МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РАЗРОБКА ІНТЕРНЕТ-МАГАЗИНУ З МОДУЛЯМИ УПРАВЛІННЯ ТОВАРАМИ ТА ОСОБИСТИМ КАБІНЕТОМ ПОКУПЦЯ»**

Виконав: студент 4 курсу, групи 6.1222
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
М.С. Слободяник
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, доцент, к.т.н. Борю С. Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри програмної інженерії, доцент, к.ф.-м.н., Лісняк А. О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри комп'ютерних наук, к.пед.н., доцент

_____ Пшенична О.С.
(підпис)

« ____ « _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУСТУДЕНТЦІ СТУДЕНТОВІ

Слободянику Михайлу Сергійовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Разробка Інтернет-магазину з модулями управління товарами та особистим кабінетом покупця

керівник роботи Борю Сергій Юрійович, к.т.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 « грудня 2025 року № 1741-с

2. Строк подання студентом роботи 04.06.2026

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка інформаційної системи для онлайн реалізації товарів з особистим кабінетом користувача.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 05.12.2025

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	07.01.2026	
2.	Збір вихідних даних.	10.01.2026	
3.	Обробка методичних та теоретичних	15.01.2026	
	джерел.		
4.	Розробка першого та другого розділу.	23.01.2026	
5.	Розробка третього розділу.	01.02.2026	
6.	Оформлення та нормоконтроль	10.06.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	20.06.2026	

Студент _____ Слободяник М.С.
(підпис) (ініціали та прізвище)

Керівник роботи _____ Борю С.Ю.
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка інформаційної системи для онлайн реалізації товарів з особистим кабінетом користувача»: 122 с., 46 рис., 15 табл., 18 джерел, 4 додатків.

БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, ЕЛЕКТРОННА КОМЕРЦІЯ, ІНТЕРНЕТ-МАГАЗИН, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС, КОШИК, ОСОБИСТИЙ КАБІНЕТ, СИСТЕМА АВТОРИЗАЦІЇ, УПРАВЛІННЯ ТОВАРАМИ, CRUD-ОПЕРАЦІЇ, REST API.

Об’єкт дослідження – інформаційні системи електронної комерції.

Предмет дослідження – розробка вебзастосунку інтернет-магазину з особистим кабінетом користувача.

Мета роботи – розробити інформаційну систему інтернет-магазину з каталогом товарів, кошиком та особистим кабінетом користувача.

Методи дослідження – аналіз наукових джерел і існуючих систем електронної комерції, методи системного аналізу, проєктування програмних систем, методи об’єктно-орієнтованого програмування, методи розробки вебзастосунків, тестування програмного забезпечення (unit та end-to-end тестування), а також методи моделювання архітектури інформаційних систем.

У кваліфікаційній роботі проведено аналіз сучасних інформаційних систем електронної комерції та визначено їх основні функціональні можливості. Спроєктовано архітектуру інтернет-магазину та структуру бази даних, що забезпечує зберігання та обробку даних користувачів, товарів і замовлень. Реалізовано серверну та клієнтську частини системи, включаючи модулі авторизації, управління товарами, кошиком і особистим кабінетом користувача. Проведено тестування основних функцій системи, яке підтвердило її працездатність і коректність роботи всіх ключових модулів.

SUMMARY

Bachelor's Qualifying Thesis «Development of an Information System for Online Product Sales with a User Personal Account»: 122 pages, 46 figures, 15 tables, 18 sources, 4 supplements.

DATABASE, WEB APPLICATION, E-COMMERCE, ONLINE STORE, USER INTERFACE, SHOPPING CART, PERSONAL ACCOUNT, AUTHENTICATION SYSTEM, PRODUCT MANAGEMENT, CRUD OPERATIONS, REST API.

Object of research – e-commerce information systems.

Subject of research – development of an online store web application with a user personal account.

Purpose of the work – to develop an online store information system with a product catalog, shopping cart, and user personal account.

Research methods – analysis of scientific sources and existing e-commerce systems, methods of system analysis, software system design, object-oriented programming methods, web application development methods, software testing methods (unit and end-to-end testing), as well as methods of information system architecture modeling.

The qualification work analyzes modern e-commerce information systems and identifies their main functional capabilities. The architecture of the online store and the database structure for storing and processing user, product, and order data were designed. The server-side and client-side parts of the system were implemented, including modules for authentication, product management, shopping cart functionality, and user personal account management. Testing of the main system functions was carried out, confirming the operability and correctness of all key modules.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ 8
1 Особливості інформаційних систем в онлайн торгівлі 10
1.1 Роль інформаційної системи в онлайн торгівлі. Особливості використання інформаційних систем в онлайн торгівлі 10
1.2 Переваги ведення онлайн торгівлі за допомогою інформаційних систем…. 12
1.3 Основні цілі та задачі, які вирішуються особистим кабінетом користувача 13
1.4 Огляд існуючих інформаційних систем в торгівлі 14
1.5 Мета роботи 22
1.6 Висновки 22
2 Розробка інформаційної системи для онлайн реалізації товарів з особистим кабінетом користувача 24
2.1 Аналіз вимог до системи інтернет-магазину 24
2.2 Проєктування архітектури вебзастосунку 27
2.3 Проєктування бази даних для управління товарами та користувачами.. 37
2.4 Розробка функціоналу управління товарами (CRUD-операції) 42
2.5 Реалізація особистого кабінету покупця 51
2.6 Обґрунтування вибору технологій розробки 59
2.7 Висновки 60
3 Реалізація та тестування інтернет-магазину 61
3.1 Реалізація серверної частини системи 61
3.2 Розробка клієнтського інтерфейсу 66

3.3 Інтеграція модулів управління товарами та користувацького кабінету… 70
3.4 Тестування функціональності вебсистеми 76
3.5 Інструкція користувача 80
3.6 Висновки 82
Висновки 83
Перелік посилань 85
Додаток А Реалізація серверної логіки та API-модуля e-commerce застосунку 87
Додаток Б Реалізація клієнтської частини та React-компонентів інтерфейсу 100
Додаток В Приклад синхронізації з SWR 114
Додаток Г Вигляд розробленої інформаційної системи 116

ВСТУП

Розвиток інформаційних технологій та стрімка цифровізація економічних процесів зумовлюють суттєві трансформації у сфері торгівлі. Сучасні інформаційні системи електронної комерції виступають ключовим інструментом організації продажів, взаємодії з клієнтами та автоматизації бізнес-процесів. Інтернет-магазини сьогодні є складними веборієнтованими системами, які поєднують у собі функції управління товарами, обробки замовлень, аналітики та персоналізації користувацького досвіду.

Актуальність теми дипломної роботи зумовлена необхідністю розробки ефективних, масштабованих та зручних у використанні інформаційних систем для онлайн-торгівлі, які забезпечують автоматизацію ключових процесів взаємодії між продавцем і покупцем. Особливо важливим є створення функціонального особистого кабінету користувача, який дозволяє здійснювати управління замовленнями, перегляд історії покупок та персоналізацію взаємодії із системою. Аналіз сучасних платформ електронної комерції, зокрема Rozetka, показує, що ефективність таких систем значною мірою залежить від якості реалізації користувацького інтерфейсу та внутрішньої архітектури.

Метою роботи є розробка інтернет-магазину з модулями управління товарами та особистим кабінетом покупця, що забезпечує зручну взаємодію користувачів із системою та автоматизацію процесів електронної торгівлі.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати сучасний стан розвитку інформаційних систем електронної комерції;
- дослідити функціональні можливості існуючих інтернет-магазинів та систем особистих кабінетів користувачів;
- визначити вимоги до розроблюваної інформаційної системи;
- спроєктувати архітектуру інтернет-магазину та базу даних;

- розробити та протестувати модулі управління товарами й особистий кабінет користувача.

Об’єктом дослідження є процеси функціонування інформаційних систем електронної комерції.

Предметом дослідження є методи та засоби розробки інтернет-магазину з модулями управління товарами та особистим кабінетом користувача.

У процесі дослідження було проаналізовано сучасні підходи до побудови веборієнтованих інформаційних систем, визначено принципи проєктування інтернет-магазинів, а також розглянуто приклади реалізації подібних систем у провідних цифрових платформах, зокрема в екосистемах електронної комерції та цифрової дистрибуції, таких як Xbox.

Наукова новизна полягає у формалізації підходу до проєктування інтернет-магазину як єдиної інформаційної системи, що об’єднує модулі управління товарами та особистого кабінету користувача, з визначенням їх взаємозв’язків і функціональної структури.

Практичне значення результатів роботи полягає у можливості використання розробленої системи як готового рішення для малого та середнього бізнесу у сфері електронної комерції, а також як бази для подальшого розширення функціональності вебзастосунків (інтеграція платіжних систем, систем рекомендацій, аналітики тощо).

1 ОСОБЛИВОСТІ ІНФОРМАЦІЙНИХ СИСТЕМ В ОНЛАЙН ТОРГІВЛІ

1.1 Роль інформаційної системи в онлайн торгівлі. Особливості використання інформаційних систем в онлайн торгівлі

Інформаційні системи відіграють ключову роль у функціонуванні сучасної онлайн-торгівлі, оскільки забезпечують комплексну автоматизацію бізнес-процесів, пов’язаних із продажем товарів і взаємодією з клієнтами. У загальному розумінні інформаційна система являє собою сукупність взаємопов’язаних програмних, технічних, інформаційних та організаційних компонентів, які забезпечують збирання, обробку, зберігання та передачу даних для підтримки управлінських рішень і виконання операційної діяльності підприємства [1].

В умовах електронної комерції інформаційна система виступає центральним елементом цифрової інфраструктури підприємства, оскільки саме вона забезпечує взаємодію між усіма учасниками торговельного процесу: продавцем, покупцем, постачальниками та логістичними сервісами. На відміну від традиційної торгівлі, де значна частина процесів виконується вручну, онлайн-торгівля повністю базується на автоматизованих інформаційних процесах, що дозволяє суттєво підвищити швидкість обробки операцій та зменшити вплив людського фактору.

Роль інформаційної системи в онлайн-торгівлі полягає насамперед у забезпеченні безперервного функціонування електронної торговельної платформи. Це включає організацію доступу до каталогу товарів, обробку замовлень, управління складськими залишками, підтримку платіжних операцій, а також взаємодію з клієнтами через особисті кабінети. Крім того, сучасні інформаційні системи виконують аналітичну функцію, збираючи та обробляючи дані про поведінку користувачів з метою формування персоналізованих пропозицій та оптимізації маркетингових стратегій [2].

Важливою складовою інформаційних систем у сфері онлайн-торгівлі є їх інтеграційна роль. Сучасні електронні магазини рідко функціонують як ізольовані системи; зазвичай вони інтегруються з зовнішніми сервісами, такими як платіжні шлюзи, системи доставки, CRM та ERP-рішення. Така інтеграція дозволяє забезпечити наскрізну автоматизацію бізнес-процесів – від моменту оформлення замовлення до його доставки кінцевому споживачу.

Особливістю використання інформаційних систем в онлайн-торгівлі є їх орієнтація на безперервну роботу та високу доступність. Електронні торговельні платформи повинні функціонувати у режимі 24/7, забезпечуючи стабільний доступ користувачів до сервісів незалежно від часу доби та географічного розташування. Це висуває підвищені вимоги до архітектури системи, зокрема щодо масштабованості, відмовостійкості та продуктивності [3].

Ще однією суттєвою особливістю є домінування користувацького інтерфейсу як основного засобу взаємодії з системою. В онлайн-торгівлі саме інтерфейс визначає якість користувацького досвіду, тому значна увага приділяється зручності навігації, швидкості завантаження сторінок, адаптивності дизайну та простоті виконання ключових операцій, таких як пошук товарів або оформлення замовлення. Окрему роль відіграє персоналізація, яка реалізується на основі аналізу великих обсягів даних про користувачів. Інформаційні системи в електронній комерції активно застосовують алгоритми аналізу поведінки клієнтів для формування індивідуальних рекомендацій, таргетованої реклами та оптимізації асортименту товарів. Це дозволяє підвищити ефективність продажів і рівень задоволеності користувачів.

Крім того, важливою характеристикою є підвищені вимоги до інформаційної безпеки. Оскільки інформаційні системи онлайн-торгівлі обробляють персональні дані користувачів і платіжну інформацію, вони повинні забезпечувати захист від несанкціонованого доступу, витоку даних та інших кіберзагроз. Це вимагає впровадження сучасних механізмів шифрування, автентифікації та контролю доступу [2].

Отже, інформаційні системи в онлайн-торгівлі виконують багатофункціональну роль, поєднуючи операційні, аналітичні та інтеграційні

функції. Їх ефективне використання забезпечує автоматизацію бізнес-процесів, підвищення якості обслуговування клієнтів та загальну конкурентоспроможність електронних торговельних платформ у сучасних умовах цифрової економіки.

1.2 Переваги ведення онлайн торгівлі за допомогою інформаційних систем

Використання інформаційних систем в онлайн-торгівлі забезпечує значне підвищення ефективності бізнес-процесів та створює низку конкурентних переваг для підприємств. Сучасні електронні торговельні платформи базуються на інтегрованих програмних рішеннях, які автоматизують ключові операції, пов’язані з продажем товарів, обробкою замовлень, обліком запасів та взаємодією з клієнтами.

Однією з основних переваг є зниження операційних витрат та підвищення продуктивності. Інформаційні системи дозволяють автоматизувати рутинні процеси, такі як оформлення замовлень, оновлення складу, формування рахунків і обробка платежів, що зменшує потребу в ручній праці та мінімізує кількість помилок. Крім того, використання хмарних технологій дозволяє зменшити витрати на інфраструктуру та обслуговування системи [4].

Важливою перевагою є розширення ринку збуту та забезпечення цілодобової доступності. Онлайн-торгівля на основі інформаційних систем не обмежується географічними межами та дозволяє здійснювати продажі 24/7, що значно збільшує потенційну кількість клієнтів і обсяг продажів. Додатково підтримка багатомовності та мультивалютності сприяє виходу на міжнародні ринки [5].

Інформаційні системи також суттєво покращують якість обслуговування клієнтів. Завдяки збору та аналізу даних про поведінку користувачів забезпечується персоналізація пропозицій, рекомендацій та маркетингових кампаній, що підвищує рівень задоволеності та лояльності покупців.

Ще однією перевагою є ефективне управління запасами та логістикою. Інтеграція з системами управління складом дозволяє підтримувати актуальну інформацію про наявність товарів, оптимізувати закупівлі та скорочувати витрати на зберігання продукції [6]. У результаті підвищується швидкість обробки замовлень та покращується якість доставки.

Окремо слід відзначити покращення процесу прийняття управлінських рішень. Інформаційні системи забезпечують накопичення та аналіз великих обсягів даних, що дозволяє формувати аналітичні звіти, прогнозувати попит та оптимізувати бізнес-стратегії [7].

Отже, впровадження інформаційних систем в онлайн-торгівлю сприяє зниженню витрат, розширенню ринку, підвищенню якості обслуговування клієнтів, покращенню управління ресурсами та підвищенню ефективності прийняття рішень, що в цілому забезпечує конкурентоспроможність підприємства в умовах цифрової економіки.

1.3 Основні цілі та задачі, які вирішуються особистим кабінетом користувача

Особистий кабінет користувача в інформаційних системах онлайн-торгівлі є ключовим інтерфейсом взаємодії між користувачем і системою, який забезпечує доступ до персоналізованих функцій, даних та сервісів. У сучасних вебзастосунках він розглядається як важливий інструмент автоматизації процесів обслуговування клієнтів, управління інформацією та підвищення ефективності електронної комерції.

Основною метою особистого кабінету є забезпечення зручної та персоналізованої взаємодії користувача з інформаційною системою. Завдяки йому формується індивідуальне середовище, у якому користувач отримує доступ до своїх даних, замовлень та налаштувань, що значно спрощує використання сервісу та скорочує час виконання типових операцій [8].

Іншою важливою метою є підвищення рівня лояльності користувачів та

стимулювання повторних покупок. Використання історії замовлень, персоналізованих пропозицій та зручних механізмів повторного оформлення покупок позитивно впливає на користувацький досвід та сприяє зростанню активності клієнтів [9].

Основними задачами, які вирішує особистий кабінет користувача, є управління персональними даними, перегляд та редагування профілю, а також налаштування параметрів безпеки, зокрема зміна пароля та контроль авторизаційних даних. Також важливою задачею є управління замовленнями, що включає перегляд історії покупок, відстеження статусу замовлень, оформлення нових замовлень і контроль оплат. Це дозволяє користувачу самостійно здійснювати більшість операцій без необхідності звернення до служби підтримки. Крім того, особистий кабінет забезпечує комунікацію між користувачем і системою через повідомлення, звернення до служби підтримки або інтегровані чати. Це сприяє оперативному вирішенню проблем та підвищує якість обслуговування [8].

Отже, особистий кабінет користувача виконує функції персоналізації, автоматизації та оптимізації взаємодії з інформаційною системою, забезпечуючи зручність використання інтернет-магазину та підвищення ефективності бізнес-процесів.

1.4 Огляд існуючих інформаційних систем в торгівлі

У сучасному світі компанії, пов’язані зі сферою продажів, використовують складні інформаційні системи для автоматизації бізнес-процесів. Для аналізу та подальшого порівняння оберемо два сучасні веб-ресурси електронної комерції: універсальний маркетплейс Rozetka та цифрову платформу Xbox Store. Ці системи представляють різні підходи до реалізації онлайн-торгівлі та взаємодії з користувачем. Для аналізу та подальшого порівняння застосуємо такі критерії, як: наповненість головної сторінки, наявність особистого кабінету користувача та його можливості (перегляд замовлень, редагування особистої інформації),

пошук та фільтрація товару, наявність функції «Додати в список побажань» а також додаткові спеціальні можливості.

Rozetka [10] є найбільшою платформою електронної комерції в Україні, яка еволюціонувала з вузькоспеціалізованого інтернет-магазину електроніки до масштабного маркетплейсу. З погляду інформаційних систем, цей ресурс є складним програмним комплексом, орієнтованим на високу навантаженість та персоналізацію. Головна сторінка виконує роль потужного маркетингового інструменту (рис. 1.1). Вона містить динамічні блоки: інтерактивний каталог категорій, блоки «Акційні пропозиції», «Рекомендації на основі переглядів» та «Товари дня». Система використовує алгоритми аналізу поведінки користувача для підбору найбільш релевантного контенту вже при першому візиті.

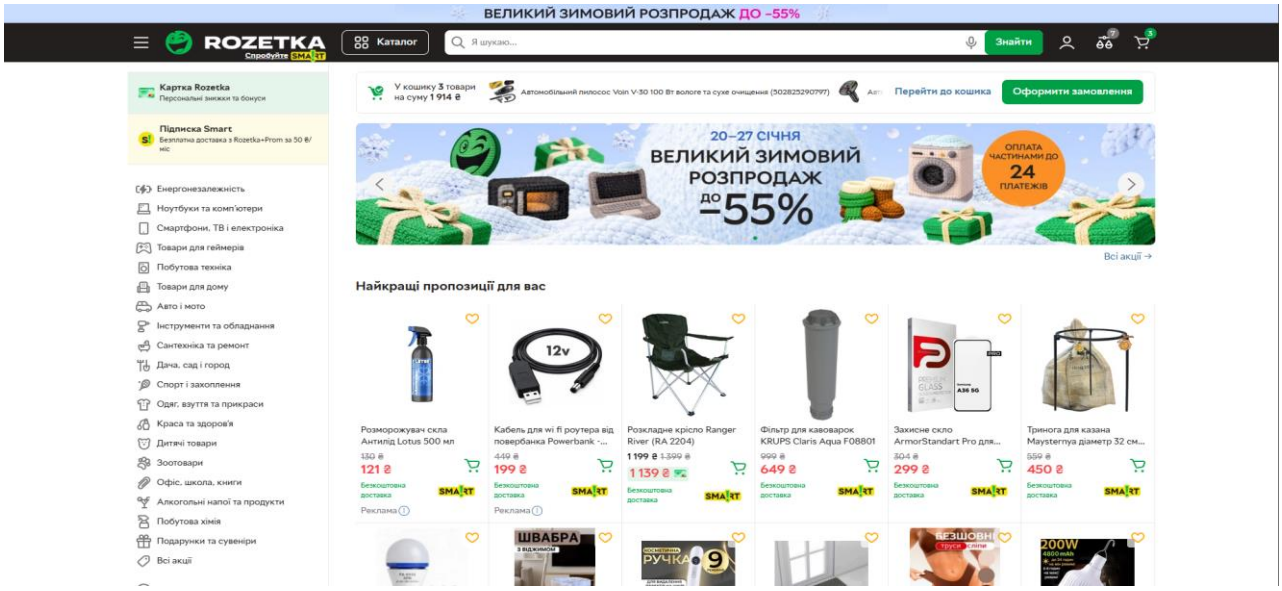


Рисунок 1.1 – Головна сторінка Rozetka

У верхньому правому куту бачимо такі елементи: особистий кабінет користувача, пошук по сайту та кошик. Більшу частину головного меню займають інформаційні банери з назвами розділів, які ведуть на сторінку із переліком товарів.

Щоб увійти до персонального кабінету користувач натискає на логотип профілю у шапці сайту. На рисунку 1.2 зображено інтерфейс кабінету користувача даного сайту.

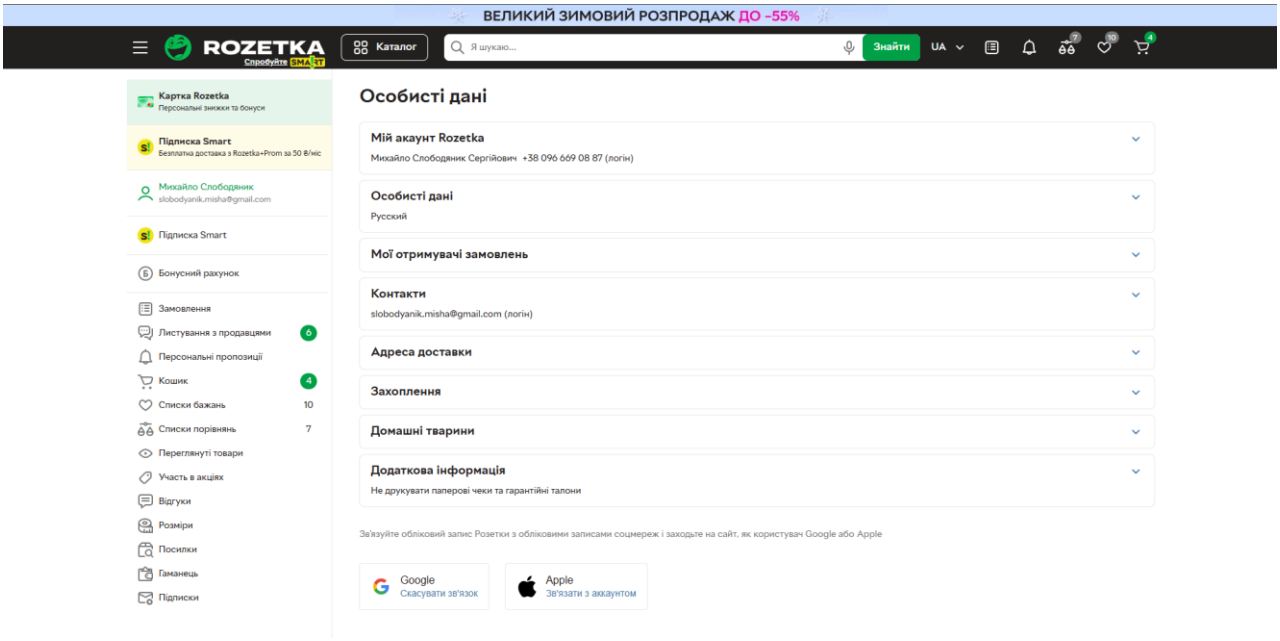


Рисунок 1.2 – Інтерфейс кабінету користувача Rozetka

Тобто аналіз інтернет-магазину Rozetka показав, що ефективність сучасної інформаційної системи електронної комерції значною мірою залежить від якісної реалізації особистого кабінету користувача та інтеграції сервісів управління замовленнями, оплатами й персональними даними. Розглянута платформа демонструє важливість персоналізації та зручної взаємодії користувача з системою як ключових чинників успішної онлайн-торгівлі.

Xbox Store (Microsoft Store) (рис. 1.3) – це глобальна екосистема електронної комерції, що спеціалізується на реалізації цифрового контенту: відеоігор, додаткового програмного забезпечення (DLC), передплат та мультимедійних сервісів [11]. З погляду архітектури інформаційних систем, цей ресурс є взірцем інтеграції хмарних технологій та систем управління цифровими правами (DRM). Платформа забезпечує централізоване зберігання даних користувачів, синхронізацію покупок між різними пристроями та автоматичне оновлення цифрового контенту. Важливою особливістю системи є інтеграція з обліковим записом Microsoft, що дозволяє користувачам отримувати доступ до придбаного контенту на консолях Xbox, персональних комп’ютерах та інших пристроях.

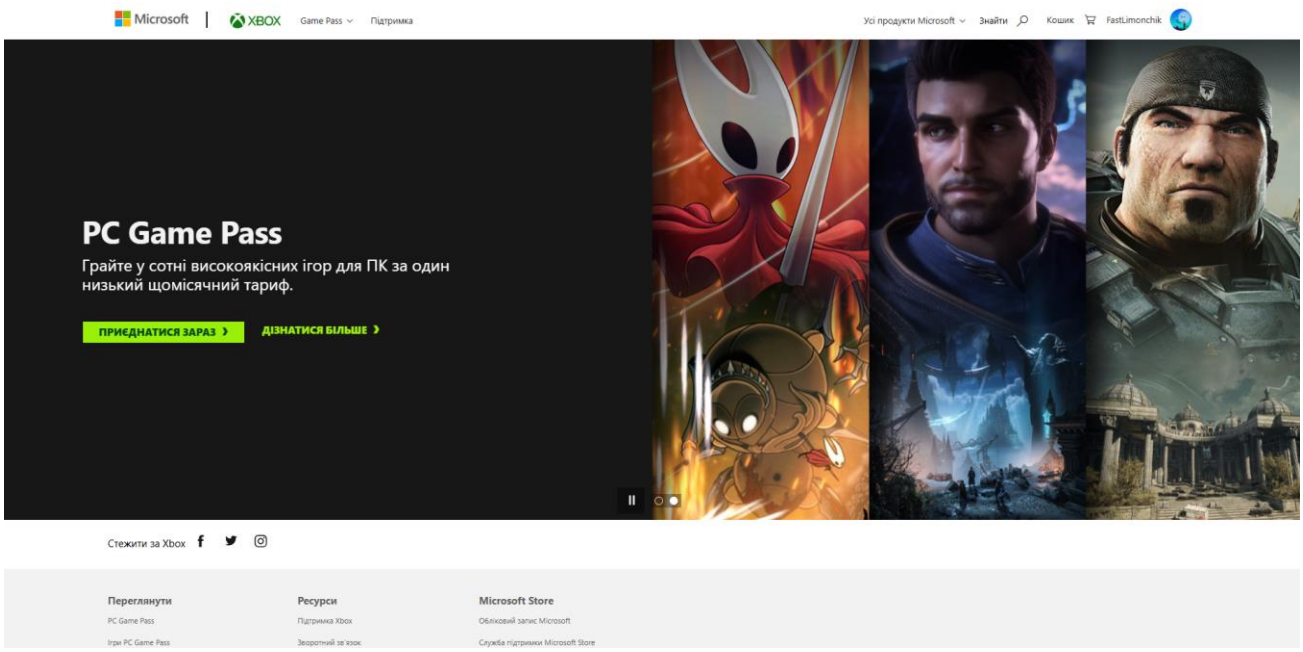


Рисунок 1.3 – Головна сторінка Xbox

Для входу в особистий кабінет натиснемо «Войти», після чого користувача зустрічає сторінка входу (рис. 1.4).

