовлень	Успішно	Перевірка сортування, фільтрації
	Адміністрування	Зміна статусу замовлення	Успішно	Перевірка всіх можливих статусів
		Відмітка про оплату	Успішно	Перевірка оновлення статусу оплати

Рисунок В.14 – Слайд 14

ВИСНОВКИ

У результаті виконання дипломної роботи було досягнуто поставлену мету — створено сучасну Систему онлайн-продажу комп'ютерних комплектуючих, яка забезпечує зручний інтерфейс для користувачів і ефективне управління товарами та замовленнями для адміністраторів.

У процесі розробки проведено детальний аналіз існуючих рішень у сфері електронної комерції, що дозволило визначити основні недоліки конкурентів і сформулювати вимоги до нового веб-додатку. На основі цього обґрунтовано вибір технологічного стеку: для серверної частини використано Django, що забезпечило високу швидкість розробки, масштабованість, а для клієнтської частини — HTML, CSS, JavaScript і Bootstrap, що дозволило реалізувати сучасний адаптивний інтерфейс.

Розроблена модульна архітектура системи з чітким розділенням функціоналу на окремі компоненти (`accounts`, `products`, `cart`, `orders`) спростила підтримку та подальший розвиток проєкту. Створено повноцінний каталог товарів із розширеною системою фільтрації та пошуку, реалізовано зручну роботу з кошиком і оформленням замовлень, а також адміністративну панель для управління асортиментом і обробки замовлень.

Комплексне тестування підтвердило стабільну роботу всіх функціональних модулів, високу продуктивність і адаптивність інтерфейсу для різних пристроїв. Система готова до впровадження в реальних умовах експлуатації, має значний потенціал для подальшого розвитку — зокрема, інтеграції з платіжними сервісами, впровадження системи рекомендацій, розширення API та додавання нових функцій.

Розроблений інтернет-магазин може бути використаний як окремими продавцями, так і компаніями, що спеціалізуються на продажі комп'ютерних компонентів, і повністю відповідає сучасним вимогам до електронної комерції.

Рисунок В.15 – Слайд 15