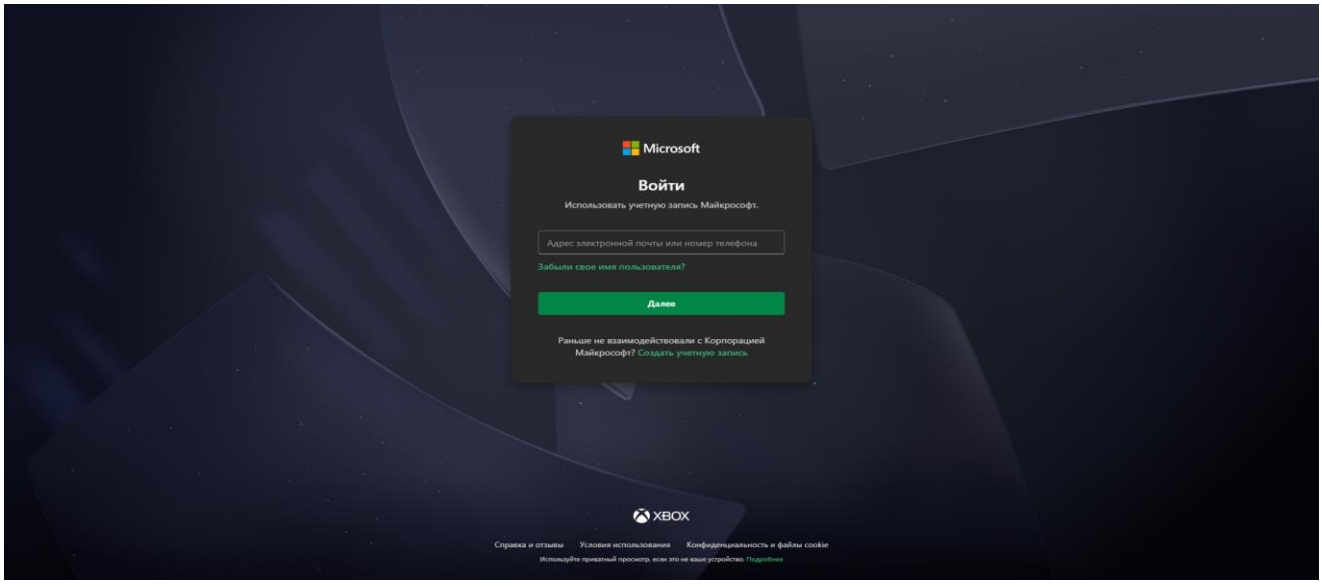


Рисунок 1.4 – Сторінка входу Xbox

Після вводу даних для входу в акаунт користувача перенаправляє із сайту Xbox за посиланням https://www.xbox.com/uk-UA/play/user/ , яке виступає в ролі

особистого кабінету користувача (рис. 1.5). Це означає, що користувач не створює окремий акаунт для Xbox, а входить за допомогою уніфікованого акаунту Xbox, який використовується для всіх сервісів Xbox – консолей, підписок, ігор.

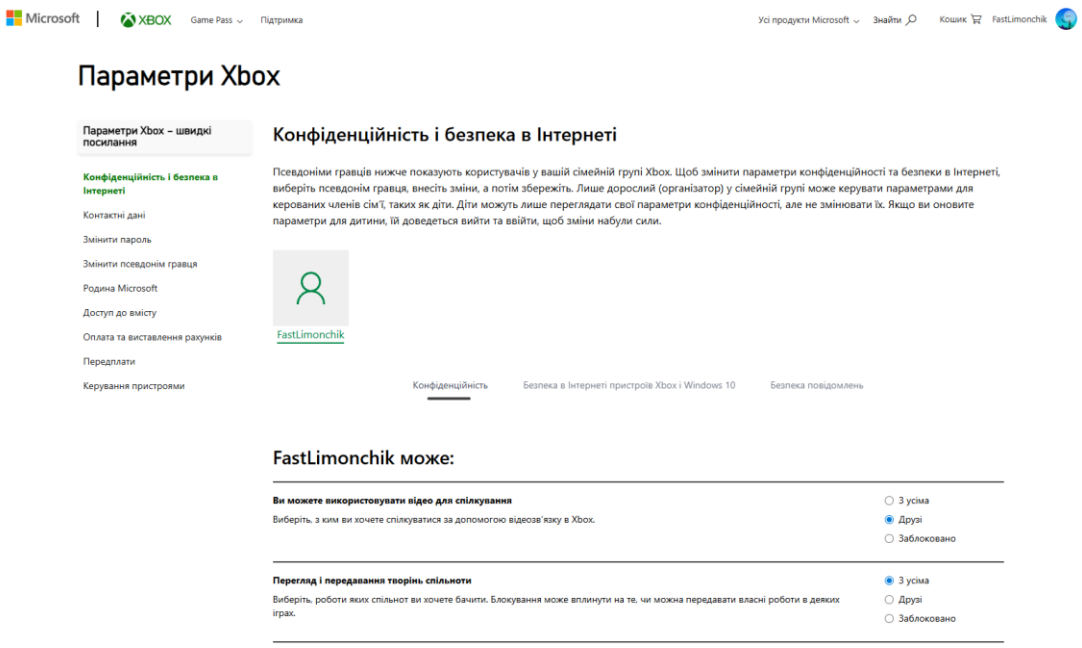


Рисунок 1.5 – Інтерфейс кабінету користувача Xbox

В особистому кабінеті користувача зібрано всі налаштування облікового запису, які охоплюють як особисту інформацію, так і ігрові та платіжні параметри. На вкладці «Профіль» користувач має змогу передивлятись та редагувати мережевий ідентифікатор, ім’я, картинку профілю та аватар, додавати інформацію про себе та змінювати мову. Особистий кабінет користувача Xbox також надає можливість керувати підписками, передивлятись історію транзакцій та замовлень, додавати або змінювати способи оплати, або оплачувати кодами (вкладка «Активувати коди»), керувати взаємодією з іншими програмами або сервісами (вкладки «Авторизовані програми» та «Прив’язані служби»). Особистий кабінет також надає можливість користувачу керувати просунутими налаштуваннями, такими як налаштування безпеки, конфіденційності та налаштування пристроїв.

Для того, щоб оглянути можливості фільтрації товару за різними категоріями, перейдемо до списку товарів. Для того, щоб передивитись товар, необхідно обрати необхідну платформу та вибрати «Ігри», після чого користувача перенаправляє на каталог ігор (рис. 1.6).

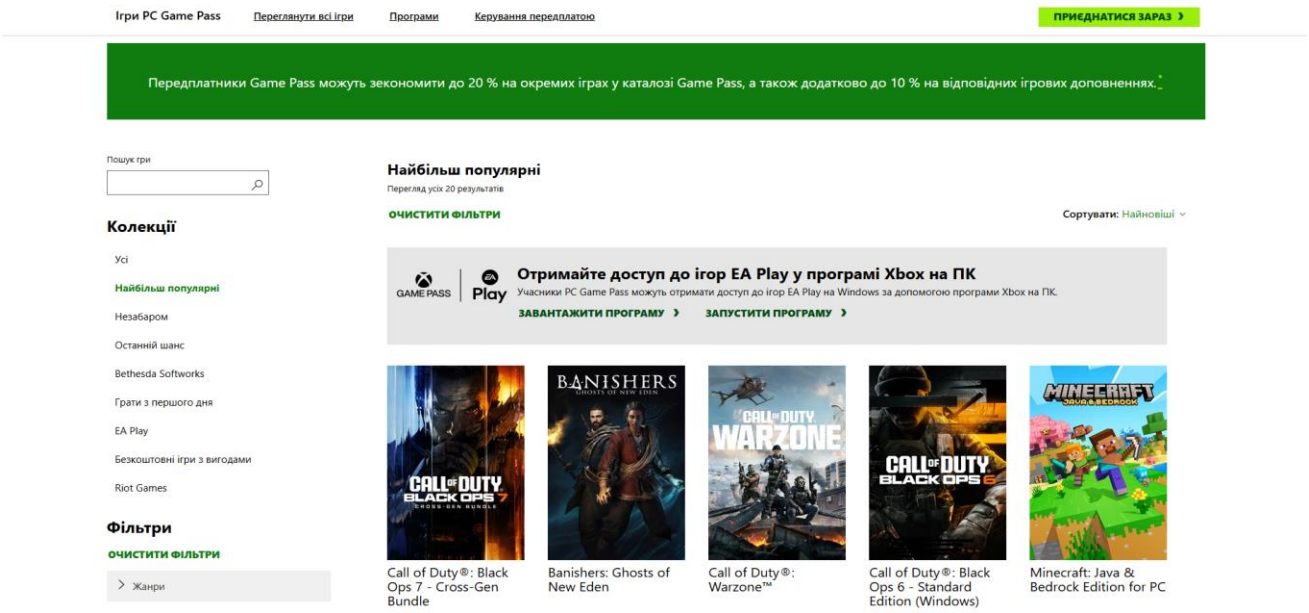


Рисунок 1.6 – Каталог ігор Xbox

Система надає можливість фільтрувати товар тільки за 4 категоріями: нещодавно випущені, умовно-безкоштовні ігри, спорт та інді-ігри. Це можна вважати певним функціональним обмеженням, оскільки відсутні більш гнучкі параметри фільтрації, такі як рейтинг гри, мова локалізації або наявність багатокористувацького режиму. Для інтернет-магазину, що реалізує велику кількість цифрових товарів, розширені можливості фільтрації товару є критично важливими, адже вони підвищують зручність пошуку, скорочують час взаємодії з каталогом і покращують користувацький досвід.

Перейшовши на будь-яку гру із каталогу, потрапляємо на головну сторінку товару (рис. 1.7). На головній сторінці товару можна спостерігати основні характеристики та можливості гри: кількість гравців, наявність внутрішньо-ігрових покупок, наявність мережевої гри, вікові обмеження та наявність

додаткових функцій. Користувач має можливість обрати пробну версію гри, а також додати гру до «Списку бажань».

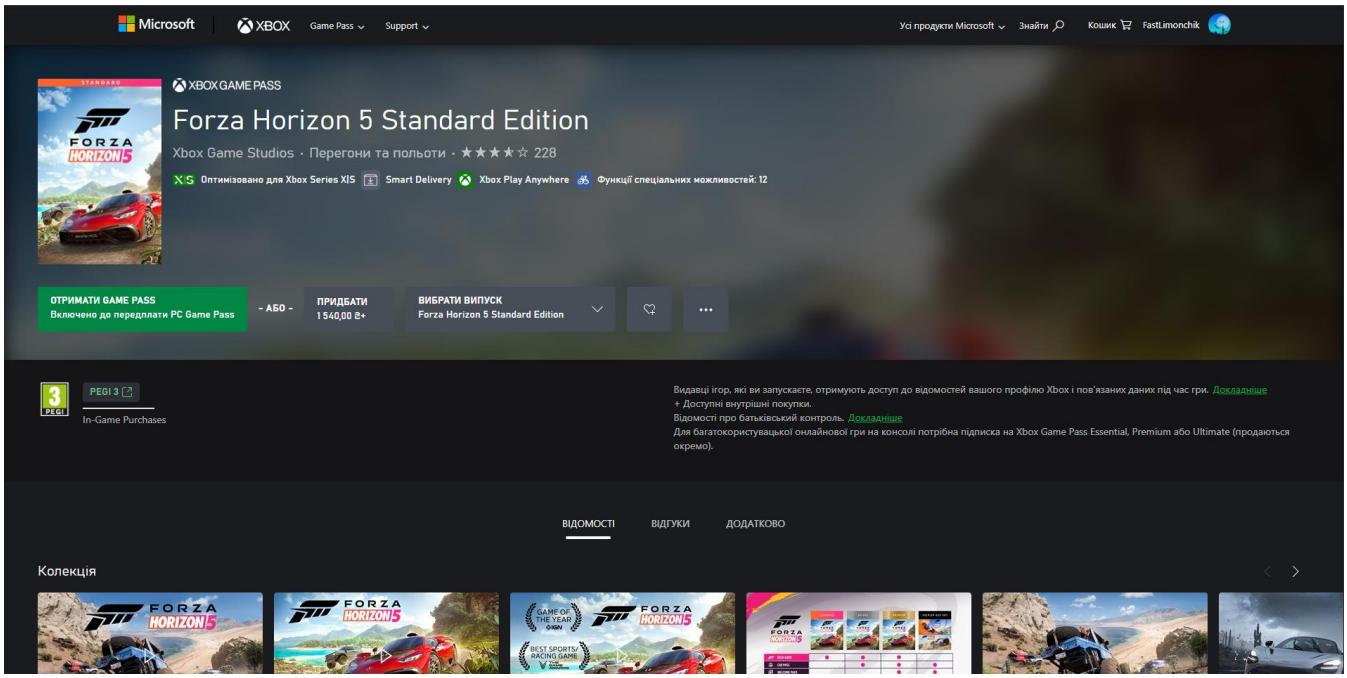


Рисунок 1.7 – Головна сторінка товару Xbox

Тобто Аналіз Xbox показав, що інформаційна система має розвинений особистий кабінет, який об’єднує керування профілем, підписками, платежами та налаштуваннями користувача. Це забезпечує зручну та централізовану взаємодію з сервісом.

Водночас виявлено обмежені можливості фільтрації товарів у каталозі, що знижує зручність пошуку. Це підкреслює важливість розширених інструментів навігації та фільтрації в інтернет-магазинах.

На основі проведеного огляду інформаційних систем Rozetka та Xbox можна виділити перелік спільних функціональних рішень, а також окремі відмінності у реалізації особистих кабінетів, фільтрації товарів та загальної зручності використання. Порівняння двох ІС наведено в таблиці 1.1.

Таблиця 1.1 – Порівняння оглянутих інформаційних систем

Критерії	**Rozetka**	**Xbox**
Пошук товару	Реалізовано	Реалізовано
Наповненість головної сторінки	Головна сторінка наповнена банерами із популярними товарами та спеціальними пропозиціями	Головна сторінка наповнена банерами із популярними іграми та спеціальними пропозиціями
Наявність особистого кабінету користувача	Реалізовано	Реалізовано
Перегляд замовлень	Реалізовано	Реалізовано
Можливість редагування особистої інформації	Реалізовано	Реалізовано
Фільтрація товару	Має розширені можливості фільтрації товарів	Має обмежені можливості фільтрації товарів
Наявність функції «Додати в список бажань»	Реалізовано	Реалізовано
Додаткові спеціальні можливості	Логістичні та фінансові сервіси: власна система точок видачі, інтеграція з картою відділень перевізників. Програми розстрочки («Оплата частинами»), бонусний рахунок «Premium». Можливість порівняння характеристик товарів та відстеження цін	Технологічна інтеграція: Smart Delivery (автоматичне отримання кращої версії гри), Xbox Game Pass, хмарні збереження, система досягнень (Gamerscore), віддалене встановлення ігор та батьківський контроль

Отже, обидві розглянуті інформаційні системи відповідають базовим вимогам до сучасних e-commerce платформ. Проте кожна з розглянутих систем має певні недоліки, які можна мінімізувати. Наприклад, спрощення інтерфейсу головної сторінки та усунення інформаційного перевантаження допоможе користувачам Rozetka швидше фокусуватися на виборі товарів, не відволікаючись на надмірну кількість рекламних блоків. Своєю чергою оптимізація швидкості завантаження медіаконтенту та впровадження більш гнучких інструментів управління кошиком для регіональних ринків значно покращить доступність сервісів Xbox Store для користувачів із низькою швидкістю інтернет-з’єднання.

1.5 Мета роботи

Метою роботи є розробка інформаційної системи для онлайн-реалізації товарів з особистим кабінетом користувача.

Для досягнення поставленої мети в роботі вирішуються наступні завдання:

- вибір та аналіз предметної області та виявлення особливостей інформаційної системи для онлайн-реалізації товару обраної предметної області
- розробка вимог до Інтернет-магазину;
- формалізація основних задач Інтернет-магазину;
- проєктування архітектури Інтернет-магазину;
- розробка інтерфейсу користувача;
- розробка алгоритмів реалізації задач інформаційної системи;
- вибір інструментальних засобів для реалізації програмного продукту;
- розробка системи класів (бібліотек, програм, допоміжних утиліт, SQL-запитів);
- опис основних модулів програмного продукту;

1.6 Висновки

У першому розділі розглянуто теоретичні основи функціонування інформаційних систем в онлайн-торгівлі, їх роль, особливості та значення в умовах розвитку цифрової економіки. Встановлено, що сучасні інформаційні системи є ключовим інструментом автоматизації бізнес-процесів електронної комерції, забезпечуючи взаємодію між користувачами, товарами, замовленнями та зовнішніми сервісами.

Проаналізовано переваги використання інформаційних систем у сфері онлайн-торгівлі, серед яких зниження витрат, підвищення ефективності

управління, розширення ринку збуту, покращення якості обслуговування клієнтів та можливість персоналізації взаємодії з користувачами.

Окрему увагу приділено особистому кабінету користувача як важливому елементу сучасних e-commerce систем. Визначено його основні функції, цілі та задачі, що полягають у забезпеченні зручного доступу до персональних даних, історії замовлень, налаштувань профілю та інструментів взаємодії з системою.

Також проведено огляд існуючих інформаційних систем онлайн-торгівлі, на прикладі платформ Rozetka та Xbox Store, що дозволило визначити спільні риси та відмінності у реалізації функціоналу, особистих кабінетів і механізмів фільтрації товарів.

У результаті встановлено, що ефективна інформаційна система онлайн-торгівлі повинна поєднувати функціональність, зручність використання, продуктивність та високий рівень персоналізації, що є основою для подальшого проєктування та розробки власної системи.

2 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ОНЛАЙН РЕАЛІЗАЦІЇ ТОВАРІВ З ОСОБИСТИМ КАБІНЕТОМ КОРИСТУВАЧА

2.1 Аналіз вимог до системи інтернет-магазину

Розроблювана система інтернет-магазину є повнофункціональним веб-застосунком, призначеним для реалізації процесів електронної комерції. Вона забезпечує користувачам можливість перегляду каталогу товарів, здійснення пошуку та фільтрації продукції, оформлення замовлень, а також управління персональним кабінетом. Окремо передбачено адміністративний модуль, який дозволяє здійснювати управління товарами, категоріями та користувачами системи.

Загальна архітектура системи орієнтована на розподіл функціональності між клієнтською та серверною частинами, що забезпечує масштабованість, зручність підтримки та можливість подальшого розширення функціоналу.

Функціональні вимоги (табл. 2.1) визначають перелік основних можливостей системи та описують поведінку її ключових модулів у процесі взаємодії з користувачем. Вони охоплюють усі етапи роботи інтернет-магазину – від перегляду каталогу до управління замовленнями та адміністративного контролю.

Таблиця 2.1 – Функціональні вимоги за модулями системи

ID	Модуль	Вимога	Пріоритет
FR-01	Каталог	Перегляд списку товарів з пагінацією	Високий
FR-02	Каталог	Фільтрація товарів за категорією	Високий
FR-03	Каталог	Сортування товарів (ціна, назва, дата)	Високий
FR-04	Каталог	Пошук товарів за назвою	Високий
FR-05	Каталог	Перегляд детальної картки товару	Високий
FR-06	Кошик	Додавання товарів до кошика	Високий
FR-07	Кошик	Зміна кількості товарів у кошику	Високий
FR-08	Кошик	Видалення товарів з кошика	Високий
FR-09	Кошик	Розрахунок загальної суми замовлення	Високий
FR-10	Замовлення	Оформлення замовлення з вибором доставки	Високий

Продовження таблиці 2.1

ID	Модуль	Вимога	Пріоритет
FR-11	Замовлення	Вибір способу оплати	Високий
FR-12	Замовлення	Перегляд історії замовлень	Середній
FR-13	Аутентифікація	Реєстрація користувача	Високий
FR-14	Аутентифікація	Авторизація користувача	Високий
FR-15	Аутентифікація	Вихід із системи	Високий
FR-16	Особистий кабінет	Редагування профілю користувача	Середній
FR-17	Особистий кабінет	Перегляд історії замовлень	Середній
FR-18	Wishlist	Додавання товарів до списку бажань	Середній
FR-19	Wishlist	Видалення товарів зі списку бажань	Середній
FR-20	Адмін-панель	Управління товарами (CRUD)	Високий
FR-21	Адмін-панель	Управління категоріями	Високий
FR-22	Адмін-панель	Управління замовленнями	Високий
FR-23	Адмін-панель	Управління користувачами	Середній
FR-24	Адмін-панель	Перегляд статистики продажів	Низький

Функціональні вимоги демонструють, що система охоплює повний цикл електронної комерції: від взаємодії користувача з каталогом товарів до обробки замовлень та адміністративного управління. Особливу роль відіграє модуль кошика та замовлень, оскільки саме він забезпечує реалізацію основного бізнес-процесу – купівлі товару.

Нефункціональні вимоги (табл. 2.2) визначають якісні характеристики системи, що впливають на її продуктивність, безпеку, стабільність та зручність використання. Вони є критично важливими для забезпечення стабільної роботи інтернет-магазину в умовах реального навантаження.

Таблиця 2.2 – Нефункціональні вимоги до системи

Категорія	Вимога	Метрика
Продуктивність	Час завантаження сторінки	< 3 сек
Продуктивність	Час відповіді API	< 500 мс
Масштабованість	Одночасні користувачі	до 1000
Безпека	Хешування паролів	bcrypt
Безпека	Аутентифікація	JWT
Безпека	Захист від SQL-ін’єкцій	параметризовані запити
Доступність	Безвідмовна робота	99.5%
Сумісність	Підтримка браузерів	Chrome, Firefox, Edge, Safari
Адаптивність	Підтримка пристроїв	Desktop, Tablet, Mobile
Локалізація	Мова інтерфейсу	Українська

Нефункціональні вимоги визначають, що система повинна бути не лише

функціонально повною, але й швидкою, безпечною та зручною у використанні. Особливу увагу приділено питанням продуктивності та захисту даних користувачів, що є критичним для будь-якої інформаційної системи електронної комерції.

Подальший аналіз системи передбачає визначення ролей користувачів та їхніх прав доступу (табл. 2.3). У розроблюваній системі виділено чотири основні ролі: гість, покупець (customer), менеджер та адміністратор. Така модель дозволяє забезпечити чітке розмежування доступу до функціоналу та підвищити безпеку системи.

Таблиця 2.3 – Матриця доступу за ролями користувачів

Функція	**Гість**	**Customer**	**Manager**	**Admin**
Перегляд каталогу	✓	✓	✓	✓
Пошук товарів	✓	✓	✓	✓
Перегляд товару	✓	✓	✓	✓
Реєстрація	✓	–	–	–
Авторизація	✓	✓	✓	✓
Додавання в кошик	–	✓	✓	✓
Оформлення замовлення	–	✓	✓	✓
Wishlist	–	✓	✓	✓
Особистий кабінет	–	✓	✓	✓
Управління товарами	–	–	✓	✓
Управління категоріями	–	–	✓	✓
Обробка замовлень	–	–	✓	✓
Статистика	–	–	✓	✓
Управління користувачами	–	–	–	✓
Призначення ролей	–	–	–	✓
Налаштування системи	–	–	–	✓

Як видно з таблиці 2.3, гість має найобмеженіший доступ і може лише переглядати каталог товарів, здійснювати пошук та перегляд окремих товарних позицій, а також проходити процес реєстрації та авторизації. Роль покупця (customer) передбачає повноцінну взаємодію з функціоналом інтернет-магазину, зокрема можливість додавання товарів до кошика, оформлення замовлень, використання списку бажань (wishlist) та доступ до особистого кабінету.

Ролі менеджера та адміністратора мають розширені права доступу, які включають управління товарами, категоріями та обробку замовлень. Менеджер відповідає за операційні процеси магазину, тоді як адміністратор додатково має доступ до управління користувачами, призначення ролей та налаштування системи. Такий підхід забезпечує чітку ієрархію доступу та дозволяє ефективно розподіляти обов’язки між учасниками системи.

Отже, запропонована модель ролей дозволяє організувати багаторівневу систему доступу, де кожен користувач має лише ті можливості, які відповідають його функціональній ролі в системі. Це підвищує як безпеку, так і керованість інформаційної системи.

2.2 Проєктування архітектури вебзастосунку

Архітектура вебзастосунку інтернет-магазину побудована на основі патерну MVC (Model–View–Controller) [12] із використанням сучасного фреймворку Next.js 16 [13], який підтримує концепції App Router та Server Components. Такий підхід дозволяє поєднати серверну та клієнтську логіку в єдиній структурі, забезпечуючи при цьому чіткий розподіл відповідальності між компонентами системи, що позитивно впливає на масштабованість і зручність подальшого супроводу.

Загальна архітектура системи включає три основні рівні: клієнтський рівень представлення, серверний рівень бізнес-логіки та рівень доступу до даних (рис. 2.1). Взаємодія між цими рівнями здійснюється через HTTP-запити та API у форматі JSON, що забезпечує універсальність і сумісність між різними частинами застосунку.

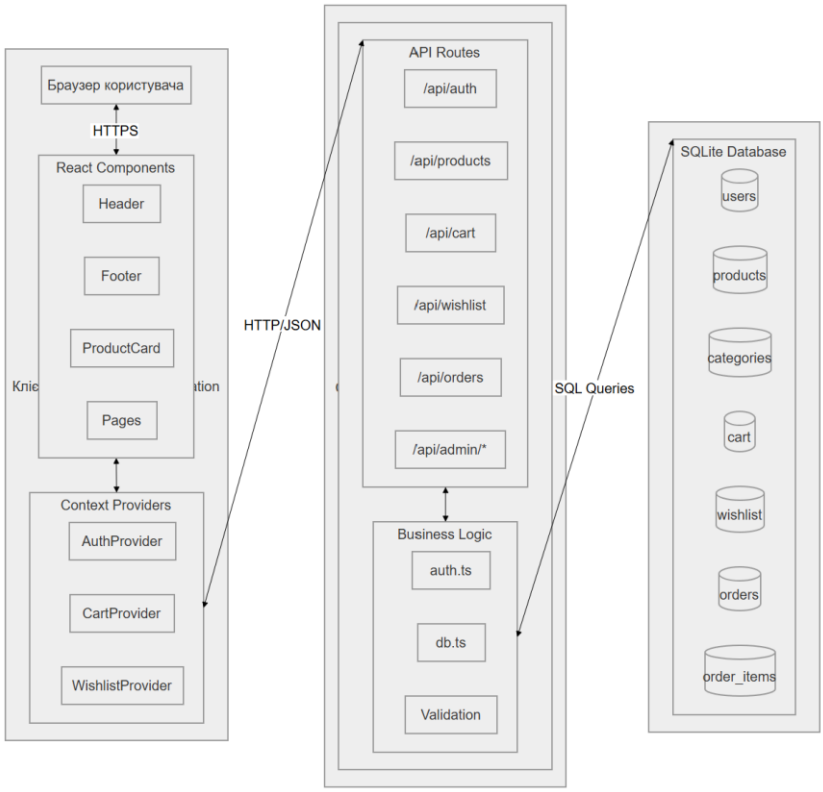


Рисунок 2.1 – Високорівнева архітектура системи інтернет-магазину

На рисунку 2.2 показано високорівневу архітектуру інформаційної системи інтернет-магазину, яка складається з трьох основних рівнів: клієнтського, серверного та рівня доступу до даних.

Клієнтський рівень відповідає за інтерфейс користувача, який реалізовано за допомогою React [14] і Next.js [13]. Тут розміщені сторінки застосунку та UI-компоненти (Header, Footer, ProductCard), а також контекстні провайдери (AuthProvider, CartProvider, WishlistProvider), що забезпечують керування станом.

Серверний рівень реалізує бізнес-логіку через API Routes Next.js, включаючи автентифікацію, обробку запитів і валідацію даних. Саме тут працюють основні ендпоінти системи для товарів, кошика, замовлень та адміністративних функцій.

Рівень даних представлений базою даних SQLite [15], у якій зберігаються основні сутності системи. Взаємодія між рівнями здійснюється через HTTP/JSON-запити, що забезпечує структурованість і масштабованість системи.

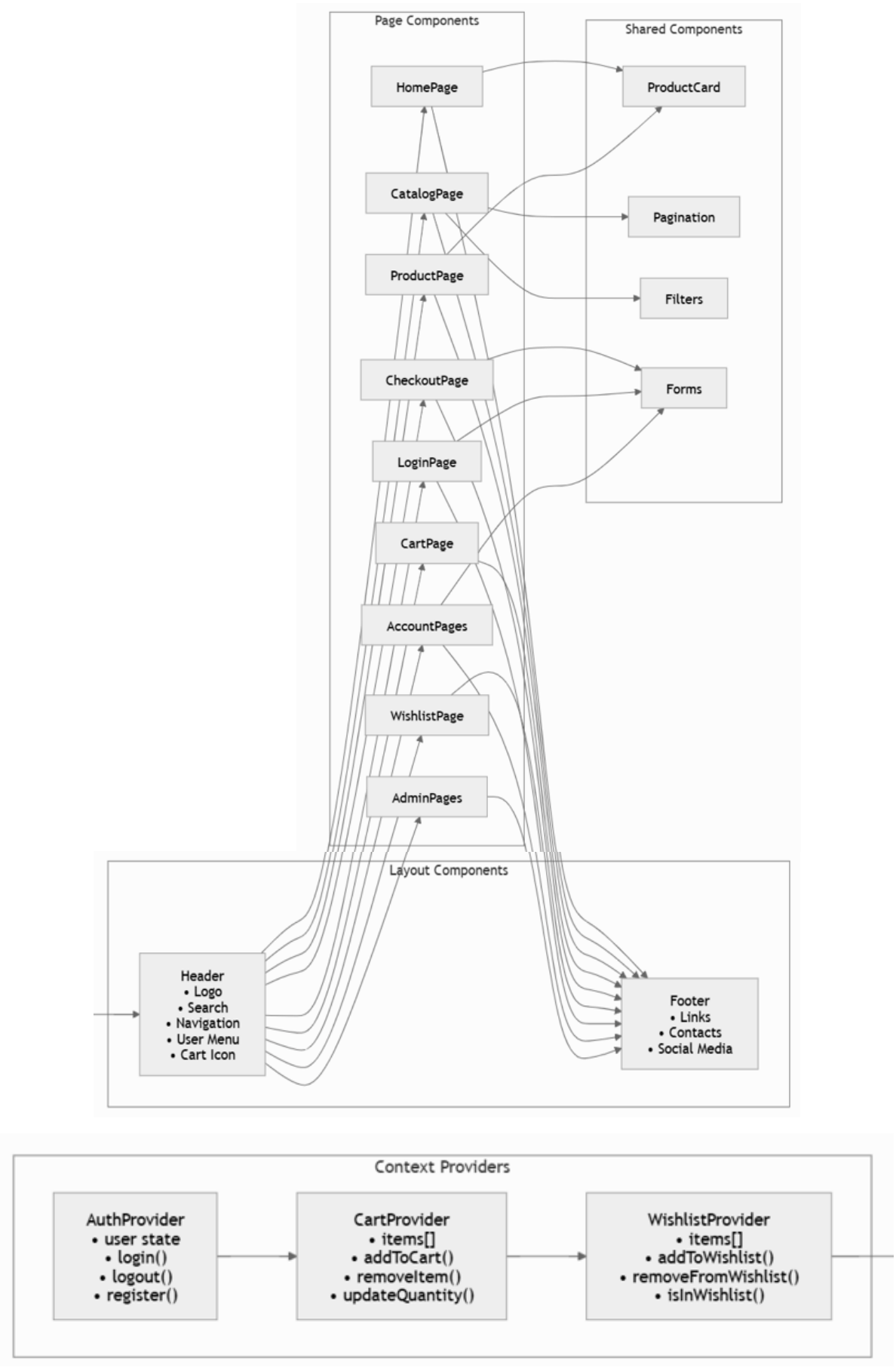


Рисунок 2.2 – Компонентна архітектура React-застосунку

У межах представленої архітектури користувач взаємодіє з клієнтською частиною застосунку через браузер, де відбувається відображення інтерфейсу та обробка взаємодій. Далі запити передаються до серверної частини, реалізованої через API Routes Next.js, які виконують роль контролера та забезпечують бізнес-логіку системи, зокрема обробку користувачів, товарів, кошика та замовлень. Рівень даних реалізовано на основі SQLite бази даних, у якій зберігаються всі основні сутності системи, включаючи інформацію про товари, категорії, користувачів, замовлення та списки бажань. Така багаторівнева структура дозволяє забезпечити логічну ізоляцію компонентів та підвищує гнучкість системи.

Подальший аналіз архітектури передбачає розгляд її реалізації через адаптований патерн MVC, який враховує особливості React-екосистеми. У класичному розумінні модель відповідає за дані, представлення – за інтерфейс, а контролер – за обробку логіки та взаємодію між ними. У даній системі модель представлена базою даних SQLite та TypeScript-типами, представлення реалізовано через React-компоненти та сторінки Next.js, а роль контролера виконують API маршрути та контекстні провайдери стану.

Відповідність між компонентами системи та класичним MVC-патерном наведено у таблиці 2.4.

Таблиця 2.4 – Відповідність компонентів системи MVC-патерну

Шар MVC	**Компоненти системи**	**Відповідальність**
Model	SQLite, db.ts, TypeScript-типи	Зберігання даних і бізнес-логіка
View	React Components, сторінки Next.js	Відображення інтерфейсу користувача
Controller	API Routes, Context Providers	Обробка запитів і керування станом

Тобто реалізована архітектура забезпечує логічне розділення відповідальностей, що дозволяє незалежно змінювати окремі частини системи без впливу на інші компоненти, що є важливою умовою для масштабованих вебзастосунків.

Далі варто розглянути компонентну архітектуру фронтенд-частини системи, яка базується на принципах повторного використання компонентів та

модульності. Основним елементом є кореневий layout застосунку, який об’єднує всі сторінки та забезпечує єдину структуру інтерфейсу. Компонентна структура наведена на рисунку 2.2.

На верхньому рівні знаходиться App (layout.tsx), який виступає базовим каркасом застосунку та об’єднує всі інші частини. До нього підключаються layout-компоненти, контекст-провайдери та спільні UI-компоненти.

Ключову роль відіграють Context Providers, які забезпечують глобальний стан застосунку:

AuthProvider – керування авторизацією користувача (вхід, вихід, реєстрація, стан user);

CartProvider – управління кошиком (додавання, видалення, зміна кількості товарів);

WishlistProvider – робота зі списком бажань.

Інтерфейс користувача формується через shared components, такі як Header, Footer, ProductCard, Pagination, Filters та Forms, які використовуються повторно в різних частинах системи.

Окремо виділені сторінки застосунку: HomePage, CatalogPage, ProductPage, CartPage, CheckoutPage, WishlistPage, LoginPage, AccountPages та AdminPages, які реалізують основний функціонал інтернет-магазину.

Таким чином, діаграма показує модульну структуру React-застосунку, де логіка стану винесена в контексти, а інтерфейс побудований з повторно використовуваних компонентів і сторінок.

Серверна частина системи реалізована у вигляді REST API, що забезпечує обмін даними між клієнтом і сервером у форматі JSON. API структуровано за функціональними модулями, включаючи роботу з користувачами, товарами, кошиком, замовленнями та адміністративними функціями. Основні ендпоінти системи наведено у таблиці 2.5.

Таблиця 2.5 – Специфікація REST API системи

Метод	Endpoint	Опис	Авторизація
POST	/api/auth	Реєстрація та авторизація користувача	Ні
GET	/api/products	Отримання списку товарів	Ні
GET	/api/products/[slug]	Отримання детальної інформації про товар	Ні
POST	/api/cart	Додавання товару до кошика	Так
DELETE	/api/cart	Видалення товару з кошика	Так
GET	/api/orders	Отримання замовлень користувача	Так
POST	/api/orders	Створення нового замовлення	Так
GET	/api/admin/users	Отримання списку користувачів	Admin
GET	/api/admin/stats	Отримання статистики системи	Manager та Admin

Структура API логічно поділена на публічні та захищені ендпоінти, що дозволяє забезпечити контроль доступу та підвищити рівень безпеки системи. Загальна структура API наведена на рисунку 2.3.

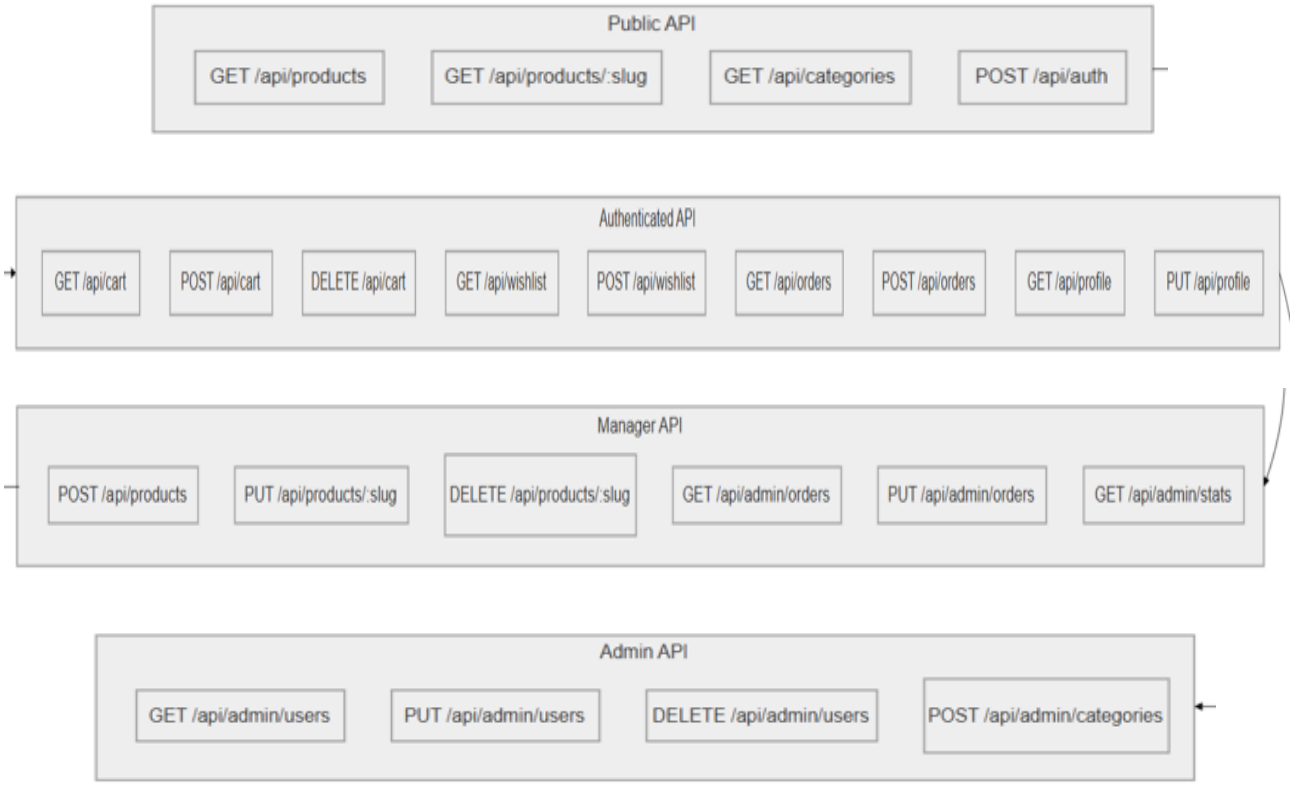


Рисунок 2.3 – Структура API ендпоінтів системи

У системі виділено чотири основні групи API:

Public API – відкриті ендпоінти для роботи з каталогом товарів, категоріями та авторизацією користувачів (перегляд товарів, отримання інформації про продукти, вхід у систему).

Authenticated API – ендпоінти, доступні лише авторизованим користувачам, які забезпечують роботу з кошиком, списком бажань, профілем та замовленнями.

Manager API – набір адміністративних функцій для керування товарами, замовленнями та аналітикою системи.

Admin API – розширений рівень доступу для управління користувачами, категоріями, замовленнями та системними статистиками.

Наступним етапом проєктування архітектури є організація файлової структури проєкту, яка побудована за принципом модульності та логічного групування компонентів. Основні каталоги включають app для сторінок, components для UI-компонентів, api для серверної логіки, а також допоміжні модулі utils і lib. Структуру проєкту наведено на рисунку 2.4.

Проєкт поділено на кілька основних зон:

app/ – сторінки застосунку (каталог, товар, кошик, оформлення замовлення, профіль, адмін-панель) та API-роути Next.js.

components/ – повторно використовувані UI-компоненти (header, footer, product card, провайдери стану).

lib/ – допоміжні утиліти та логіка роботи з базою даних.

tests/ – модульні та end-to-end тести.

ui components (shadcn) – набір стандартних інтерфейсних компонентів.

utils та auth файли – допоміжна логіка, аутентифікація та конфігурація.

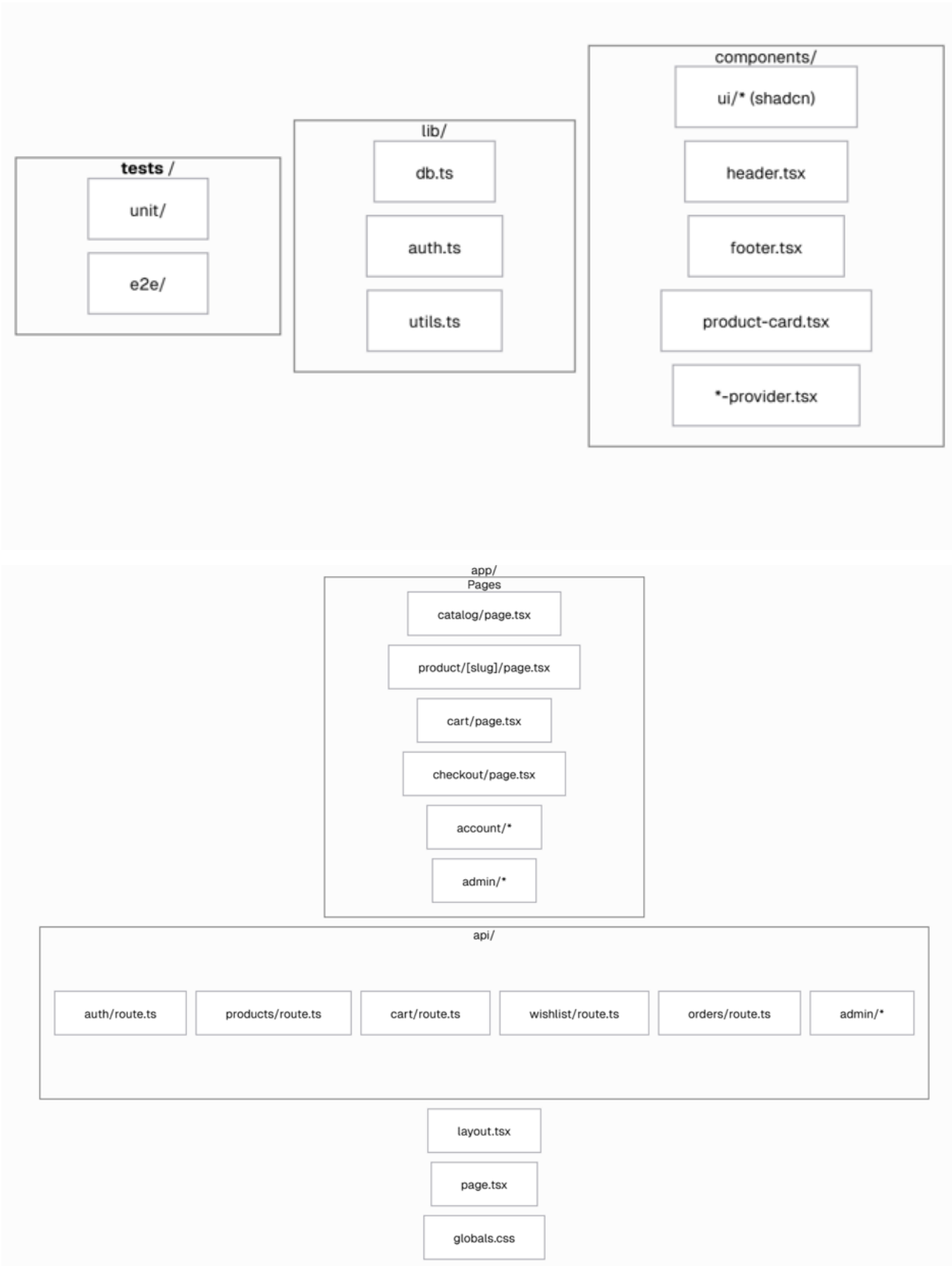


Рисунок 2.4 – Структура директорій проєкту

Така організація дозволяє чітко розмежувати відповідальність між частинами системи, що спрощує її супровід та масштабування.

Завершальним етапом архітектурного проєктування є опис процесу розгортання системи. Застосунок розгортається на платформі Vercel із використанням CI/CD підходу, що дозволяє автоматизувати процеси збірки, тестування та публікації оновлень. Загальна схема розгортання наведена на рисунку 2.5.

Основні елементи:

GitHub Repository + CI/CD pipeline (Jenkins) – джерело коду та автоматизована система збірки й деплою.

Vercel Platform – основне середовище розгортання Next.js застосунку.

Next.js Application – серверна та клієнтська частина, що працює як serverless функції.

SQLite Database – локальна база даних для зберігання інформації про користувачів, товари та замовлення.

Vercel Edge Network (CDN) – глобальна мережа доставки контенту (Європа, Америка, Азія).

Користувачі через Інтернет – кінцеві клієнти, які отримують доступ до системи через браузер.

Отже, запропонована архітектура вебзастосунку забезпечує високу продуктивність, масштабованість і зручність підтримки завдяки чіткому розділенню шарів, модульній структурі та використанню сучасних технологічних рішень.

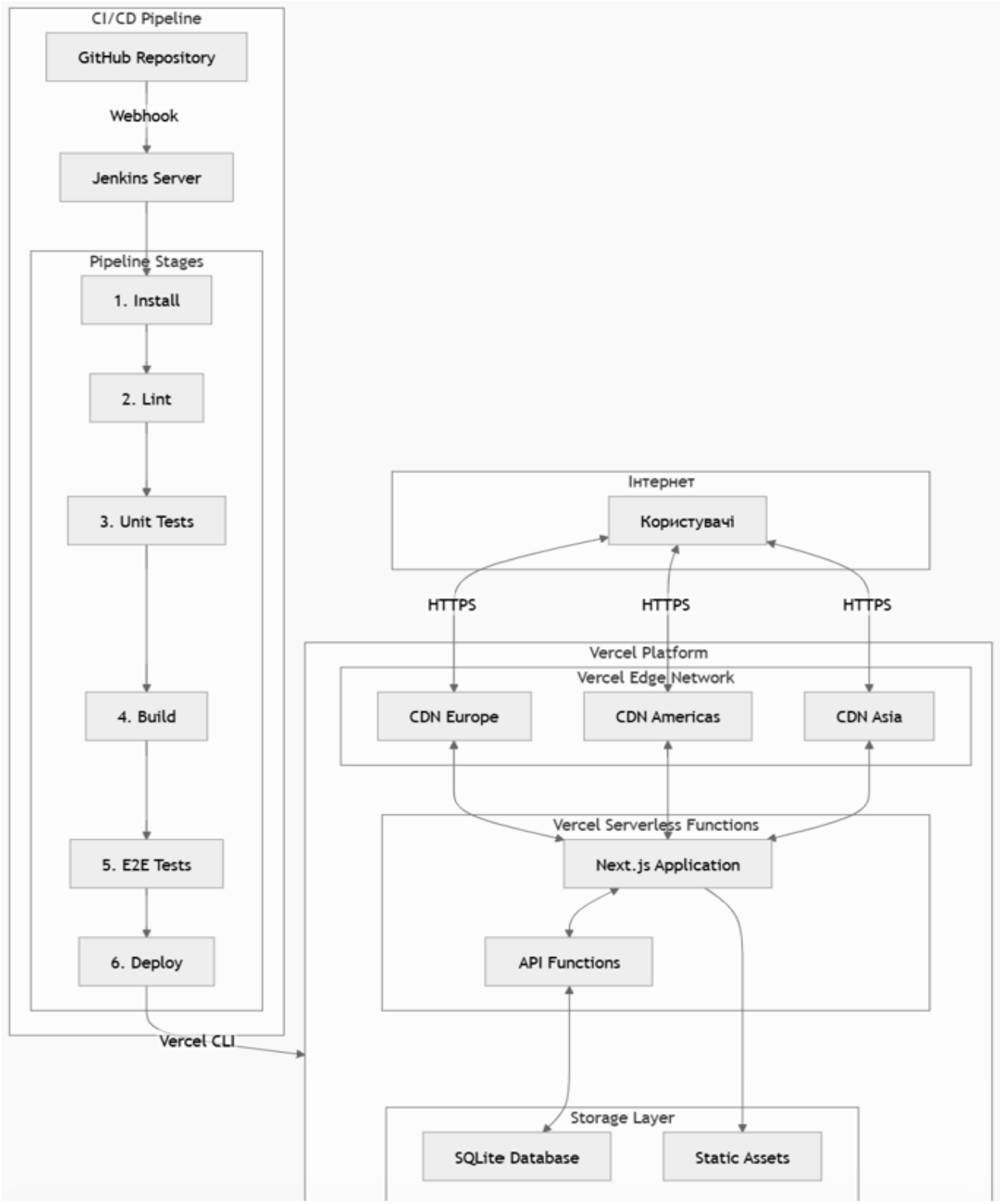


Рисунок 2.5 – Діаграма розгортання системи інтернет-магазину

2.3 Проєктування бази даних для управління товарами та користувачами

Проєктування бази даних є ключовим етапом розробки інтернет-магазину, оскільки забезпечує зберігання даних про користувачів, товари та замовлення. Для реалізації системи обрано SQLite як легку реляційну СУБД з високою продуктивністю та підтримкою ACID-транзакцій, що не потребує окремого серверного розгортання.

Під час проєктування враховано функціональні вимоги системи щодо каталогу товарів, користувачів, кошика та замовлень. Модель даних побудована на основі взаємопов'язаних сутностей зі зв'язками «один до багатьох» і «багато до одного». Загальна структура наведена на ER-діаграмі (рис. 2.6).

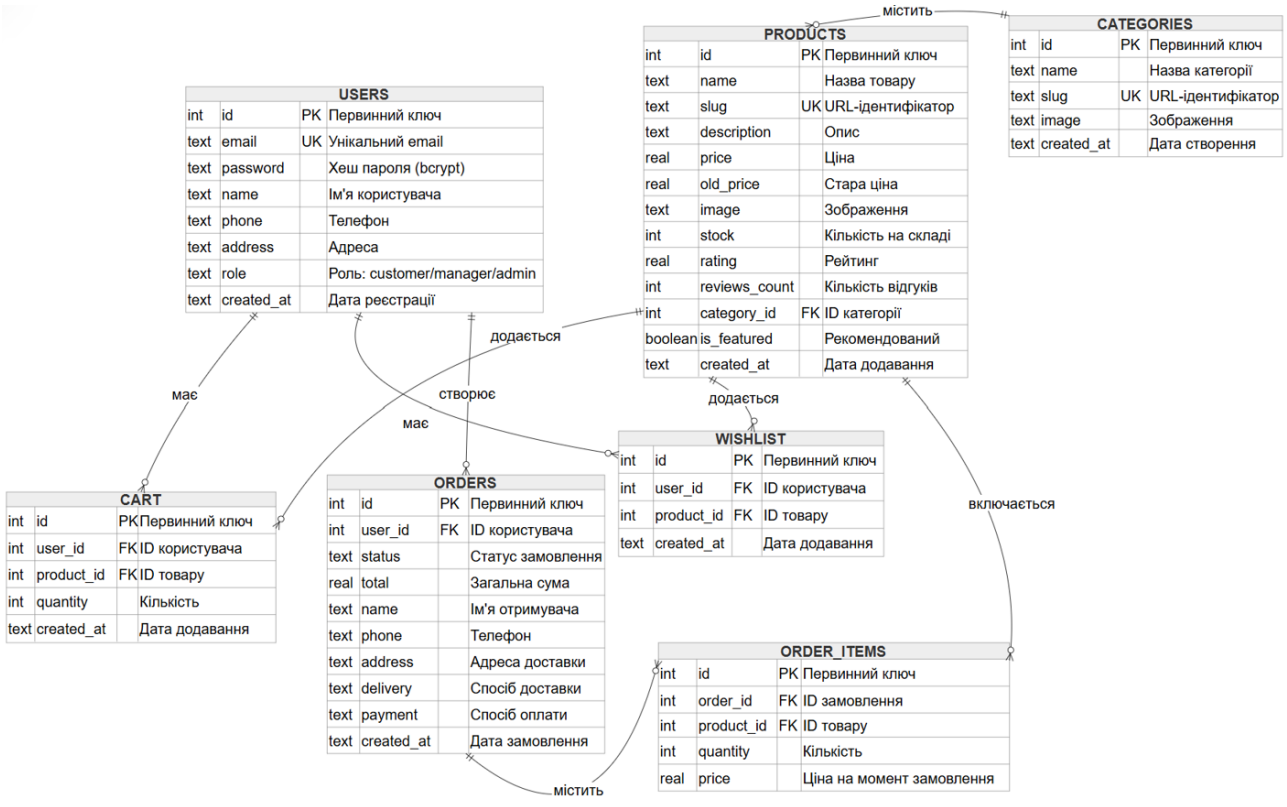


Рисунок 2.6 – ER-діаграма бази даних

Основною сутністю системи є таблиця users (табл. 2.6), яка містить інформацію про зареєстрованих користувачів вебзастосунку. У таблиці

зберігаються електронна пошта користувача, хешований пароль, ім’я, контактний телефон, адреса доставки, роль користувача та дата реєстрації. Поле email є унікальним, що виключає можливість дублювання облікових записів. Для підвищення безпеки паролі зберігаються у вигляді bcrypt-хешів. Таблиця users пов’язана з таблицями cart, wishlist та orders, оскільки кожен користувач може мати власний кошик, список бажань та історію замовлень.

Таблиця 2.6 – Структура таблиці users

Поле	**Тип**	**Обмеження**	**Опис**
id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Унікальний ідентифікатор
email	TEXT	UNIQUE, NOT NULL	Електронна пошта
password	TEXT	NOT NULL	Хеш пароля
name	TEXT	NOT NULL	Повне ім’я
phone	TEXT	NULL	Контактний телефон
address	TEXT	NULL	Адреса доставки
role	TEXT	DEFAULT ‘customer’	Роль користувача
created_at	TEXT	DEFAULT CURRENT_TIMESTAMP	Дата реєстрації

Для структурування товарів у системі використовується таблиця categories, яка містить назву категорії, URL-ідентифікатор (slug), зображення та дату створення. Кожна категорія може містити необмежену кількість товарів, що реалізується через зовнішній ключ category_id у таблиці products.

Центральним елементом системи є таблиця products (табл. 2.7), яка використовується для зберігання інформації про товари інтернет-магазину. У таблиці містяться назва товару, опис, ціна, стара ціна для реалізації механізму знижок, зображення, кількість товару на складі, рейтинг, кількість відгуків та ознака рекомендованого товару.

Для формування SEO-оптимізованих URL-адрес використовується поле slug, яке має обмеження унікальності. Таблиця products має зв’язки з таблицями cart, wishlist та order_items, що дозволяє використовувати товари у кошику, списку бажань та замовленнях користувачів.

Таблиця 2.7 – Структура таблиці products

Поле	Тип	Обмеження	Опис
id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Унікальний ідентифікатор
name	TEXT	NOT NULL	Назва товару
slug	TEXT	UNIQUE, NOT NULL	URL-ідентифікатор
description	TEXT	NULL	Опис товару
price	REAL	NOT NULL	Поточна ціна
old_price	REAL	NULL	Стара ціна
image	TEXT	NULL	URL зображення
stock	INTEGER	DEFAULT 0	Кількість на складі
rating	REAL	DEFAULT 0	Середній рейтинг
reviews_count	INTEGER	DEFAULT 0	Кількість відгуків
category_id	INTEGER	FOREIGN KEY	Посилання на категорію
is_featured	BOOLEAN	DEFAULT FALSE	Рекомендований товар
created_at	TEXT	DEFAULT CURRENT_TIMESTAMP	Дата додавання

Для реалізації функціоналу кошика створено таблицю cart, у якій зберігаються товари, додані користувачем перед оформленням замовлення. Таблиця містить посилання на користувача та товар, а також кількість одиниць товару. Обмеження UNIQUE(user_id, product_id) забезпечує відсутність дублювання одного й того самого товару в кошику конкретного користувача.

Схожий принцип використовується для реалізації списку бажань через таблицю wishlist. Вона дозволяє користувачам зберігати товари для подальшого перегляду або покупки. Для уникнення повторного додавання одного товару до списку бажань також використовується унікальна комбінація user_id та product_id.

Збереження інформації про покупки реалізовано за допомогою таблиці orders (табл. 2.8). У ній містяться дані про користувача, статус замовлення, загальну суму, контактні дані отримувача, адресу доставки, спосіб доставки, спосіб оплати та дату створення замовлення.

Таблиця orders пов’язана з таблицею order_items, у якій зберігаються окремі позиції кожного замовлення. Така структура дозволяє одному замовленню містити декілька товарів із фіксацією їхньої ціни на момент покупки.

Таблиця 2.8 – Структура таблиці orders

Поле	Тип	Обмеження	Опис
id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Номер замовлення
user_id	INTEGER	FOREIGN KEY, NOT NULL	ID покупця
status	TEXT	DEFAULT ‘pending’	Статус замовлення
total	REAL	NOT NULL	Загальна сума
name	TEXT	NOT NULL	Ім’я отримувача
phone	TEXT	NOT NULL	Телефон отримувача
address	TEXT	NOT NULL	Адреса доставки
delivery	TEXT	NOT NULL	Спосіб доставки
payment	TEXT	NOT NULL	Спосіб оплати
created_at	TEXT	DEFAULT CURRENT_TIMESTAMP	Дата створення

Опис структури та взаємозв’язків основних таблиць бази даних розроблюваної інформаційної системи наведено у таблиці 2.9. База даних спроєктована таким чином, щоб забезпечити цілісність даних, підтримку зв’язків між сутностями та ефективну реалізацію бізнес-логіки інтернет-магазину.

Таблиця 2.9 – Загальна характеристика таблиць бази даних

Таблиця	Опис	Кількість полів	Зв’язки
users	Користувачі системи	8	1:N з cart, wishlist, orders
categories	Категорії товарів	5	1:N з products
products	Каталог товарів	13	N:1 з categories, 1:N з cart, wishlist, order_items
cart	Кошик користувачів	5	N:1 з users, products
wishlist	Список бажань	4	N:1 з users, products
orders	Замовлення	10	N:1 з users, 1:N з order_items
order_items	Позиції замовлень	5	N:1 з orders, products

Окрему увагу під час проєктування бази даних було приділено життєвому циклу замовлення. Для цього реалізовано систему статусів, яка дозволяє відстежувати етапи обробки замовлення – від моменту створення до завершення або повернення коштів. Діаграма станів замовлення наведена на рисунку 2.7.

Після створення замовлення воно отримує статус Pending, який означає очікування обробки менеджером. Далі замовлення переходить у стан Processing, після чого може бути відправлене користувачу (Shipped) та доставлене (Delivered). Після підтвердження отримання товару замовлення переходить у

фінальний стан Completed. У разі скасування або повернення товару система підтримує стани Cancelled, Returned та Refunded.

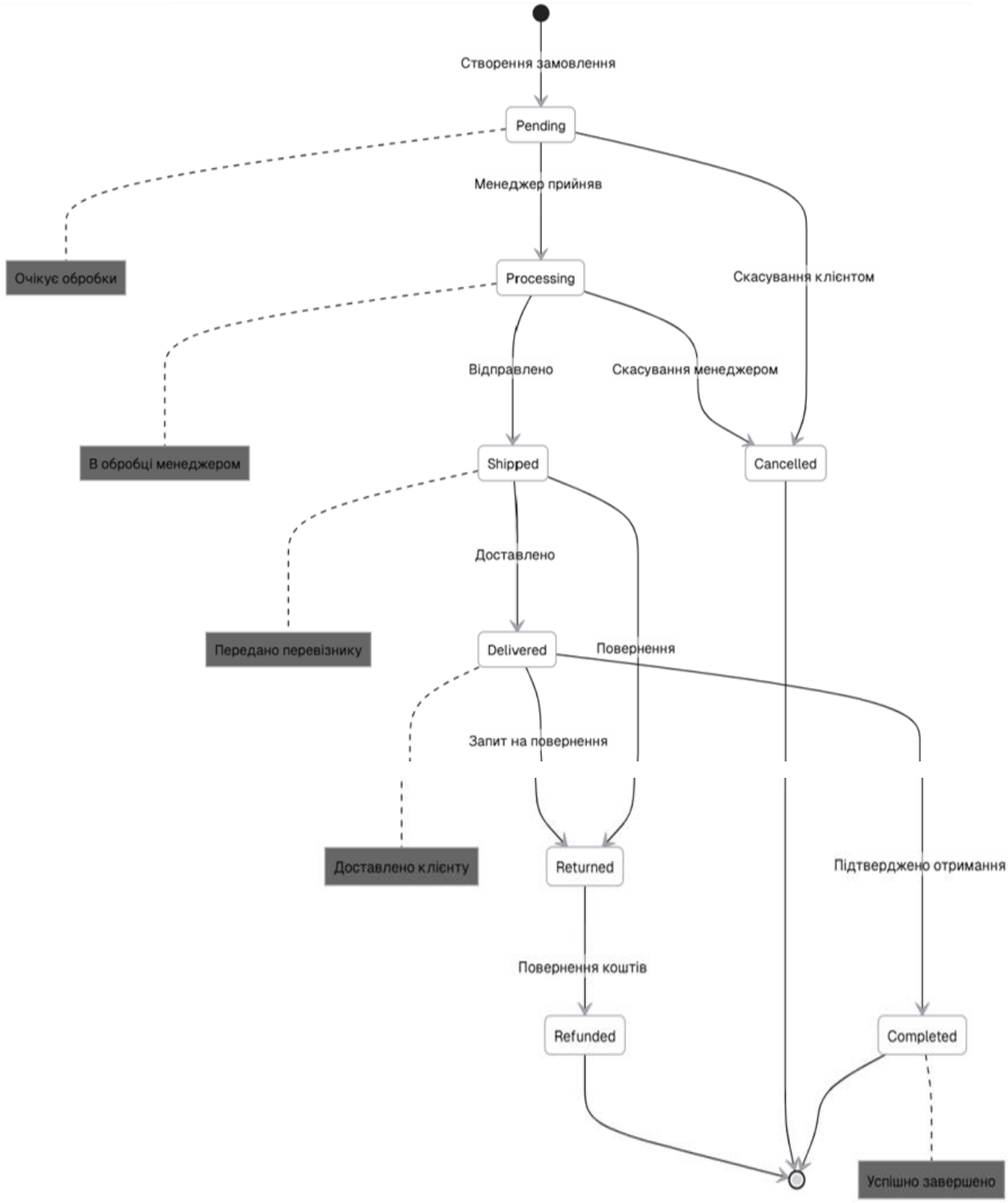


Рисунок 2.7 – Життєвий цикл замовлення

Для реалізації фізичної структури бази даних було створено SQL-схему, що містить команди CREATE TABLE для формування таблиць, зовнішніх

ключів та обмежень цілісності. У схемі також реалізовано індекси для оптимізації продуктивності запитів. Зокрема, індекси створені для полів category_id, slug, user_id, status та order_id, що дозволяє пришвидшити пошук товарів, завантаження кошика користувача та обробку замовлень.

Використання зовнішніх ключів забезпечує цілісність даних між таблицями, а застосування каскадного видалення (ON DELETE CASCADE) дозволяє автоматично видаляти пов’язані записи у кошику, списку бажань та позиціях замовлень при видаленні користувача або товару. Таким чином, спроєктована база даних забезпечує стабільну роботу вебзастосунку, підтримує масштабованість системи та дозволяє ефективно реалізувати всі основні функції інтернет-магазину.

2.4 Розробка функціоналу управління товарами (CRUD-операції)

У процесі розробки вебзастосунку інтернет-магазину було реалізовано модуль управління товарами, який забезпечує виконання повного набору CRUD-операцій (Create, Read, Update, Delete). Основним призначенням модуля є надання адміністраторам та менеджерам можливості створювати нові товари, редагувати інформацію про існуючі позиції, переглядати каталог та видаляти неактуальні записи. Архітектура модуля побудована з урахуванням принципів масштабованості, розділення відповідальності між компонентами системи та забезпечення безпечної взаємодії між клієнтською і серверною частинами застосунку.

На високому рівні модуль складається з трьох основних рівнів: інтерфейсу користувача, API-рівня та рівня взаємодії з базою даних. Інтерфейс користувача включає адміністративні сторінки для роботи з товарами, зокрема сторінку списку товарів /admin/products, сторінку створення нового товару /admin/products/new та сторінку редагування /admin/products/[slug]. API-рівень реалізує набір endpoint-ів для обробки HTTP-запитів, а рівень бази даних

забезпечує виконання SQL-операцій над таблицею products. Загальна структура модуля управління товарами наведена на рисунку 2.8.

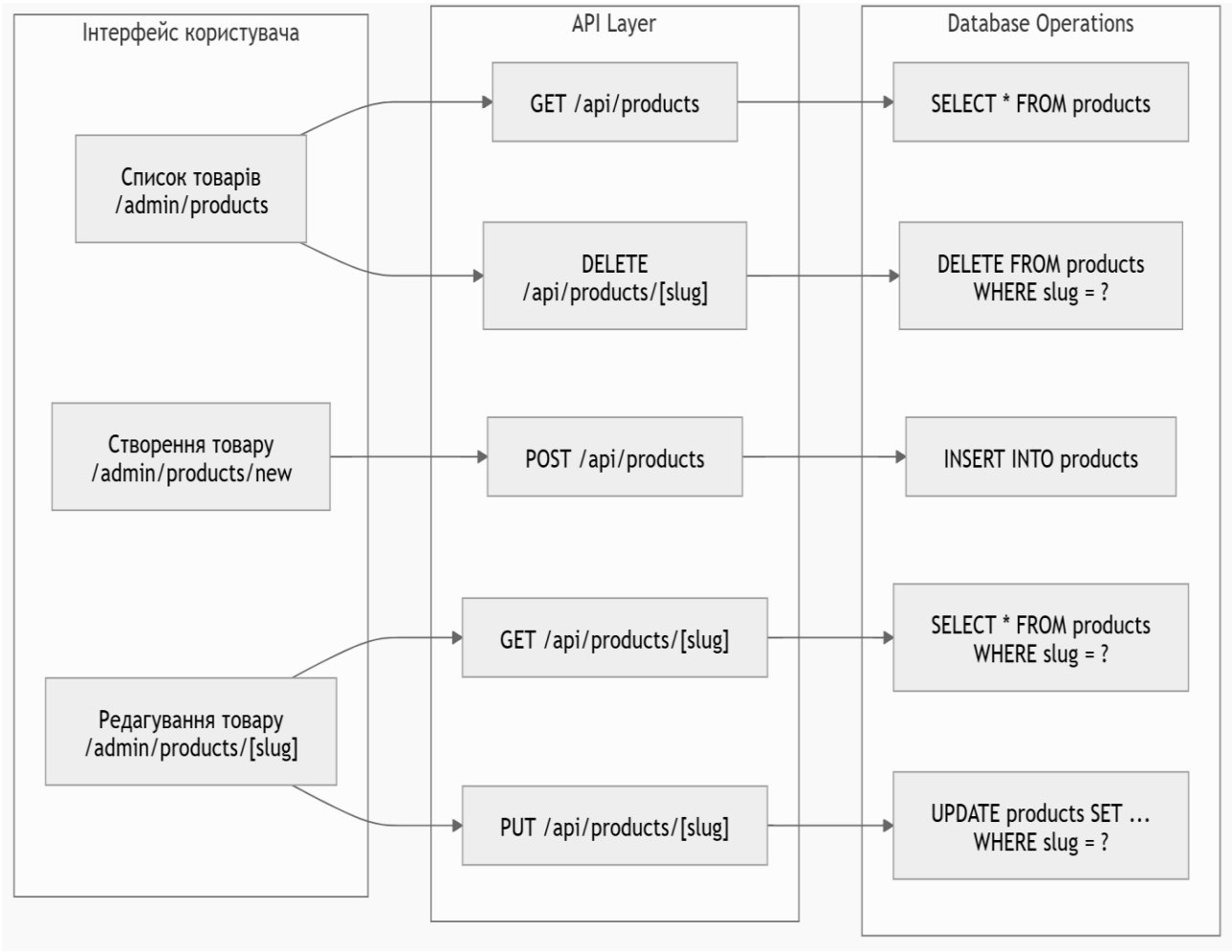


Рисунок 2.8 – Структура модуля управління товарами

На рівні інтерфейсу користувача (UI) представлено три основні сторінки адміністративної панелі:

- список товарів (/admin/products);
- створення нового товару (/admin/products/new);
- редагування існуючого товару (/admin/products/[slug]).

Ці екрани забезпечують користувачу можливість керувати товарами в системі.

Другий рівень – API Layer, який виконує проміжну роль між інтерфейсом і базою даних. Тут реалізовано стандартні CRUD-операції:

- GET – отримання списку товарів або конкретного товару;
- POST – створення нового товару;
- PUT – оновлення існуючого товару;
- DELETE – видалення товару.

Третій рівень – Database Operations, де безпосередньо виконуються SQL-запити до таблиці products, зокрема вибірка, вставка, оновлення та видалення записів.

Для реалізації CRUD-функціоналу використовуються REST API endpoint-и, кожен з яких відповідає окремій операції над сутністю товару. Доступ до створення, редагування та видалення товарів мають лише користувачі з ролями manager або admin, тоді як операції перегляду доступні всім користувачам системи. Специфікація CRUD-операцій наведена в таблиці 2.10.

Таблиця 2.10 – CRUD-операції модуля товарів

Операція	**HTTP Метод**	**Endpoint**	**Опис**	**Доступ**
Create	POST	/api/products	Створення нового товару	Manager, Admin
Read (all)	GET	/api/products	Отримання списку товарів	Всі
Read (one)	GET	/api/products/[slug]	Отримання деталей товару	Всі
Update	PUT	/api/products/[slug]	Оновлення товару	Manager, Admin
Delete	DELETE	/api/products/[slug]	Видалення товару	Manager, Admin

Операція Create (POST /api/products) використовується для створення нового товару і доступна лише ролям Manager та Admin.

Операція Read (all) (GET /api/products) забезпечує отримання списку всіх товарів і є доступною для всіх користувачів.

Операція Read (one) (GET /api/products/[slug]) дозволяє переглядати деталі конкретного товару і також є публічною.

Операція Update (PUT /api/products/[slug]) використовується для оновлення даних товару та доступна лише Manager та Admin.

Операція Delete (DELETE /api/products/[slug]) призначена для видалення товару і також обмежена ролями Manager та Admin.

Однією з ключових операцій системи є створення нового товару. Після переходу менеджера на сторінку створення товару система відображає форму введення даних, яка містить поля назви товару, ціни, категорії, опису, URL зображення та кількості товару на складі. Після натискання кнопки збереження виконується первинна валідація введених даних на стороні клієнта, а потім формується POST-запит до endpoint-а /api/products.

На серверному рівні middleware виконує перевірку JWT-токена та ролі користувача. Якщо користувач має відповідні права доступу, API створює slug товару, виконує SQL-запит INSERT INTO products та повертає результат операції. У випадку помилки валідації або відсутності авторизації система повертає повідомлення про помилку. Послідовність взаємодії між компонентами системи під час створення товару наведена на рисунку 2.9.

Діаграма відображає процес створення нового товару в адміністративній частині системи. Менеджер відкриває сторінку додавання товару, заповнює форму та ініціює збереження даних. Після цього виконується валідація введеної інформації на стороні клієнта. У разі успішної перевірки дані надсилаються до API, де відбувається перевірка авторизації користувача та його ролі. Після підтвердження прав доступу система генерує унікальний slug, зберігає товар у базі даних SQLite та повертає результат виконання операції. У випадку помилки авторизації або валідації користувач отримує відповідне повідомлення.

Після створення товару система також підтримує механізм його оновлення. Для редагування менеджер переходить на сторінку /admin/products/[slug], після чого клієнтська частина виконує GET-запит до API та отримує поточні дані товару з бази даних. Отримана інформація автоматично заповнює форму редагування. Після внесення змін користувач натискає кнопку оновлення, а система надсилає PUT-запит до endpoint-а /api/products/[slug].

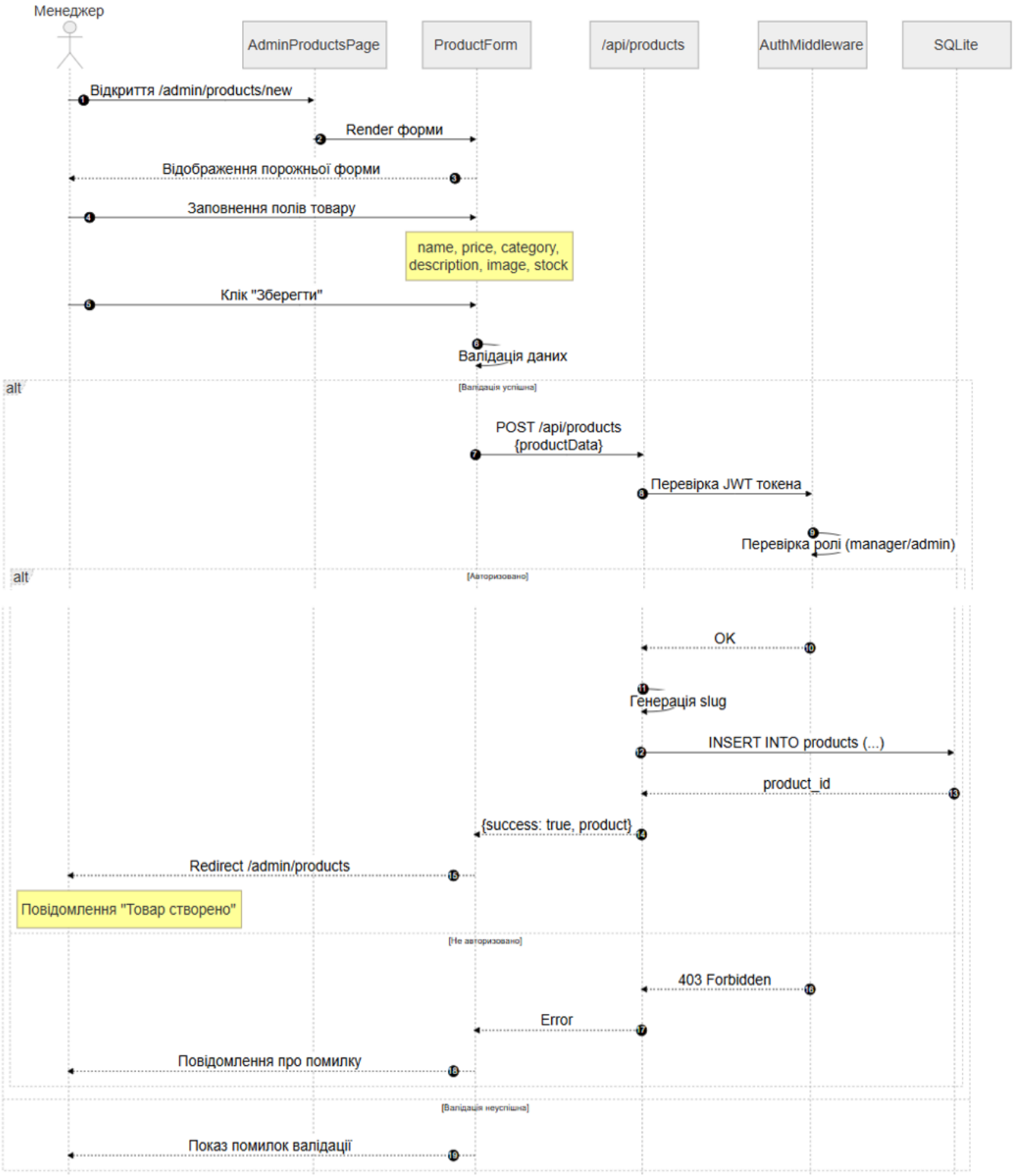


Рисунок 2.9 – Процес створення нового товару

На серверному рівні повторно виконується перевірка авторизації та ролі користувача, після чого формується SQL-запит UPDATE products SET ... WHERE slug = ?. Після успішного оновлення система повертає повідомлення про успішне

завершення операції. Діаграма послідовності процесу оновлення товару наведена на рисунку 2.10.

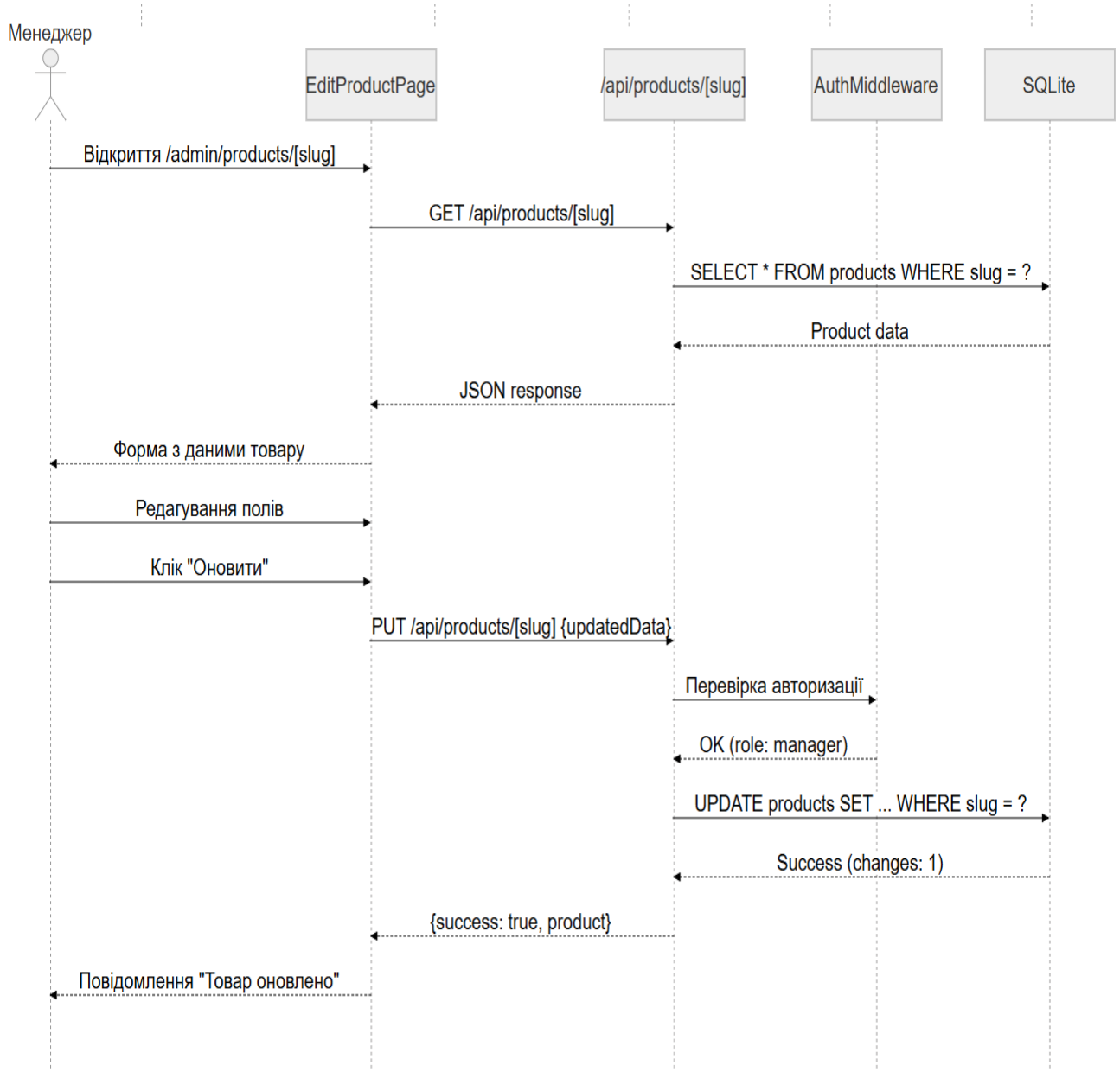


Рисунок 2.10 – Процес оновлення товару

Діаграма демонструє процес редагування існуючого товару. Менеджер відкриває сторінку редагування, після чого система отримує поточні дані товару через API-запит до бази даних. Отримана інформація відображається у формі для редагування. Після внесення змін користувач відправляє оновлені дані, які проходять перевірку авторизації. У разі успішної перевірки виконується оновлення запису в базі даних SQLite. Після завершення операції система

повертає підтвердження та повідомляє користувача про успішне оновлення товару.

Важливим елементом реалізації CRUD-функціоналу є система валідації даних (табл. 2.11). Перевірка коректності введеної інформації виконується як на стороні клієнта, так і на сервері. Це дозволяє мінімізувати ризик внесення некоректних або потенційно небезпечних даних до бази даних. Для кожного поля товару визначено окремі правила валідації, що забезпечують цілісність та консистентність інформації в системі.

Таблиця 2.11 – Правила валідації полів товару

Поле	**Тип**	**Обов'язкове**	**Правила валідації**
name	string	Так	Мін. 2 символи, макс. 200 символів
price	number	Так	> 0, макс. 2 знаки після коми
old_price	number	Ні	> price (якщо вказано)
description	string	Ні	Макс. 5000 символів
image	string	Ні	Валідний URL
stock	number	Ні	>= 0, ціле число
category_id	number	Так	Існуючий ID категорії
is_featured	boolean	Ні	true/false

Обов'язковими є ключові поля, такі як name, price та category_id, без яких неможливо створити запис товару. Назва товару повинна містити від 2 до 200 символів, ціна – бути додатною числовою величиною з точністю до двох знаків після коми, а ідентифікатор категорії має відповідати існуючому запису в системі.

Додаткові поля, такі як old_price, description, image, stock та is_featured, є необов'язковими, але також мають визначені правила: стара ціна повинна перевищувати поточну (за наявності), опис обмежений до 5000 символів, зображення має бути коректним URL, кількість товару на складі не може бути від'ємною, а поле "рекомендований товар" приймає лише логічні значення true/false.

Окрім базових CRUD-операцій, модуль управління товарами реалізує розширені механізми пошуку та фільтрації, що забезпечують більш зручну навігацію по каталогу. Під час виконання GET-запиту до endpoint /api/products

система аналізує наявність query-параметрів і на їх основі формує динамічний SQL-запит.

Користувач має можливість фільтрувати товари за категорією, ціновим діапазоном та ключовими словами, а також застосовувати сортування за ціною, назвою або датою додавання. Це дозволяє швидко звужувати вибірку товарів відповідно до заданих критеріїв і підвищує зручність роботи з великим каталогом.

Алгоритм фільтрації побудований за принципом поступового додавання умов до базового SQL-запиту. У випадку відсутності параметрів виконується стандартний запит SELECT * FROM products, який повертає повний перелік товарів. Якщо ж передані додаткові параметри, система динамічно формує умови WHERE, AND, а також опціонально додає ORDER BY для сортування та LIMIT для обмеження кількості результатів.

Такий підхід забезпечує гнучкість роботи API, зменшує навантаження на базу даних за рахунок вибіркового отримання даних і підвищує продуктивність системи при роботі з великим обсягом товарів. Алгоритм реалізації фільтрації наведений на рисунку 2.11.

Отже, реалізовано повний набір CRUD-операцій для управління товарами, що забезпечує створення, перегляд, оновлення та видалення записів у базі даних. Передбачено перевірку прав доступу, валідацію даних, а також можливості фільтрації та пошуку, що підвищує ефективність роботи з товарним каталогом.

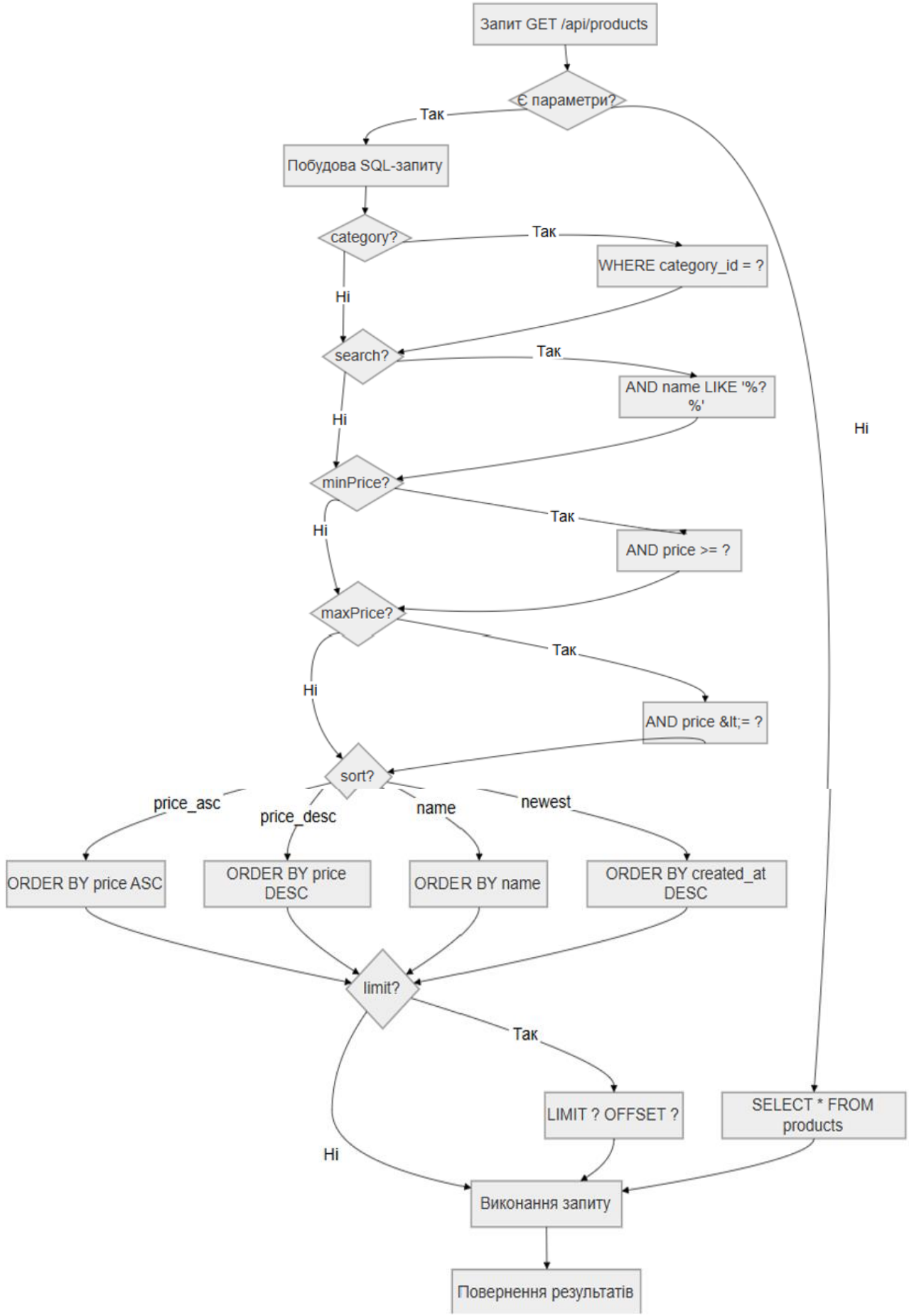


Рисунок 2.11 – Алгоритм фільтрації товарів

2.5 Реалізація особистого кабінету покупця

Особистий кабінет покупця є ключовим компонентом користувацької частини вебзастосунку, який забезпечує централізований доступ до персональної інформації, історії взаємодії з платформою та інструментів управління акаунтом. Його реалізація спрямована на підвищення зручності використання системи, персоналізацію досвіду користувача та забезпечення безпечної роботи з персональними даними.

Основна функціональність кабінету реалізована через модульну структуру, де кожен функціональний блок відповідає окремому маршруту застосунку (рис. 2.12). Центральною точкою входу є сторінка /account, яка виконує роль інформаційного дашборду та забезпечує навігацію до підрозділів системи.

Архітектура особистого кабінету включає такі логічні компоненти:

– модуль перегляду та редагування профілю користувача;
– модуль управління замовленнями;
– модуль списку бажань (wishlist);
– модуль безпеки (зміна пароля, авторизаційні дані);
– інформаційно-аналітичний дашборд користувача.

Система побудована за принципом розділення інтерфейсів (табл. 2.12):

Таблиця 2.12 – Сторінки особистого кабінету

Сторінка	URL	Опис	Функціональність
Огляд	/account	Головна сторінка кабінету	Відображення статистики користувача, останніх замовлень та персональної інформації
Замовлення	/account/orders	Історія замовлень	Перегляд списку замовлень, фільтрація за статусом, перегляд деталей замовлення
Налаштування	/account/settings	Профіль користувача	Редагування персональних даних, зміна пароля, оновлення контактної інформації
Wishlist	/wishlist	Список бажань	Додавання та видалення товарів, перенесення товарів у кошик

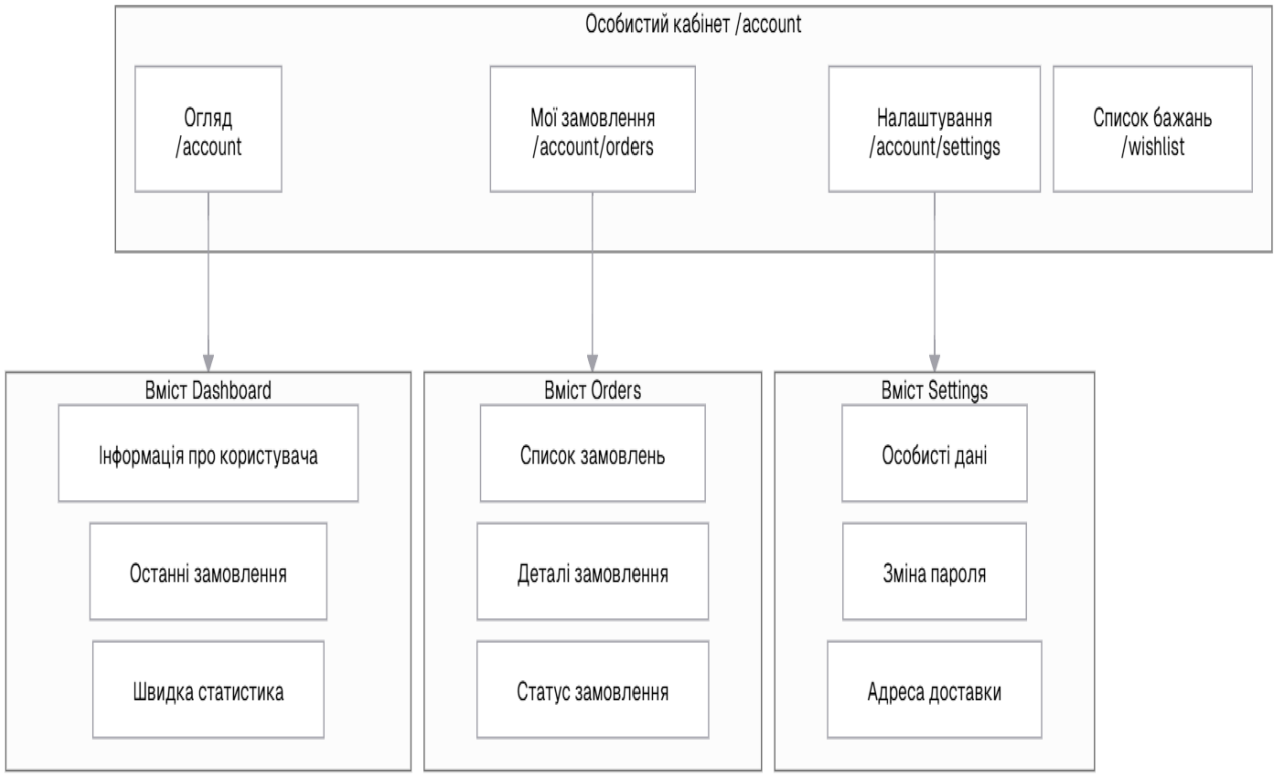


Рисунок 2.12 – Структура особистого кабінету

Дані маршрути взаємодіють із серверною частиною через REST API та використовують JWT-токен для визначення користувача. Це дозволяє ізолювати персональні дані та забезпечити захищений доступ до кожного функціонального блоку. Реалізація особистого кабінету побудована на основі компонентного підходу, де кожна сторінка є незалежним функціональним модулем, що взаємодіє з API через стандартизовані запити.

Процес авторизації користувача реалізовано через механізм JWT (JSON Web Token) та включає як серверну, так і клієнтську перевірку даних (рис. 2.13).

Алгоритм авторизації передбачає такі основні етапи:

– введення користувачем облікових даних;
– відправка запиту на сервер;
– перевірка користувача в базі даних;
– валідація пароля;
– генерація токена доступу;
– збереження токена в cookie;
– перенаправлення користувача до особистого кабінету.

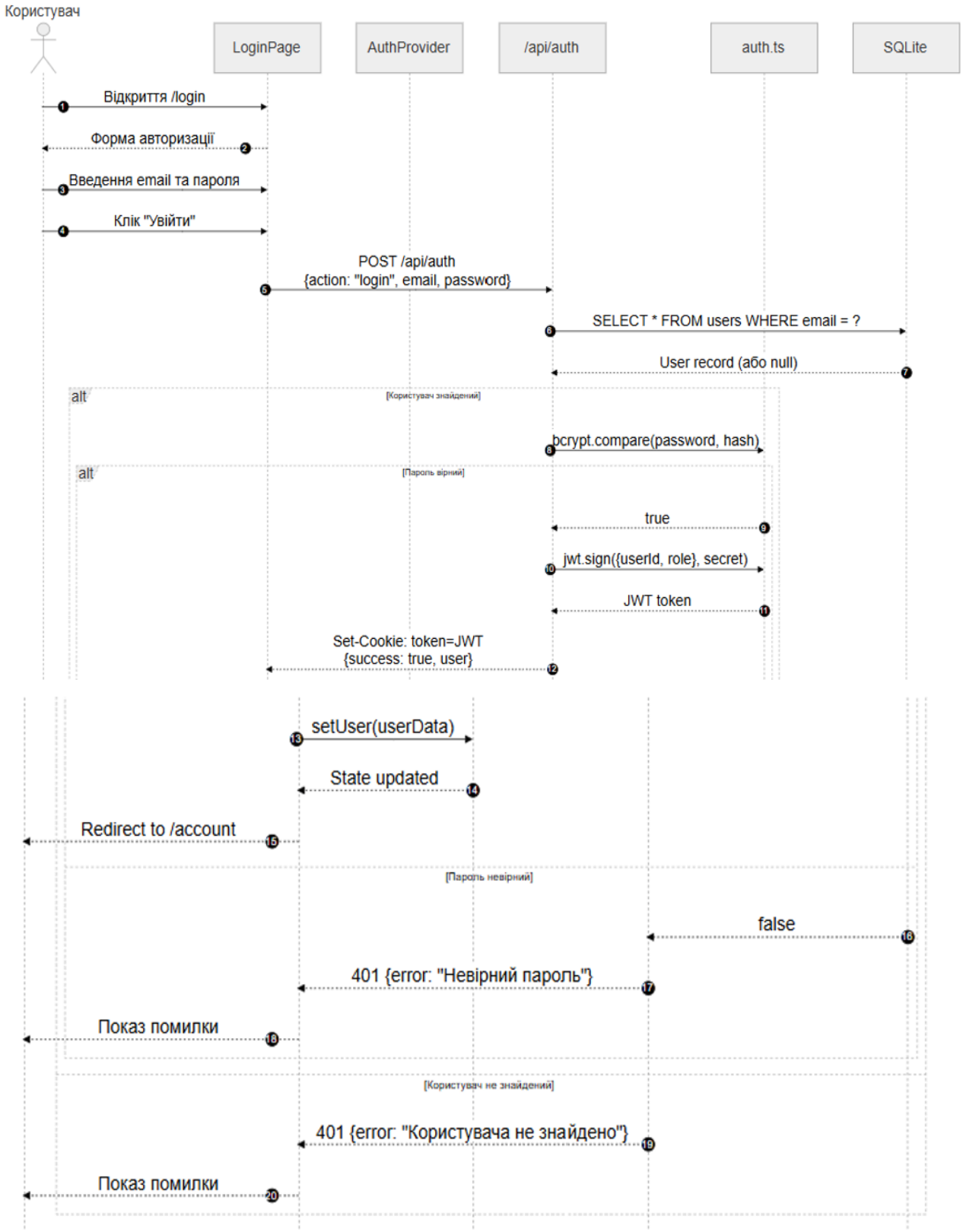


Рисунок 2.13 – Процес авторизації користувача

Взаємодія між компонентами відбувається через такі рівні:

– клієнтський інтерфейс (Login Page);
– API шар (/api/auth);
– база даних (SQL запити до таблиці users);
– сервіс хешування паролів (bcrypt);
– модуль генерації JWT.

У разі успішної авторизації користувач отримує токен, який зберігається в cookie та використовується для доступу до захищених маршрутів, зокрема /account.

Діаграма відображає повний процес входу користувача в систему та перевірки його облікових даних. Процес починається з відкриття сторінки логіну, після чого користувач вводить email і пароль та ініціює запит авторизації.

Далі дані передаються до API, де відбувається пошук користувача в базі даних за email. Якщо користувач знайдений, система виконує перевірку пароля за допомогою bcrypt. У разі успішної верифікації генерується JWT-токен, який зберігається у cookie та використовується для подальшої авторизації запитів.

Після цього користувацький стан оновлюється в застосунку, і відбувається перенаправлення до особистого кабінету. У випадку помилки (невірний пароль або відсутність користувача) система повертає відповідний HTTP-код і повідомлення про помилку, яке відображається користувачу.

Реєстрація нового користувача є аналогічним процесом, однак включає додатковий етап перевірки унікальності email та створення нового запису в базі даних (рис. 2.14).

Основні етапи реєстрації:

– введення реєстраційних даних (ім’я, email, пароль);
– клієнтська валідація;
– перевірка на сервері унікальності email;
– хешування пароля;
– створення нового користувача;
– автоматична авторизація після реєстрації.

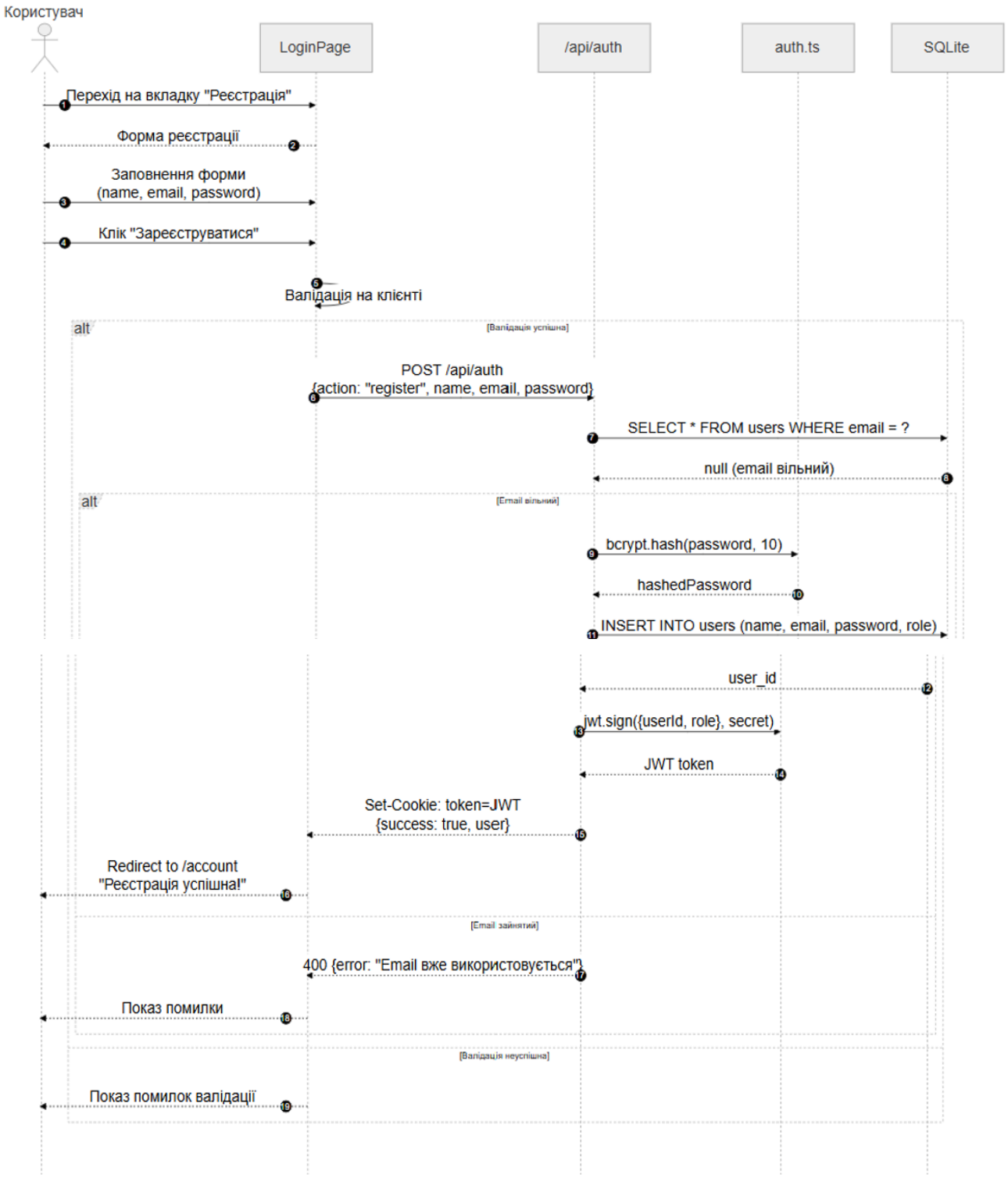


Рисунок 2.14 – Процес реєстрації нового користувача

Діаграма демонструє процес створення нового облікового запису в системі. Користувач переходить на вкладку реєстрації, заповнює форму та надсилає дані для створення акаунту.

На сервері виконується перевірка коректності введених даних і перевірка унікальності email. Якщо email не зайнятий, пароль хешується за допомогою bcrypt, після чого створюється новий запис у таблиці користувачів.

Далі система генерує JWT-токен, який зберігається в cookie, і автоматично авторизує користувача з перенаправленням до особистого кабінету. У випадку помилки (наприклад, якщо email вже використовується або валідація не пройдена) користувач отримує повідомлення про проблему.

У разі успішного виконання операції система:

– створює запис у таблиці users;
– генерує JWT-токен;
– автоматично виконує вхід користувача;
– перенаправляє його до сторінки /account.

У випадку помилки (наприклад, зайнятий email) система повертає відповідний HTTP-код (400) та повідомлення про помилку, яке відображається в інтерфейсі.

Функціонал управління замовленнями забезпечує користувачу доступ до повної історії покупок із можливістю перегляду детальної інформації про кожне замовлення.

Процес отримання історії замовлень включає (рис. 2.15):

– визначення користувача за JWT-токеном;
– запит до таблиці orders із фільтрацією за user_id;
– отримання пов’язаних товарів через таблицю order_items;
– об’єднання даних продуктів та замовлень;
– формування структурованої відповіді API.

Діаграма показує процес отримання та відображення історії замовлень користувача. Після відкриття сторінки замовлень система надсилає запит до API, використовуючи ідентифікатор користувача, отриманий із JWT-токена.

На сервері виконується вибірка всіх замовлень користувача з бази даних із сортуванням за датою створення. Для кожного замовлення додатково завантажуються пов’язані товари через зв’язану таблицю order_items.

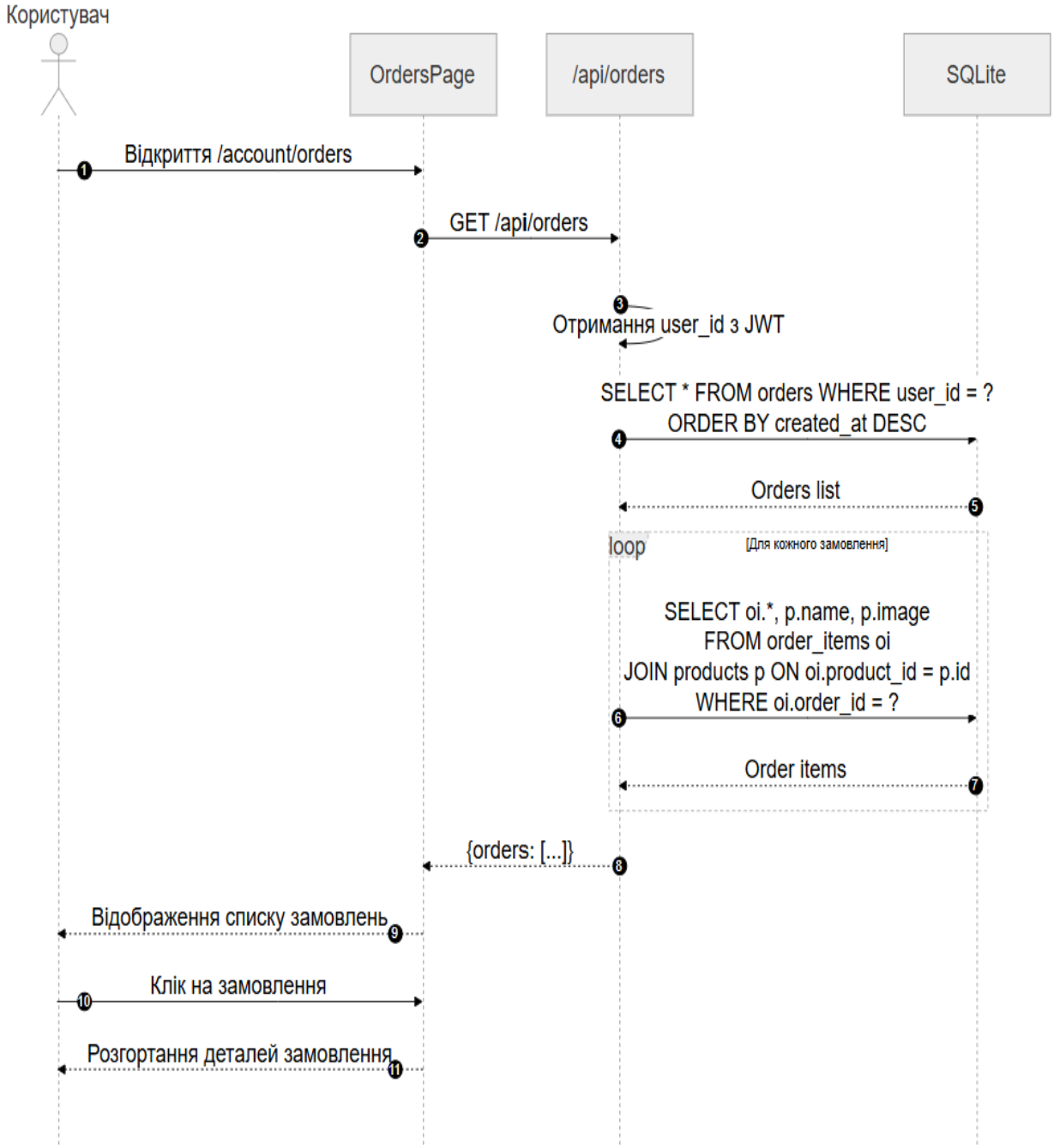


Рисунок 2.15 – Процес перегляду історії замовлень

Після отримання даних система формує відповідь і відображає список замовлень у інтерфейсі. Користувач може переглядати деталі кожного замовлення шляхом розгортання відповідного елемента.

Система виконує:

– SQL JOIN операції між таблицями orders, order_items та products;

– сортування замовлень за датою створення;

– повернення агрегованих даних у форматі JSON.

Інтерфейс дозволяє користувачу розгортати кожне замовлення для перегляду складу, вартості та статусу доставки.

Функціонал списку бажань (wishlist) призначений для збереження товарів, які користувач планує придбати в майбутньому. Він реалізується як окремий модуль, що взаємодіє з таблицею користувацьких вподобань у базі даних.

Основні операції:

– додавання товару до wishlist;

– видалення товару;

– перевірка наявності товару у списку;

– перенесення товару в кошик.

Система забезпечує синхронізацію між клієнтським інтерфейсом і сервером у режимі реального часу через API-запити. Це дозволяє підтримувати актуальний стан списку незалежно від пристрою користувача.

Редагування профілю користувача реалізовано як окремий захищений модуль, доступ до якого можливий лише після авторизації. Оновлення профілю виконується через PUT-запит до API, після чого сервер:

– перевіряє валідність даних;

– оновлює запис у базі даних;

– при необхідності перевидає JWT-токен;

– повертає оновлені дані користувача.

Цей підхід забезпечує цілісність даних і безпеку персональної інформації користувача.

Отже, особистий кабінет покупця забезпечує доступ до персональних даних, історії замовлень, налаштувань профілю та списку бажань. Реалізований функціонал дозволяє керувати акаунтом і переглядати інформацію про покупки. Використання JWT-автентифікації забезпечує безпечну взаємодію із системою.

2.6 Обґрунтування вибору технологій розробки

Для реалізації інтернет-магазину було обрано сучасний технологічний стек, який забезпечує баланс між продуктивністю, масштабованістю, зручністю розробки та простотою підтримки. Основний акцент зроблено на використанні перевірених інструментів, які дозволяють швидко створювати високонавантажені та надійні вебзастосунки.

Базовим фреймворком для розробки обрано Next.js, який поєднує можливості серверного та клієнтського рендерингу [13]. Це дозволяє покращити SEO-оптимізацію, прискорити початкове завантаження сторінок і реалізувати серверну логіку без окремого бекенд-сервера завдяки API Routes. Додатковою перевагою є підтримка сучасної архітектури App Router, що спрощує структуру проєкту та підвищує його масштабованість.

Для побудови інтерфейсу використано React, який забезпечує компонентний підхід до розробки та дозволяє повторно використовувати UI-елементи. У поєднанні з TypeScript це підвищує надійність коду, зменшує кількість помилок і покращує читабельність проєкту. TypeScript забезпечує статичну типізацію, що особливо важливо для великих застосунків з великою кількістю взаємодій між компонентами та API [16].

Для стилізації інтерфейсу обрано Tailwind CSS, який дозволяє швидко створювати адаптивний дизайн без написання окремих CSS-файлів [17]. Utility-first підхід значно пришвидшує розробку та забезпечує гнучкість у налаштуванні зовнішнього вигляду застосунку. У поєднанні з бібліотекою shadcn/ui це дозволяє отримати сучасний, консистентний та легко розширюваний інтерфейс.

Як систему керування базою даних використано SQLite, що є оптимальним рішенням для невеликих та середніх проєктів [15]. Вона не потребує окремого серверного розгортання, легко інтегрується у застосунок та забезпечує високу швидкість роботи з локальними даними. Це особливо зручно для serverless-архітектури, де важлива простота розгортання та мінімальні вимоги до інфраструктури.

Для реалізації серверної логіки та API використано вбудовані можливості Next.js, що дозволило уникнути створення окремого бекенд-застосунку. Аутентифікація користувачів реалізована за допомогою JWT-токенів, що забезпечує безпечну передачу даних та контроль доступу. Для захисту паролів використано bcrypt, який забезпечує надійне хешування та захист облікових даних.

Для тестування застосунку застосовано Jest [18] для модульного тестування та Playwright [19] для end-to-end тестів, що дозволяє перевіряти як окремі функції, так і повні сценарії взаємодії користувача з системою. Це підвищує стабільність і зменшує ризик появи критичних помилок у продакшені.

Збірки та розгортання реалізована за допомогою Visual Code [20].

Отже, обраний технологічний стек забезпечує високу продуктивність, сучасну архітектуру, простоту підтримки та можливість подальшого масштабування системи інтернет-магазину.

2.7 Висновки

У другому розділі здійснено розробку інформаційної системи інтернет-магазину з особистим кабінетом користувача. Проведено аналіз вимог до системи, що дозволило визначити основні функціональні можливості та ролі користувачів.

Спроєктовано архітектуру вебзастосунку та структуру бази даних, що забезпечує ефективне зберігання та обробку інформації про товари, користувачів і замовлення. Реалізовано основні CRUD-операції для управління товарами, а також механізми фільтрації та пошуку.

Окрему увагу приділено реалізації особистого кабінету користувача, включаючи функції авторизації, роботи з кошиком, замовленнями та вибраними товарами. Обґрунтовано вибір сучасного технологічного стеку, який забезпечує продуктивність, масштабованість і зручність подальшої підтримки системи.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ

3.1 Реалізація серверної частини системи

Серверна частина вебсистеми EduQuest реалізована на основі Next.js API Routes, що дозволяє будувати серверну логіку у форматі serverless-функцій без необхідності створення окремого бекенд-сервера. Такий підхід забезпечує тісну інтеграцію клієнтської та серверної частин у межах одного проєкту, спрощує розгортання системи та підвищує її масштабованість.

Загальна архітектура серверної частини системи представлена на рис. 3.1.

Схема демонструє багаторівневу структуру взаємодії компонентів, яка включає клієнтську частину, API Routes, middleware-шар, бізнес-логіку та рівень доступу до бази даних. Запити від клієнта надходять до API Routes, після чого проходять через middleware, де виконується перевірка авторизації, валідація даних та обробка помилок. Далі запит передається до бізнес-логіки, яка реалізує основні функціональні модулі системи, зокрема роботу з користувачами, товарами, кошиком та замовленнями. Завершальним етапом є взаємодія з базою даних SQLite через SQL-запити, які виконуються через уніфікований Data Access Layer. Модуль аутентифікації забезпечує безпечну реєстрацію користувачів, їх авторизацію та управління сесіями за допомогою JWT-токенів. Його реалізація базується на використанні бібліотек jsonwebtoken та bcryptjs, що дозволяє забезпечити захищене зберігання паролів та перевірку підписаних токенів. Фрагмент реалізації модуля аутентифікації наведено у додатку А.1.

У межах даного модуля реалізовано генерацію JWT-токена, його верифікацію, а також функції хешування та перевірки паролів. Генерація токена здійснюється на основі ідентифікатора користувача, email та ролі, після чого токен підписується секретним ключем і зберігається в cookies. Це дозволяє реалізувати stateless-авторизацію без зберігання сесій на сервері.

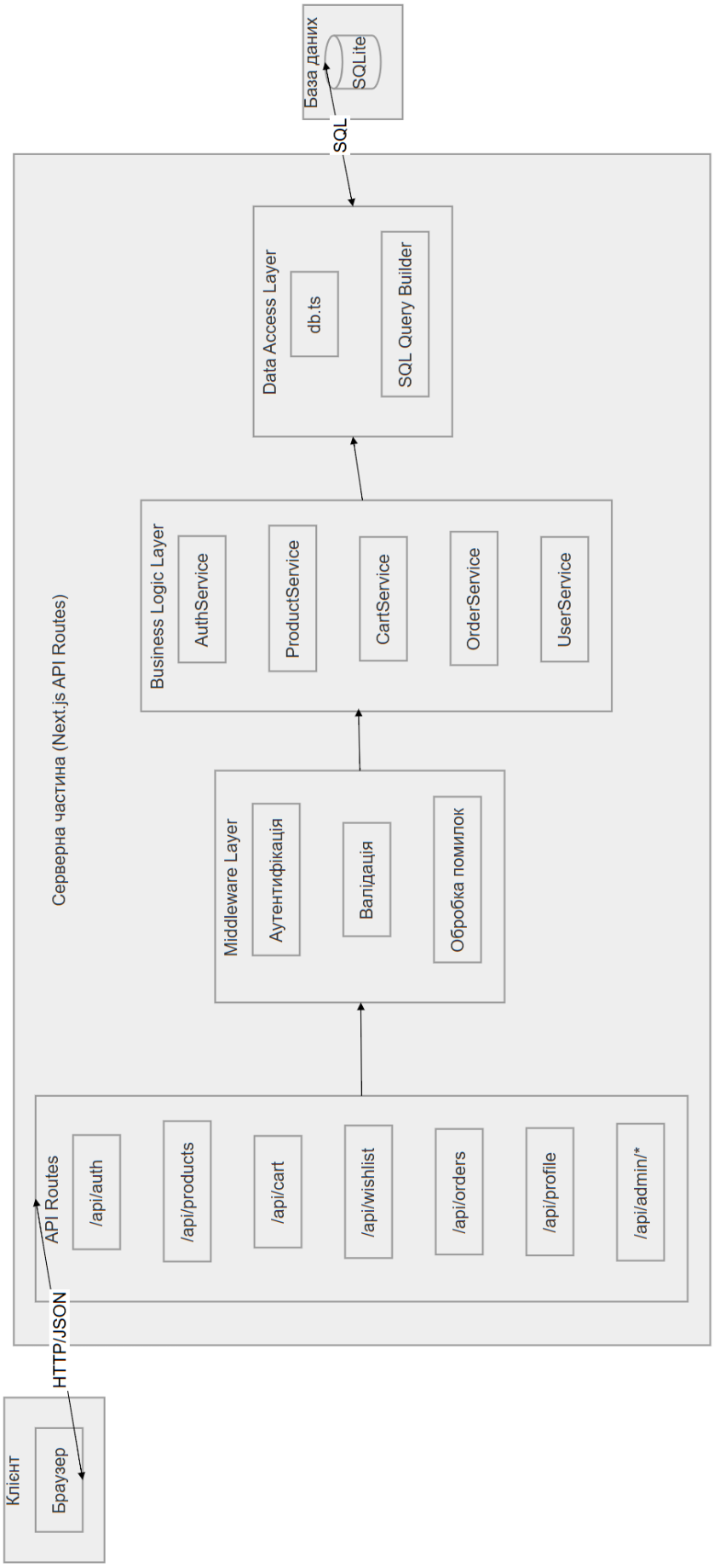


Рисунок 3.1 – Архітектура серверної частини системи

Процес аутентифікації користувача представлено на рис. 3.2.

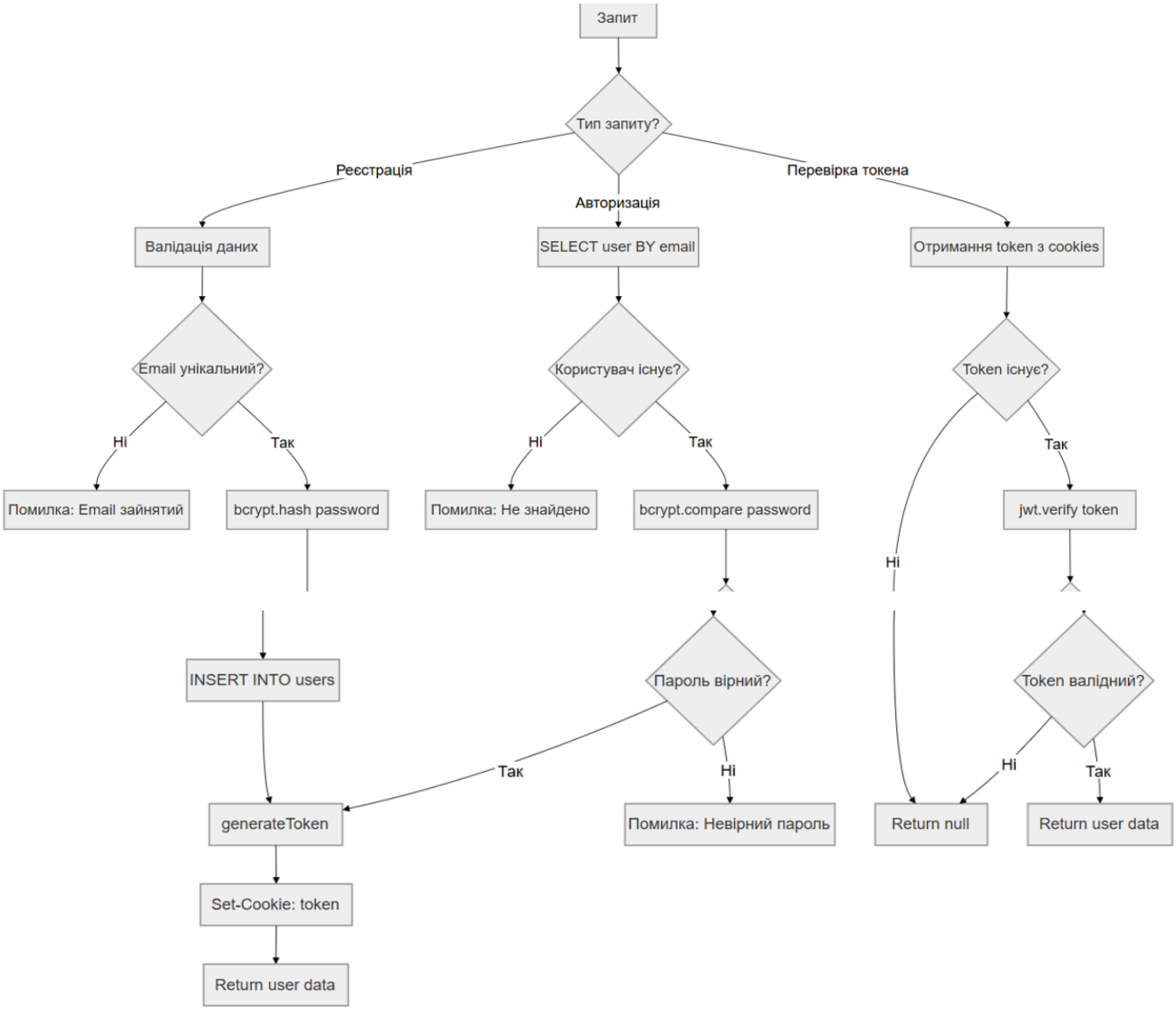


Рисунок 3.2 – Процес аутентифікації користувача

Діаграма ілюструє повний цикл взаємодії користувача із системою: від реєстрації та перевірки унікальності email до генерації JWT-токена та його подальшої перевірки при кожному запиті до захищених маршрутів API.

API для роботи з товарами забезпечує функціонал отримання, фільтрації, сортування та створення товарних позицій. Основним завданням цього модуля є ефективна обробка великих обсягів даних із мінімальним навантаженням на клієнтську частину. Реалізацію API endpoint для товарів наведено у додатку А.2.

GET-метод дозволяє отримувати список товарів із підтримкою фільтрації за категорією, пошуку за назвою, сортуванням та пагінацією. Формування SQL-запиту відбувається динамічно залежно від параметрів запиту, що дозволяє гнучко керувати вибіркою даних.

POST-метод використовується для створення нових товарів і доступний лише користувачам із ролями manager та admin. Перед збереженням товару в базу даних виконується генерація slug, який використовується для SEO-оптимізованих URL-адрес.

Логіка роботи кошика користувача представлена на рис. 3.3.

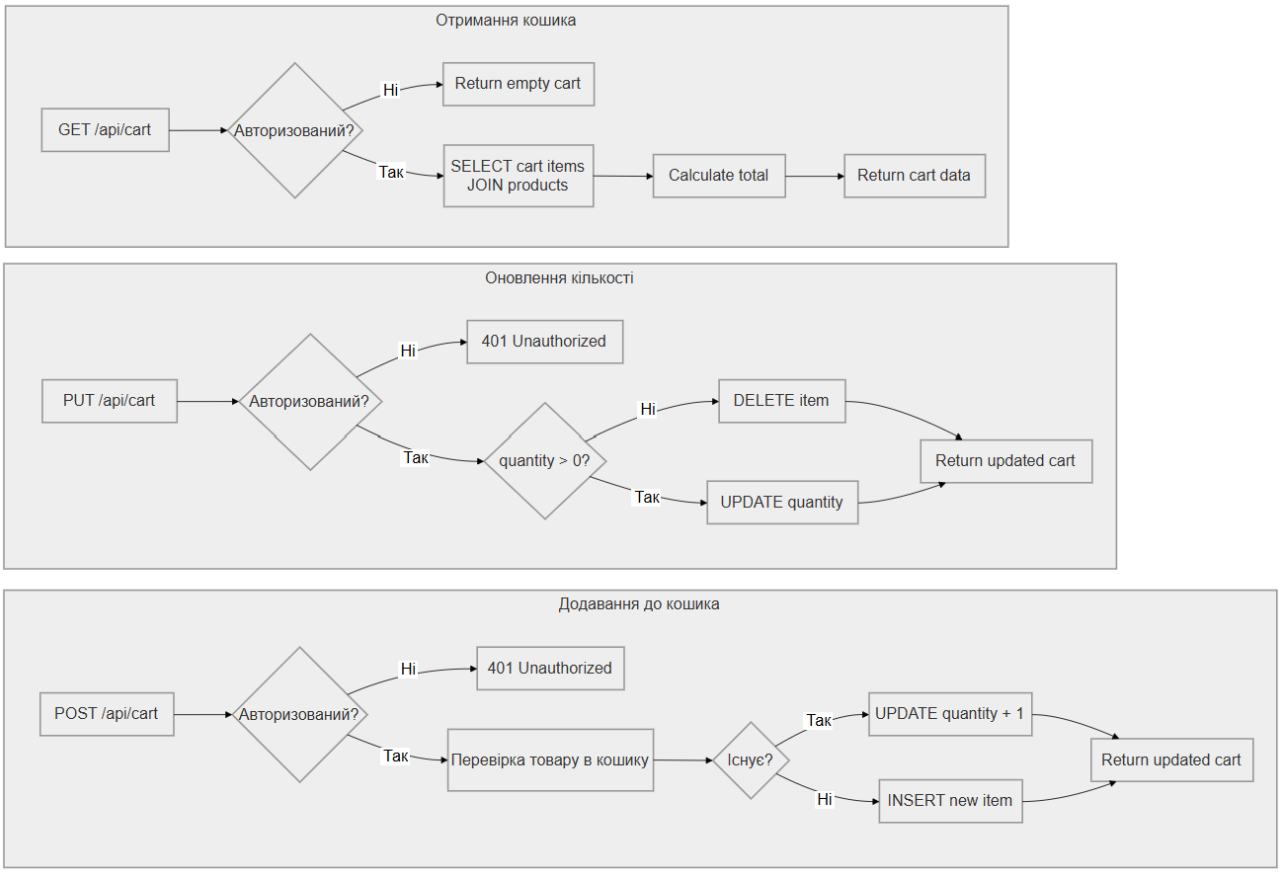


Рисунок 3.3 – Логіка роботи модуля кошика

Діаграма відображає основні сценарії взаємодії користувача з кошиком: додавання товарів, оновлення їх кількості, отримання списку позицій та видалення товарів. Реалізацію API кошика наведено у додатку А.3.

У межах цього модуля реалізовано повний цикл роботи з кошиком користувача. GET-метод повертає список товарів із розрахунком загальної вартості, POST-метод відповідає за додавання товарів до кошика або збільшення їх кількості, а DELETE-метод видаляє товар із кошика.

Процес створення замовлення представлено на рис. 3.4.

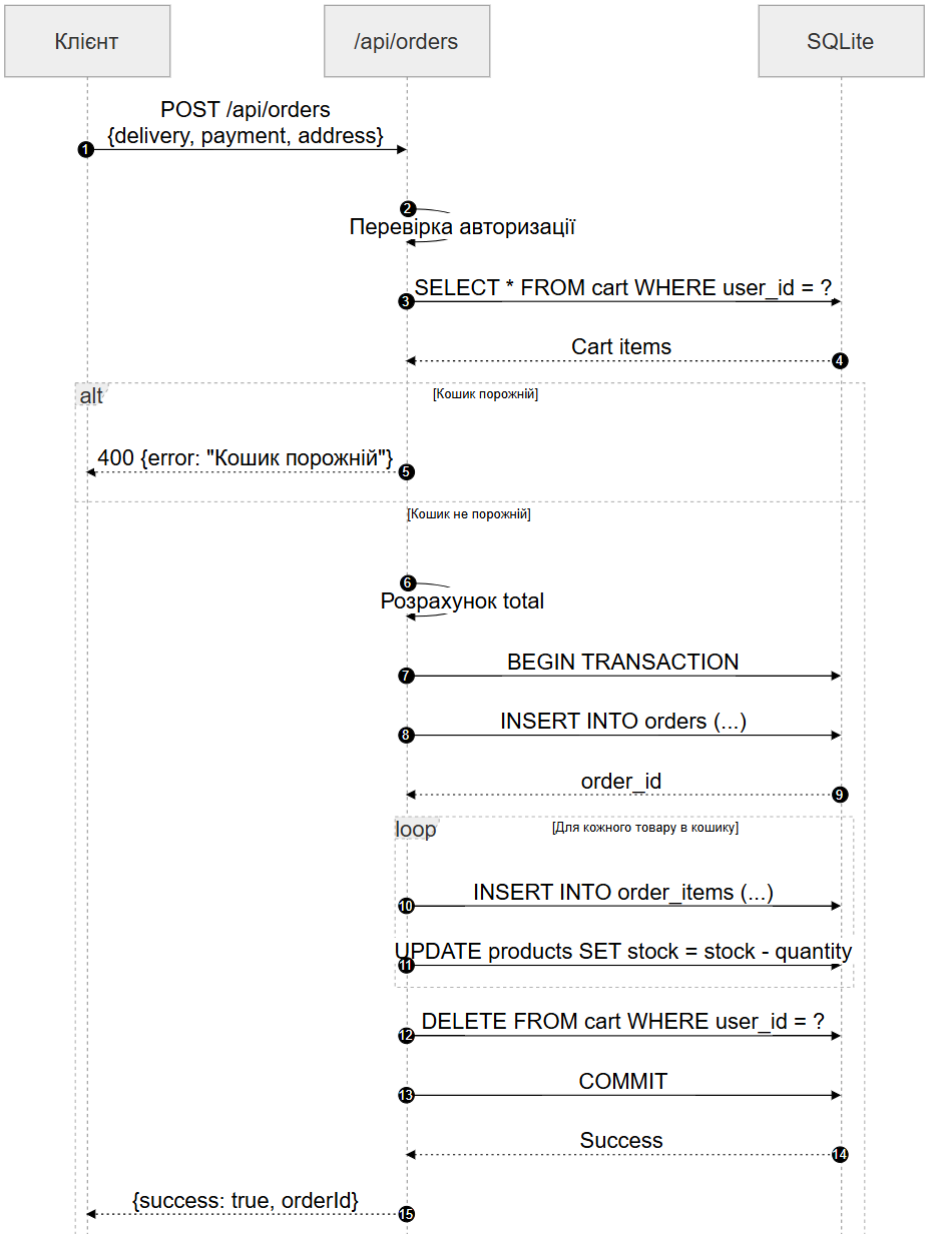


Рисунок 3.4 – Процес створення замовлення користувача

Діаграма демонструє послідовність дій від перевірки авторизації користувача до створення запису замовлення, додавання позицій у order_items,

оновлення залишків товарів та очищення кошика. Використання транзакції забезпечує цілісність даних та виключає часткове виконання операцій. Реалізація створення замовлення наведена в додатку А.4.

Отже, серверна частина системи реалізує багаторівневу архітектуру з чітким розподілом відповідальності між модулями, що забезпечує безпеку, масштабованість та простоту подальшого розвитку проєкту.

3.2 Розробка клієнтського інтерфейсу

Клієнтська частина системи реалізована на основі компонентного підходу React із використанням Next.js як основного фреймворку для рендерингу інтерфейсу та маршрутизації. Така архітектура забезпечує поділ інтерфейсу на незалежні перевикористовувані компоненти, що підвищує масштабованість і зручність підтримки проєкту. Центральним принципом побудови є ієрархічна структура компонентів, де кожен рівень відповідає за окрему функціональну область системи: сторінки, бізнес-компоненти, спільні UI-компоненти та контексти глобального стану.

Глобальний стан у системі реалізовано за допомогою Context API, що дозволяє централізовано керувати даними користувача, кошика та списку бажань без необхідності надмірного пропс-дрилінгу. Основними провайдерами виступають AuthProvider, CartProvider та WishlistProvider, які інтегруються у кореневий layout застосунку. Завдяки цьому забезпечується синхронізація стану між різними сторінками та компонентами інтерфейсу в режимі реального часу.

Структура клієнтської частини включає сторінки різних рівнів доступу: публічні (каталог, сторінка товару, контакти), авторизовані (кошик, оформлення замовлення, особистий кабінет) та адміністративні модулі (керування товарами, замовленнями та користувачами). Такий поділ забезпечує логічну сегментацію функціоналу та спрощує контроль доступу.

Архітектура клієнтської частини системи представлена на діаграмі (рис. 3.5).

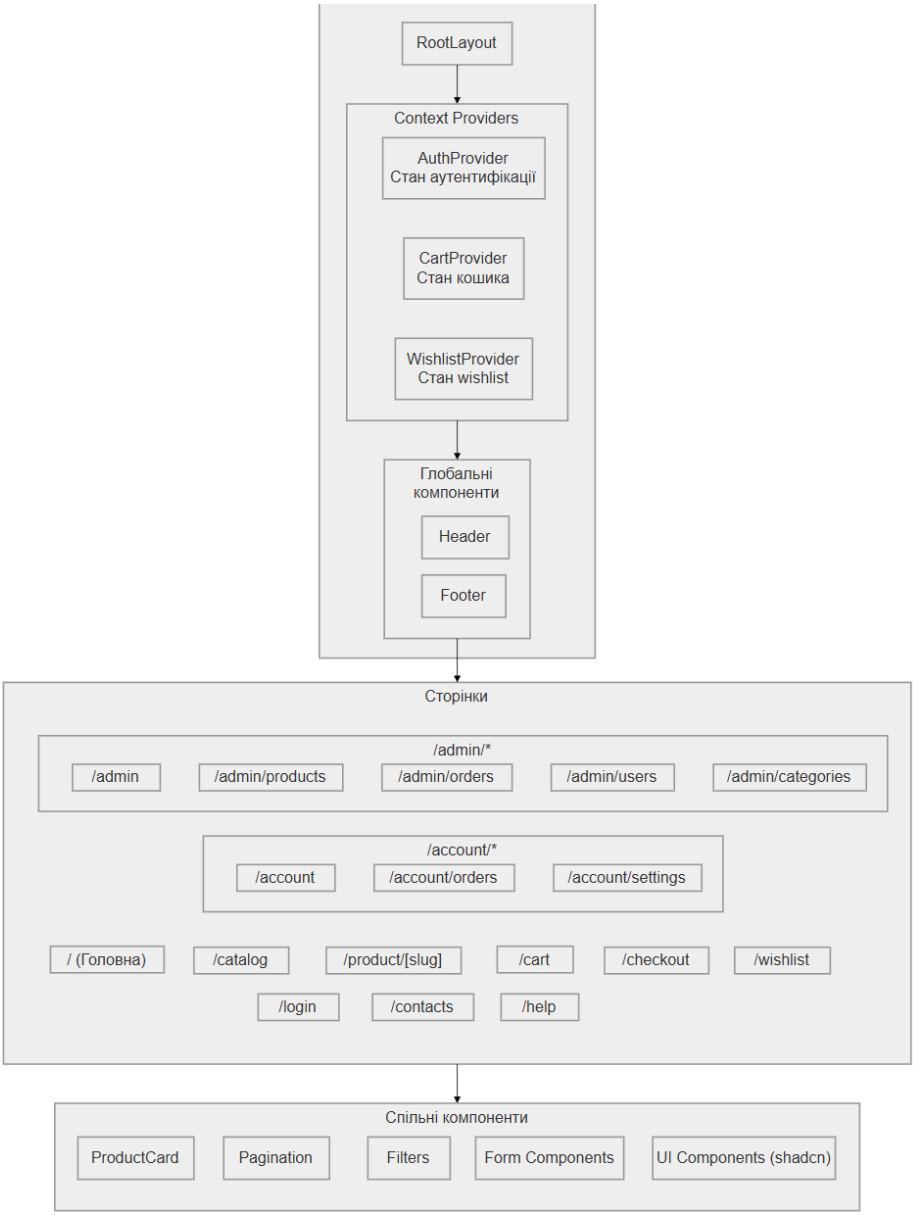


Рисунок 3.5 – Архітектура клієнтської частини

Клієнтський рівень безпосередньо взаємодіє з серверною частиною через REST API-запити у форматі HTTP/JSON, отримуючи та відправляючи дані для подальшого відображення. Додатково застосовується компонентна бібліотека shadcn/ui, яка забезпечує уніфікований дизайн інтерфейсу та прискорює розробку за рахунок готових UI-елементів.

Одним із ключових механізмів клієнтської частини є використання контекстів для управління глобальним станом. Це дозволяє уникнути надмірної вкладеності компонентів та забезпечує доступ до спільних даних з будь-якого рівня застосунку. Контекст автентифікації реалізовано через AuthProvider, який відповідає за збереження інформації про поточного користувача, виконання операцій входу, реєстрації та виходу із системи. При ініціалізації застосунку виконується перевірка активної сесії через API-запит до серверної частини, що дозволяє автоматично відновлювати стан авторизації без повторного входу користувача. Реалізація AuthProvider наведена у лістингу (додаток Б.1).

Контекст кошика реалізовано через CartProvider (додаток Б.2), який забезпечує централізоване управління товарами у кошику користувача. При зміні стану користувача відбувається автоматичне завантаження актуальних даних кошика з API, що гарантує актуальність відображуваної інформації. У межах CartProvider реалізовано основні операції: додавання товару до кошика, видалення товару, оновлення кількості та повне очищення кошика. Після кожної операції виконується повторне отримання даних з серверної частини для синхронізації стану інтерфейсу з базою даних.

Одним із ключових UI-компонентів системи є ProductCard, який відповідає за відображення товару в каталозі. Він поєднує в собі візуальну інформацію про продукт, інтерактивні елементи та логіку взаємодії з кошиком і списком бажань. Структура компонента включає зображення товару, назву, рейтинг, ціну, інформацію про знижку та кнопки взаємодії. Додатково реалізовано інтерактивні стани, такі як додавання товару до кошика та перемикання стану «у кошику», що покращує користувацький досвід.

Структура компонента ProductCard представлена на діаграмі (рис. 3.6).

Логіка компонента базується на використанні глобальних контекстів useAuth, useCart та useWishlist, що дозволяє синхронізувати дії користувача з серверною частиною без перезавантаження сторінки. Наприклад, при додаванні товару до кошика виконується API-запит, після чого оновлюється локальний стан кошика. Реалізація компонента наведена у лістингу (додаток Б.3).

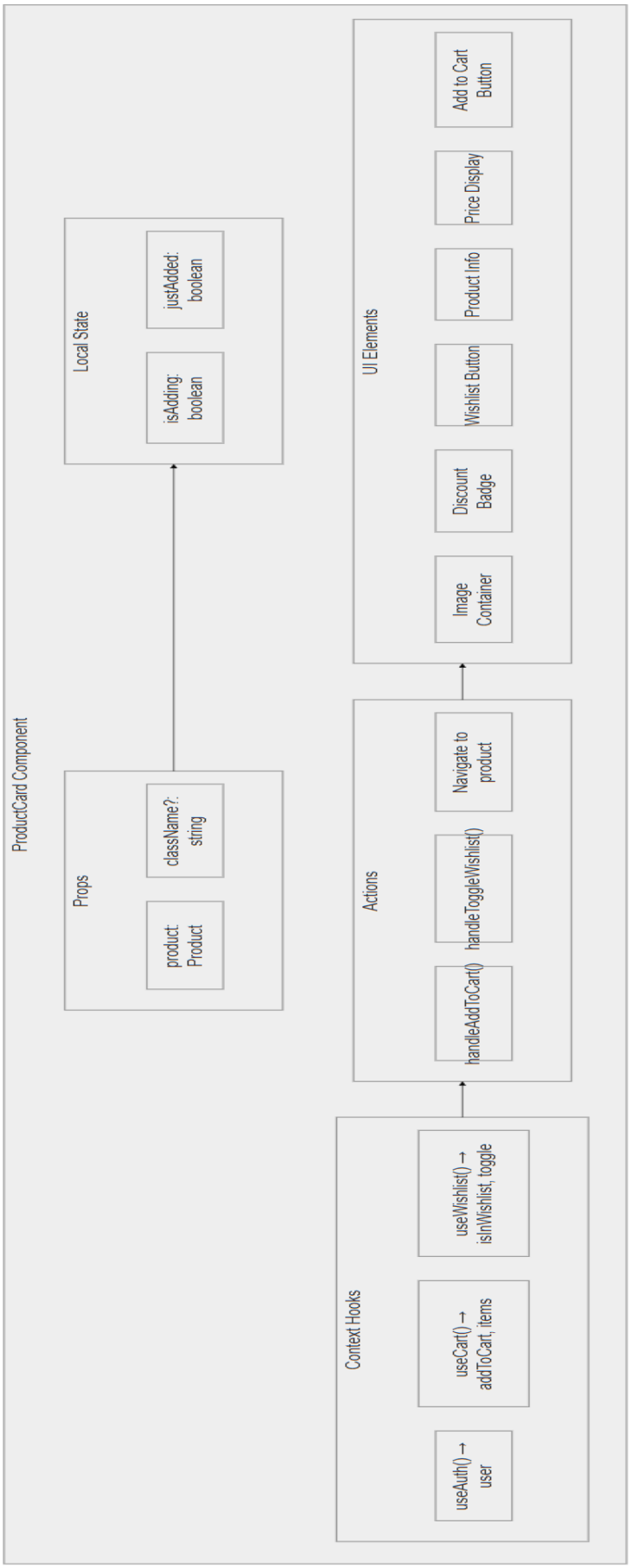


Рисунок 3.6 – Структура компонента ProductCard

Для забезпечення єдиного стилю інтерфейсу використовується дизайн-система на основі бібліотеки shadcn/ui та Tailwind CSS. Це дозволяє стандартизувати вигляд кнопок, форм, карток, модальних вікон та інших елементів інтерфейсу.

Основні UI-компоненти включають Button, Card, Input, Badge, Dialog та інші елементи, які використовуються в усіх частинах системи. Такий підхід забезпечує консистентність інтерфейсу та зменшує час розробки нових сторінок.

Особливу увагу приділено адаптивності інтерфейсу. Система підтримує різні типи пристроїв – від мобільних телефонів до великих десктопних екранів. Адаптивна сітка каталогу автоматично змінює кількість колонок залежно від розміру екрана, що представлено у лістингу.

Отже, клієнтська частина системи забезпечує високу інтерактивність, швидкодію та зручність використання завдяки компонентній архітектурі, централізованому управлінню станом та сучасним підходам до побудови UI.

3.3 Інтеграція модулів управління товарами та користувацького кабінету

Інтеграція модулів управління товарами та користувацького кабінету в системі реалізована за принципом багаторівневої архітектури, що забезпечує чітке розділення відповідальностей між клієнтським інтерфейсом, шаром управління станом та API-рівнем (рис. 3.7). Такий підхід дозволяє ефективно синхронізувати дані між різними частинами застосунку та підтримувати їх актуальність у режимі реального часу.

На рівні користувацького інтерфейсу функціонують основні сторінки системи: каталог товарів, сторінка окремого товару, кошик, оформлення замовлення та особистий кабінет користувача. Адміністративна частина представлена окремою адмін-панеллю, яка забезпечує управління товарами, замовленнями, користувачами та категоріями.

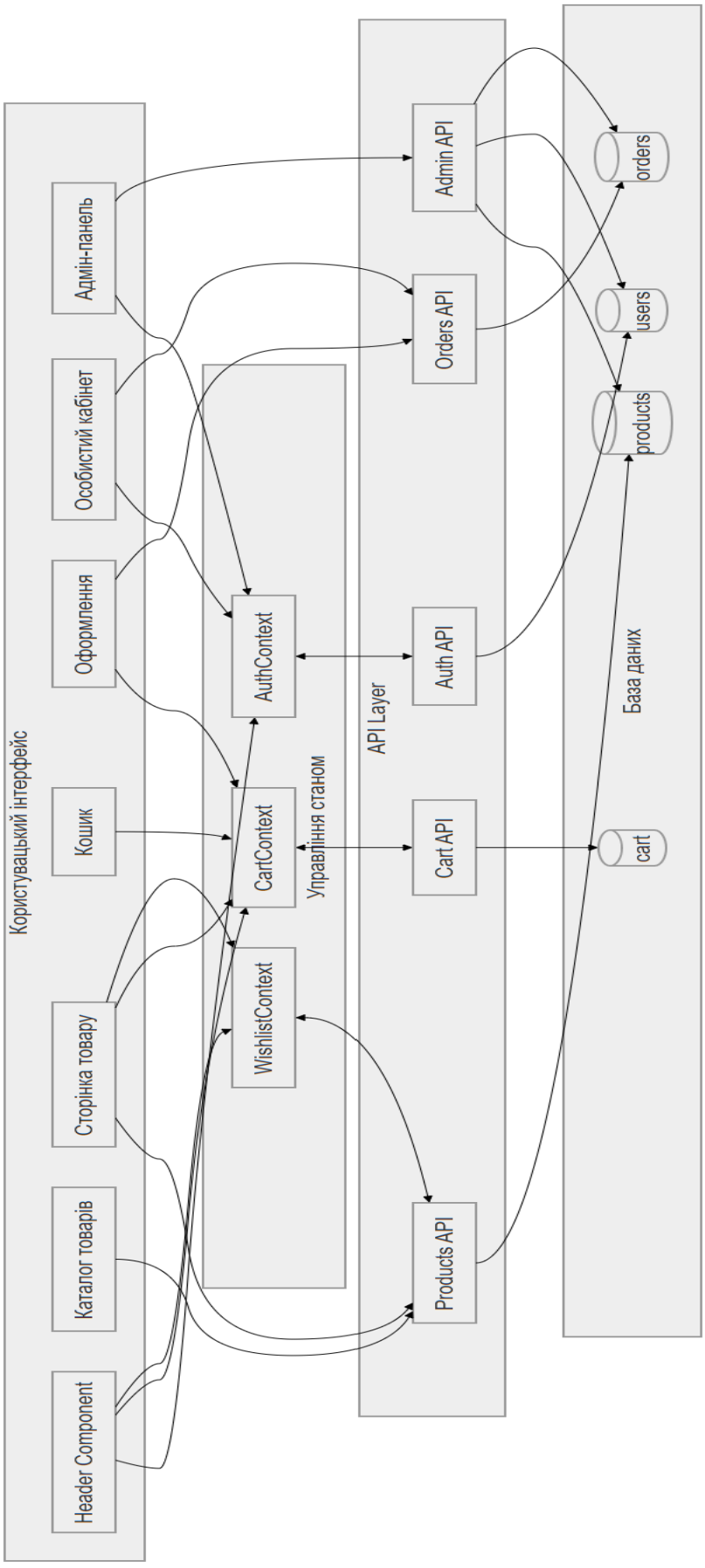


Рисунок 3.7 – Інтеграція модулів системи

Між інтерфейсом та серверною частиною знаходиться шар управління станом, реалізований через контексти React Context API. У системі використовуються окремі провайдери для ключових доменних сутностей: AuthContext відповідає за аутентифікацію, CartContext – за стан кошика, WishlistContext – за список бажань користувача. Це дозволяє уникнути надмірного prop drilling та забезпечує глобальний доступ до критичних даних.

API-рівень системи поділений на окремі сервіси: Auth API, Products API, Cart API, Orders API та Admin API. Кожен із них відповідає за свою предметну область і забезпечує CRUD-операції та бізнес-логіку на стороні сервера. Взаємодія між модулями відбувається через HTTP-запити у форматі REST.

Отже, архітектура інтеграції забезпечує зв’язок між усіма рівнями системи: користувацький інтерфейс отримує дані через API, керування станом забезпечує їх узгодженість, а серверна частина гарантує цілісність і збереження даних.

Далі наведено узагальнену структуру взаємодії модулів системи, де показано зв’язок між клієнтською частиною, API-рівнем та базою даних. Центральним елементом виступає шар управління станом, який забезпечує синхронізацію даних між різними компонентами інтерфейсу. Після діаграми важливо відзначити, що всі взаємодії проходять через єдині точки доступу до API, що дозволяє уникнути дублювання логіки та спрощує масштабування системи в майбутньому. Потік даних у системі побудований за принципом подієво-орієнтованої взаємодії, де кожна дія користувача ініціює послідовність запитів до API та оновлення стану клієнтського інтерфейсу.

Наприклад, процес покупки (рис. 3.8) починається з перегляду каталогу товарів, де користувач обирає необхідний продукт і переходить на сторінку його детального опису. Після натискання кнопки «Купити» компонент ProductCard викликає функцію додавання товару до кошика через CartContext, яка, у свою чергу, відправляє POST-запит до Cart API.

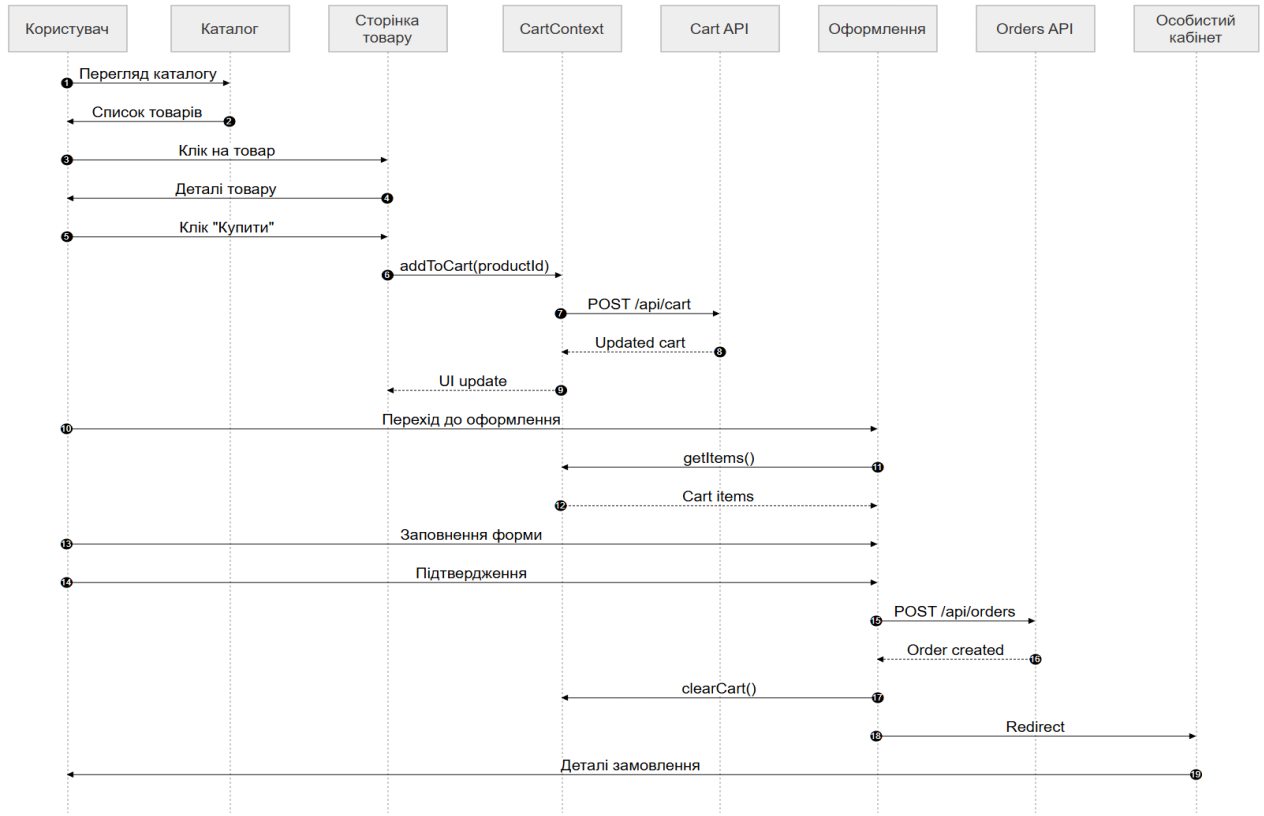


Рисунок 3.8 – Потік даних при покупці товару

Отримані дані одразу синхронізуються з глобальним станом кошика, що дозволяє оновити відображення кількості товарів у Header та інших компонентах інтерфейсу. Далі користувач переходить до сторінки оформлення замовлення, де система виконує запит до API для отримання актуального списку товарів у кошику.

Після заповнення форми оформлення замовлення відбувається відправка POST-запиту до Orders API, у результаті чого створюється нове замовлення. Після успішного створення замовлення кошик очищається, а користувач перенаправляється до сторінки деталей замовлення. Такий підхід забезпечує цілісність даних і виключає можливість розсинхронізації між клієнтським інтерфейсом та серверною частиною.

Адмін-панель є окремим модулем системи, який інтегрується з основною клієнтською частиною через спільні API-сервіси та механізми перевірки ролей користувача (рис. 3.9).

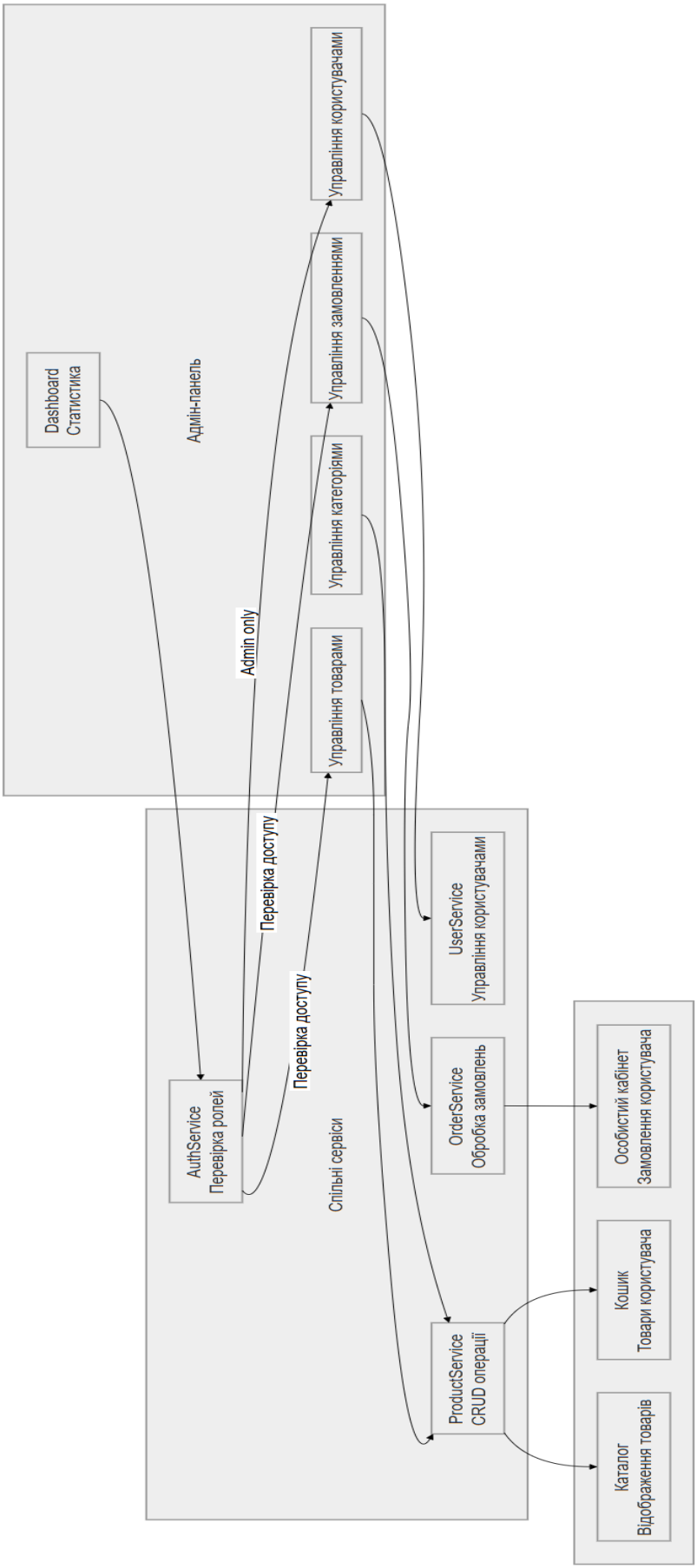


Рисунок 3.9 – Взаємодія адмін-панелі з основною системою

Доступ до адміністративних функцій обмежується на рівні AuthService, який перевіряє роль користувача перед наданням доступу до відповідних сторінок.

Адміністративний модуль взаємодіє з тими ж API-сервісами, що й клієнтська частина, однак має розширені права доступу. Зокрема, через ProductService здійснюється створення, редагування та видалення товарів, через OrderService – управління замовленнями, а через UserService – робота з обліковими записами користувачів.

Зміни, внесені в адмін-панелі, одразу відображаються у клієнтській частині системи завдяки механізмам повторної валідації запитів та оновлення кешу. Таким чином забезпечується узгодженість даних між різними модулями без необхідності ручного оновлення сторінок. Синхронізація даних між модулями реалізована через комбінацію контекстного управління станом та механізмів повторної валідації запитів. Це дозволяє підтримувати актуальність даних у різних частинах інтерфейсу без перевантаження серверної частини.

Основними подіями синхронізації є додавання товару до кошика, оновлення товарів в адмін-панелі, зміна статусу замовлення, авторизація користувача та оновлення списку бажань. Кожна з цих подій ініціює оновлення відповідного контексту або повторний запит до API. Приклад синхронізації з SWR наведено в додатку В.

У наведеному прикладі використовується механізм SWR, який забезпечує кешування запитів, автоматичну рефетчинг-синхронізацію та зменшення навантаження на сервер. Після внесення змін в адмін-панелі викликається функція mutate(), яка оновлює дані у відповідному кеші та забезпечує миттєве відображення змін у клієнтському інтерфейсі.

Інтерфейс системи побудований у вигляді інтуїтивно зрозумілої структури з поділом на основні модулі: каталог товарів, сторінку товару, кошик, оформлення замовлення та особистий кабінет користувача. Для адміністративної частини передбачено окрему панель управління товарами, замовленнями та

користувачами, що забезпечує зручне адміністрування системи. Загальний вигляд інтерфейсу наведено в додатку Г.

Отже, система забезпечує високий рівень узгодженості даних між модулями управління товарами та користувацьким кабінетом, що є критично важливим для коректної роботи e-commerce платформи.

3.4 Тестування функціональності вебсистеми

Тестування функціональності вебсистеми було виконано з метою перевірки коректності роботи основних модулів, стабільності бізнес-логіки та взаємодії клієнтської і серверної частин. У процесі тестування застосовувалися два основні підходи: модульне тестування (unit testing) та end-to-end тестування (E2E), що дозволило комплексно оцінити працездатність системи на різних рівнях.

Модульне тестування було реалізовано за допомогою Jest і спрямоване на перевірку ключових компонентів серверної частини, зокрема роботи бази даних, механізмів автентифікації та CRUD-операцій.

Результати тестування (рис. 3.10) показали повну коректність ініціалізації структури бази даних. Було підтверджено наявність та правильну конфігурацію основних таблиць: users, products, wishlist, cart_items та orders. Окремо перевірено процес заповнення seed-даними, включаючи категорії товарів, тестові продукти та користувачів із різними ролями (admin, manager, customer), що є критично важливим для подальшої роботи системи.

Особливу увагу приділено тестуванню механізмів автентифікації. Було перевірено:

- коректність хешування паролів за допомогою bcrypt;
- генерацію та валідацію JWT-токенів;
- обробку невалідних або підроблених токенів;
- передачу ролей користувачів у payload токена.

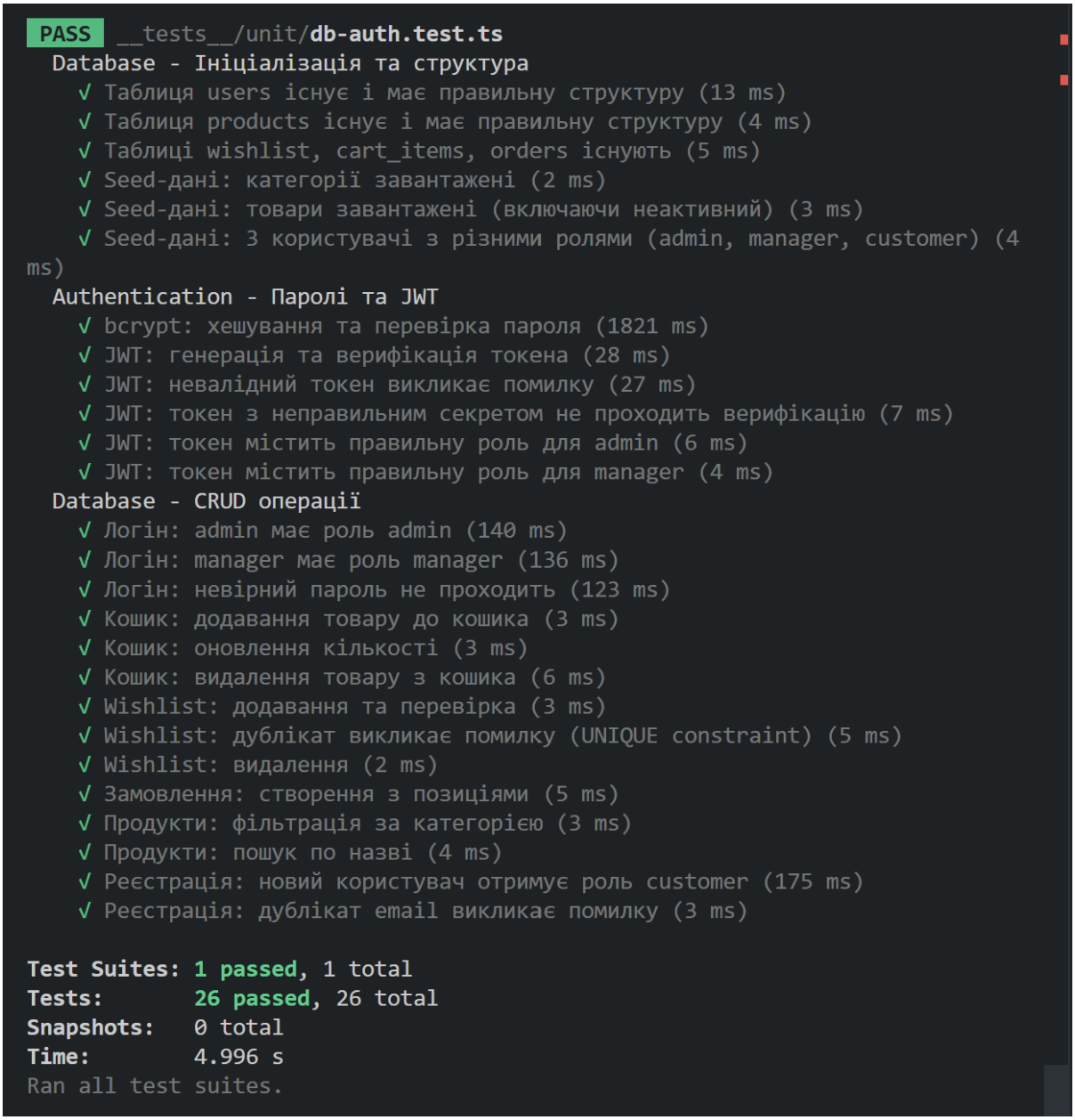


Рисунок 3.10 – Результати модульного тестування

Результати підтвердили стабільну роботу системи безпеки, включно з коректним розмежуванням доступу між ролями.

Окремий блок тестів було присвячено перевірці CRUD-операцій над основними сутностями системи.

Було перевірено:

– додавання, оновлення та видалення товарів у кошику;

- додавання та видалення товарів у wishlist з перевіркою унікальності записів;
- створення замовлень із коректною структурою позицій;
- фільтрацію товарів за категоріями;
- пошук товарів за назвою;
- процес реєстрації нових користувачів із автоматичним присвоєнням ролі customer.

Тестування показало відсутність критичних помилок у бізнес-логіці та коректну роботу обмежень бази даних, зокрема UNIQUE constraint для wishlist та перевірку валідності даних при створенні замовлень.

End-to-end тестування було реалізовано за допомогою Playwright і дозволило перевірити роботу системи з точки зору кінцевого користувача в умовах реальної взаємодії з інтерфейсом (рис. 3.11).

Було протестовано основні сценарії використання вебсистеми:

- відкриття головної сторінки та відображення ключових елементів інтерфейсу;
- коректне завантаження та відображення товарів на головній сторінці;
- робота сторінки каталогу та функції пошуку товарів;
- доступність і функціональність кошика;
- робота сторінки wishlist (вподобання);
- коректне відкриття статичних сторінок, зокрема сторінки допомоги та FAQ.

Результати E2E тестування підтвердили стабільність користувацьких сценаріїв та відсутність критичних помилок у навігації та взаємодії компонентів інтерфейсу.

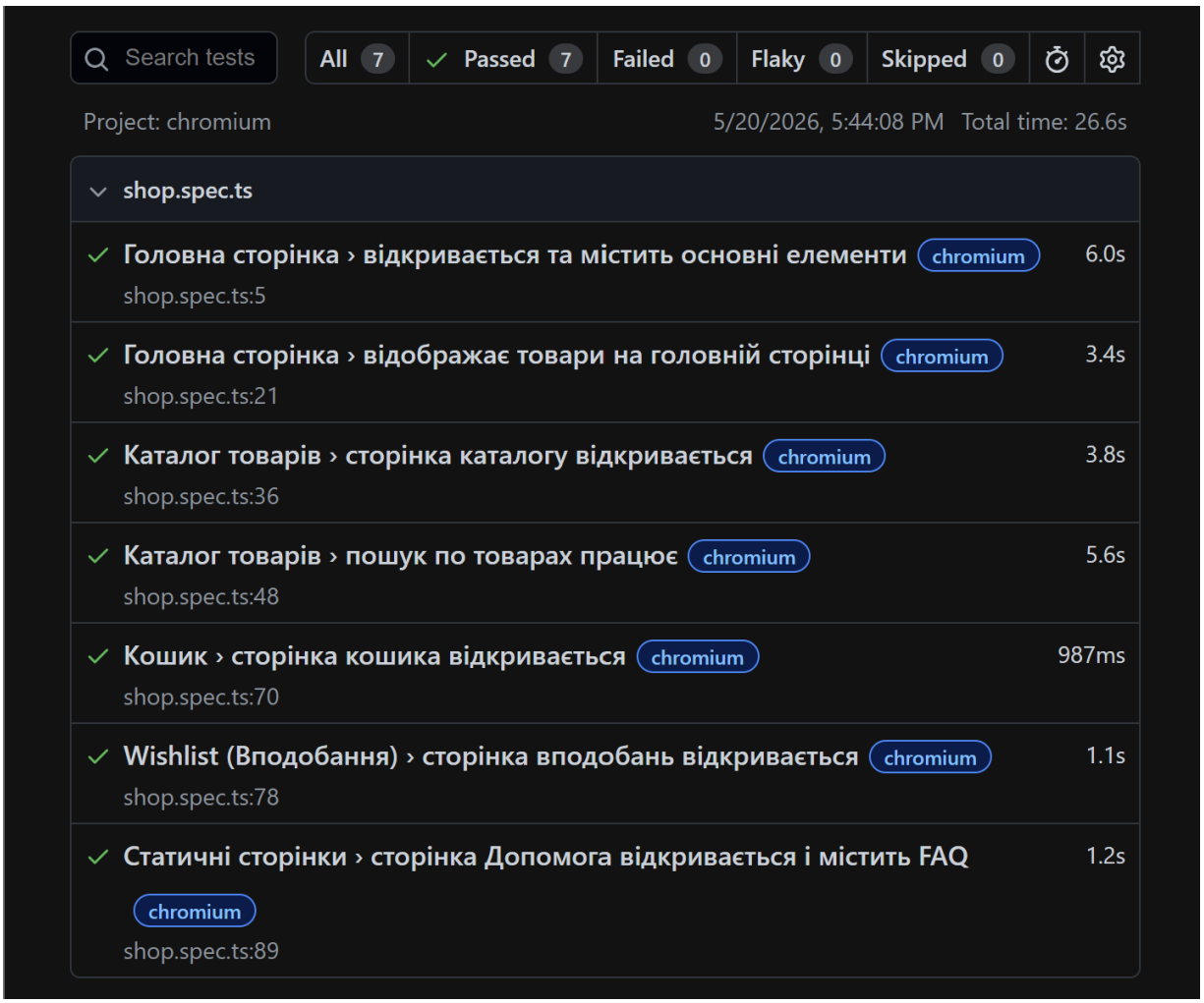


Рисунок 3.11 – Результати end-to-end тестування

Загалом було виконано 26 модульних тестів, які завершилися успішно без жодних помилок, а також 7 end-to-end тестів, що підтвердили коректність роботи основних користувацьких сценаріїв.

Час виконання модульних тестів склав приблизно 5 секунд, що свідчить про високу швидкість перевірки серверної логіки. End-to-end тести виконалися за 26.6 секунд, що є прийнятним показником для інтеграційного тестування браузерного рівня.

Отримані результати підтверджують стабільність роботи вебсистеми, правильність реалізації основних модулів та їх взаємодії, а також готовність системи до подальшого використання в умовах реальної експлуатації.

3.5 Інструкція користувача

Інструкція користувача вебсистеми інтернет-магазину «МаркетПлюс» розроблена з метою забезпечення зрозумілого та послідовного опису основних функцій системи для кінцевих користувачів, менеджерів та адміністраторів. Вона охоплює всі ключові сценарії взаємодії з платформою, включаючи реєстрацію, перегляд каталогу, оформлення замовлень та управління даними в адміністративній частині.

Вебсистема «МаркетПлюс» є сучасною платформою електронної комерції, яка забезпечує користувачам можливість перегляду товарів, їх пошуку та фільтрації, додавання до кошика та списку бажань, а також оформлення замовлень з подальшим відстеженням їх статусу.

Система також містить рольову модель доступу, яка розподіляє функціональні можливості між покупцем, менеджером та адміністратором.

Інструкція для покупця. Користувач-покупець має доступ до основних функцій інтернет-магазину. Для початку роботи необхідно пройти реєстрацію або авторизацію в системі. Після входу користувач отримує доступ до каталогу товарів, де може здійснювати перегляд, фільтрацію та сортування продукції за різними параметрами, такими як категорія, ціна та наявність. Для пошуку необхідного товару використовується рядок пошуку, який дозволяє швидко знаходити продукцію за назвою.

На сторінці товару користувач може ознайомитися з детальною інформацією, включаючи опис, ціну, наявність та рейтинг, а також виконати додавання товару до кошика або списку бажань. Кошик дозволяє змінювати кількість товарів, видаляти позиції та переходити до оформлення замовлення. Під час оформлення користувач вводить контактні дані, обирає спосіб доставки та оплати, після чого підтверджує замовлення. У особистому кабінеті доступні розділи для перегляду профілю, історії замовлень, списку бажань та налаштувань облікового запису.

Інструкція для менеджера. Менеджер має доступ до адміністративної панелі системи, що забезпечує управління товарами та обробку замовлень. Основні функції включають перегляд списку замовлень, зміну їх статусів (очікує, в обробці, відправлено, доставлено, скасовано), а також перегляд детальної інформації про кожне замовлення.

У розділі управління товарами менеджер може додавати нові товари, редагувати існуючі позиції та видаляти їх. Також доступна функція пошуку для швидкого знаходження необхідного товару. Додатково на панелі відображається аналітична інформація, зокрема кількість замовлень та загальні показники продажів.

Інструкція для адміністратора. Адміністратор має розширені права доступу, включаючи всі функції менеджера, а також додаткові можливості управління системою. Зокрема, адміністратор може керувати категоріями товарів, створювати нові категорії, редагувати або видаляти існуючі (за умови відсутності прив’язаних товарів).

Доступна функція управління користувачами, що включає перегляд списку користувачів, зміну їх ролей (customer, manager, admin), а також блокування або розблокування акаунтів.

Також, система передбачає розділ частих питань, який містить відповіді на типові запити користувачів щодо оформлення замовлень, способів оплати, умов доставки та повернення товарів. Визначено базові технічні вимоги для коректної роботи системи, зокрема використання сучасних браузерів, увімкнений JavaScript та стабільне інтернет-з’єднання. Вебсистема є повністю адаптивною та підтримує роботу на мобільних пристроях і планшетах.

Отже, інструкція користувача забезпечує повний опис функціональних можливостей вебсистеми «МаркетПлюс» та сприяє швидкому освоєнню інтерфейсу різними категоріями користувачів. Чіткий розподіл ролей і логічна структура навігації дозволяють ефективно використовувати всі можливості платформи без необхідності додаткового навчання.

3.6 Висновки

У третьому розділі здійснено практичну реалізацію інтернет-магазину та проведено його тестування. Розроблено серверну частину системи, що забезпечує обробку запитів, роботу з базою даних та реалізацію бізнес-логіки.

Створено клієнтський інтерфейс, який забезпечує зручну взаємодію користувача з системою. Виконано інтеграцію основних модулів, зокрема управління товарами та особистого кабінету користувача, що дозволило забезпечити цілісну роботу вебзастосунку.

Проведено тестування основних функціональних можливостей системи, яке підтвердило коректність її роботи. Також розроблено інструкцію користувача для подальшої експлуатації вебсистеми.

ВИСНОВКИ

У ході виконання роботи було досягнуто поставленої мети – розроблено інформаційну систему для онлайн-реалізації товарів з реалізацією особистого кабінету користувача та основними функціями інтернет-магазину. У процесі роботи послідовно виконано всі визначені завдання, що дозволило створити цілісний вебзастосунок із продуманою архітектурою та функціональністю.

На першому етапі проаналізовано сучасний стан розвитку інформаційних систем електронної комерції. Розглянуто особливості функціонування онлайн-магазинів, роль інформаційних систем у забезпеченні продажів, обробки замовлень та взаємодії з користувачами. Було встановлено, що сучасні e-commerce платформи базуються на інтеграції клієнтської частини, серверної логіки та персоналізованого користувацького кабінету, який є ключовим елементом взаємодії з клієнтом.

Досліджено функціональні можливості існуючих інтернет-магазинів та систем особистих кабінетів користувачів. У межах аналізу було розглянуто приклади великих платформ, зокрема маркетплейсів та сервісів цифрової дистрибуції. Це дозволило визначити основні функціональні модулі, які повинна містити сучасна система: каталог товарів, система пошуку і фільтрації, кошик, оформлення замовлень, історія покупок, а також адміністративна панель для управління контентом.

На основі проведеного аналізу визначено вимоги до розроблюваної інформаційної системи. Встановлено необхідність підтримки ролей користувачів (покупець, менеджер, адміністратор), реалізації CRUD-операцій для товарів, забезпечення роботи з кошиком і замовленнями, а також створення особистого кабінету користувача з можливістю перегляду та керування даними.

Наступним етапом стало проєктування архітектури інтернет-магазину та бази даних. Було розроблено багаторівневу архітектуру системи, що включає

клієнтський інтерфейс, API-рівень та серверну логіку. Спроєктовано структуру бази даних, яка забезпечує зберігання інформації про користувачів, товари, категорії, кошик, список бажань та замовлення. Така структура дозволяє ефективно організувати взаємодію між компонентами системи та забезпечує масштабованість.

У межах практичної реалізації розроблено серверну частину системи, що відповідає за обробку запитів, авторизацію користувачів та управління даними. Також створено клієнтський інтерфейс, який забезпечує зручну навігацію по каталогу товарів, перегляд сторінок товарів, роботу з кошиком і оформлення замовлень.

Особливу увагу приділено інтеграції модулів управління товарами та користувацького кабінету. Забезпечено взаємодію між контекстами стану, API-запитами та компонентами інтерфейсу, що дозволило реалізувати узгоджену роботу всієї системи в режимі реального часу.

На завершальному етапі проведено тестування функціональності вебсистеми. Виконано юніт-тести та end-to-end тестування, які підтвердили коректність роботи основних модулів: авторизації, управління товарами, кошика, списку бажань та оформлення замовлення. Усі тести були успішно пройдені, що свідчить про стабільність та працездатність розробленої системи.

Додатково створено інструкцію користувача, яка описує основні сценарії роботи з системою для покупців, менеджерів та адміністраторів, що забезпечує простоту впровадження та використання вебзастосунку.

Отже, усі поставлені завдання курсової роботи виконано в повному обсязі. Розроблена інформаційна система відповідає сучасним вимогам до вебзастосунків у сфері електронної комерції та може бути використана як основа для подальшого розширення функціональності, зокрема інтеграції платіжних систем, аналітики та розширених механізмів персоналізації користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Юдкова К. В. Особливості визначення поняття «інформаційна система». *Інформація і право*. 2015. № 2(14). С. 39–44.

2. Бреус С., Іваненко А. Інформаційні системи у торгівлі в умовах цифрової трансформації: виклики, ризики та потенціал зростання. *Економіка та суспільство*. 2025. № 75. URL : https://doi.org/10.32782/2524-0072/2025-75-46 (дата звернення : 12.05.2026).

3. Samoylenko H. T., Selivanova A. V. Distributed information systems in e-commerce. *Mathematical Machines and Systems*. 2023. № 2. P. 69–74. URL : https://doi.org/10.34121/1028-9763-2023-2-69-74 (дата звернення : 12.05.2026).

4. Arnell A. The role of information systems in the digital marketplace. Meer. 2025. URL : https://www.meer.com/en/90170-the-role-of-information-systems-in-the-digital-marketplace (дата звернення : 12.05.2026).

5. Ancuta P. I. Benefits and drawbacks of online trading versus traditional trading. *Educational factors in online trading.* Vol. 25, no. 1. P. 1253–1259. URL : https://doaj.org/article/de271c27205e4566a58b16ca9083ead1 (дата звернення : 12.05.2026).

6. Csiki D. Benefits of cloud-based trade order management systems. INDATA. 2025. URL : https://www.indataipm.com/benefits-of-cloud-based-trade-order-management-systems/ (дата звернення : 12.05.2026).

7. World Trade Organization, Organization for Economic Co-operation and Development, International Monetary Fund, United Nations. Introduction. IMF eLibrary. URL : https://doi.org/10.5089/9789287073600.071.CH001 (дата звернення : 12.05.2026).

8. Шагоферов С. Що таке особистий кабінет на сайті? LIGHT – Сайт для яскравих людей. URL : https://lt.ogo.ua/light/shho-take-osobystyj-kabinet-na-sajti/ (дата звернення : 12.05.2026).

9. Ткаченко А. Особистий кабінет в інтернет-магазині: функції та вплив на продажі. Wezom. 2025. URL : https://wezom.com.ua/ua/blog/lichnyj-kabinet-kak-marketingovyj-instrument-dlya-povysheniya-prodazh (дата звернення : 12.05.2026).

10. Інтернет-магазин ROZETKA™ : офіційний сайт онлайн-гіпермаркету Розетка в Україні. ROZETKA. URL : https://rozetka.com.ua/ (дата звернення : 12.05.2026).

11. Microsoft Store | Xbox. Xbox.com. URL : https://www.xbox.com/en-US/microsoft-store (дата звернення : 12.05.2026).

12. Necula S. Exploring the Model-View-Controller (MVC) architecture: A broad analysis of market and technological applications. Preprints.org. 2024. DOI :https://doi.org/10.20944/preprints202404.1860.v1.

13. Next.js docs. Next.js. URL : https://nextjs.org/docs (дата звернення : 12.05.2026).

14. React. URL : https://react.dev/ (дата звернення : 12.05.2026).

15. SQLite Documentation. URL : https://sqlite.org/docs.html (дата звернення : 12.05.2026).

16. Documentation – TypeScript for JavaScript programmers. URL : https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html (дата звернення : 12.05.2026).

17. Documentation – Tailwind CSS. URL : https://v2.tailwindcss.com/docs (дата звернення : 12.05.2026).

18. Getting started · Jest. URL : https://archive.jestjs.io/docs/en/22.x/getting-started.html (дата звернення : 12.05.2026).

19. Playwright documentation – DevDocs. URL : https://devdocs.io/playwright/ (дата звернення : 12.05.2026).

20. Documentation for Visual Studio Code. URL : https://code.visualstudio.com/docs (дата звернення : 12.05.2026).

ДОДАТОК А

Реалізація серверної логіки та API-модуля e-commerce застосунку

Лістинг А.1 Реалізація функцій аутентифікації (lib/auth.ts)

```
import jwt from 'jsonwebtoken'
import bcrypt from 'bcryptjs'
import { cookies } from 'next/headers'
import { getDb, User, UserRole } from './db'

const JWT_SECRET = process.env.JWT_SECRET || 'your-secret-key'

export interface AuthUser {
  id: number
  email: string
  name: string
  role: UserRole
}

// Генерація JWT токена
export function generateToken(user: AuthUser): string {
  return jwt.sign(
    { id: user.id, email: user.email, role: user.role },
    JWT_SECRET,
    { expiresIn: '7d' }
  )
}
```

```
// Верифікація токена
export function verifyToken(token: string): AuthUser | null {
  try {
    return jwt.verify(token, JWT_SECRET) as AuthUser
  } catch {
    return null
  }
}

// Хешування пароля
export async function hashPassword(password: string): Promise<string> {
  return bcrypt.hash(password, 10)
}

// Перевірка пароля
export async function verifyPassword(
  password: string,
  hash: string
): Promise<boolean> {
  return bcrypt.compare(password, hash)
}

// Отримання поточного користувача з cookies
export async function getCurrentUser(): Promise<AuthUser | null> {
  const cookieStore = await cookies()
  const token = cookieStore.get('token')?.value

  if (!token) return null
```

```
  return verifyToken(token)
}

// Перевірка ролі користувача
export function hasRole(user: AuthUser | null, roles: UserRole[]): boolean {
  if (!user) return false
  return roles.includes(user.role)
}
```

Лістинг А.2 API endpoint для товарів (app/api/products/route.ts)

```
import { NextRequest, NextResponse } from 'next/server'
import { getDb } from '@/lib/db'
import { getCurrentUser, hasRole } from '@/lib/auth'

// GET - Отримання списку товарів з фільтрацією
export async function GET(request: NextRequest) {
  const { searchParams } = new URL(request.url)
  const category = searchParams.get('category')
  const search = searchParams.get('search')
  const sort = searchParams.get('sort') || 'newest'
  const page = parseInt(searchParams.get('page') || '1')
  const limit = parseInt(searchParams.get('limit') || '12')

  const db = getDb()

  let query = `
    SELECT p.*, c.name as category_name
    FROM products p
```

```
  LEFT JOIN categories c ON p.category_id = c.id
  WHERE 1=1
`
const params: any[] = []

// Фільтрація за категорією
if (category) {
  query += ` AND c.slug = ?`
  params.push(category)
}

// Пошук за назвою
if (search) {
  query += ` AND p.name LIKE ?`
  params.push(`%${search}%`)
}

// Сортування
const sortOptions: Record<string, string> = {
  newest: 'p.created_at DESC',
  price_asc: 'p.price ASC',
  price_desc: 'p.price DESC',
  name: 'p.name ASC',
  rating: 'p.rating DESC'
}
query += ` ORDER BY ${sortOptions[sort] || sortOptions.newest}`

// Пагінація
const offset = (page - 1) * limit
query += ` LIMIT ? OFFSET ?`
```

```
  params.push(limit, offset)

  const products = db.prepare(query).all(...params)

  // Підрахунок загальної кількості
  let countQuery = `
    SELECT COUNT(*) as total FROM products p
    LEFT JOIN categories c ON p.category_id = c.id
    WHERE 1=1
  `
  if (category) countQuery += ` AND c.slug = ?`
  if (search) countQuery += ` AND p.name LIKE ?`

  const countParams = params.slice(0, -2)
  const { total } = db.prepare(countQuery).get(...countParams) as any

  return NextResponse.json({
    products,
    pagination: {
      page,
      limit,
      total,
      pages: Math.ceil(total / limit)
    }
  })
}

// POST - Створення нового товару (тільки manager/admin)
export async function POST(request: NextRequest) {
  const user = await getCurrentUser()
```

```
if (!hasRole(user, [‘manager’, ‘admin’])) {
  return NextResponse.json(
    { error: ‘Доступ заборонено’ },
    { status: 403 }
  )
}

const data = await request.json()
const db = getDb()

// Генерація slug
const slug = data.name
  .toLowerCase()
  .replace(/[^a-za-яіїєґ0-9]+/gi, ‘-’)
  .replace(/-+/g, ‘-’)
  .replace(/^-|-$/g, ‘‘)

const result = db.prepare(`
  INSERT INTO products (
    name, slug, description, price, old_price,
    image, stock, category_id, is_featured
  ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
`).run(
  data.name, slug, data.description, data.price,
  data.old_price, data.image, data.stock,
  data.category_id, data.is_featured ? 1 : 0
)

const product = db.prepare(
```

```
    'SELECT * FROM products WHERE id = ?'
  ).get(result.lastInsertRowid)

  return NextResponse.json({ success: true, product })
}
```

Лістинг А.3 Реалізація API кошика (app/api/cart/route.ts)

```
import { NextRequest, NextResponse } from 'next/server'
import { getDb } from '@/lib/db'
import { getCurrentUser } from '@/lib/auth'

// GET - Отримання кошика користувача
export async function GET() {
  const user = await getCurrentUser()

  if (!user) {
    return NextResponse.json({ items: [], total: 0 })
  }

  const db = getDb()
  const items = db.prepare(`
    SELECT
      c.id, c.quantity, c.product_id,
      p.name, p.slug, p.price, p.old_price,
      p.image, p.stock
    FROM cart c
    JOIN products p ON c.product_id = p.id
    WHERE c.user_id = ?
```

```
    ORDER BY c.created_at DESC
  `).all(user.id)

  const total = items.reduce(
    (sum: number, item: any) => sum + item.price * item.quantity,
    0
  )

  return NextResponse.json({ items, total })
}

// POST - Додавання товару до кошика
export async function POST(request: NextRequest) {
  const user = await getCurrentUser()

  if (!user) {
    return NextResponse.json(
      { error: 'Необхідна авторизація' },
      { status: 401 }
    )
  }

  const { productId, quantity = 1 } = await request.json()
  const db = getDb()

  // Перевірка наявності товару
  const product = db.prepare(
    'SELECT * FROM products WHERE id = ?'
  ).get(productId)
```

```
  if (!product) {
    return NextResponse.json(
      { error: 'Товар не знайдено' },
      { status: 404 }
    )
  }

  // Перевірка чи товар вже в кошику
  const existingItem = db.prepare(`
    SELECT * FROM cart
    WHERE user_id = ? AND product_id = ?
  `).get(user.id, productId)

  if (existingItem) {
    db.prepare(`
      UPDATE cart SET quantity = quantity + ?
      WHERE user_id = ? AND product_id = ?
    `).run(quantity, user.id, productId)
  } else {
    db.prepare(`
      INSERT INTO cart (user_id, product_id, quantity)
      VALUES (?, ?, ?)
    `).run(user.id, productId, quantity)
  }

  return NextResponse.json({ success: true })
}

// DELETE - Видалення товару з кошика
export async function DELETE(request: NextRequest) {
```

```
  const user = await getCurrentUser()

  if (!user) {
    return NextResponse.json(
      { error: 'Необхідна авторизація' },
      { status: 401 }
    )
  }

  const { productId } = await request.json()
  const db = getDb()

  db.prepare(`
    DELETE FROM cart
    WHERE user_id = ? AND product_id = ?
  `).run(user.id, productId)

  return NextResponse.json({ success: true })
}
```

Лістинг А.4 Реалізація створення замовлення

```
// POST - Створення замовлення
export async function POST(request: NextRequest) {
  const user = await getCurrentUser()

  if (!user) {
    return NextResponse.json(
      { error: 'Необхідна авторизація' },
```

```
    { status: 401 }
  )
}

const { name, phone, address, delivery, payment } = await request.json()
const db = getDb()

// Отримання товарів з кошика
const cartItems = db.prepare(`
  SELECT c.*, p.price, p.stock
  FROM cart c
  JOIN products p ON c.product_id = p.id
  WHERE c.user_id = ?
`).all(user.id) as any[]

if (cartItems.length === 0) {
  return NextResponse.json(
    { error: 'Кошик порожній' },
    { status: 400 }
  )
}

// Перевірка наявності товарів
for (const item of cartItems) {
  if (item.stock < item.quantity) {
    return NextResponse.json(
      { error: `Недостатньо товару: ${item.name}` },
      { status: 400 }
    )
  }
```

```
}

// Розрахунок загальної суми
const total = cartItems.reduce(
  (sum, item) => sum + item.price * item.quantity,
  0
)

// Транзакція створення замовлення
const createOrder = db.transaction(() => {
  // Створення замовлення
  const orderResult = db.prepare(`
    INSERT INTO orders (
      user_id, status, total, name, phone,
      address, delivery, payment
    ) VALUES (?, 'pending', ?, ?, ?, ?, ?, ?)
  `).run(user.id, total, name, phone, address, delivery, payment)

  const orderId = orderResult.lastInsertRowid

  // Додавання позицій замовлення
  const insertItem = db.prepare(`
    INSERT INTO order_items (order_id, product_id, quantity, price)
    VALUES (?, ?, ?, ?)
  `)

  const updateStock = db.prepare(`
    UPDATE products SET stock = stock - ? WHERE id = ?
  `)
```

```
    for (const item of cartItems) {
      insertItem.run(orderId, item.product_id, item.quantity, item.price)
      updateStock.run(item.quantity, item.product_id)
    }

    // Очищення кошика
    db.prepare('DELETE FROM cart WHERE user_id = ?').run(user.id)

    return orderId
  })

  const orderId = createOrder()

  return NextResponse.json({
    success: true,
    orderId,
    message: 'Замовлення успішно створено'
  })
}
```

ДОДАТОК Б

Реалізація клієнтської частини та React-компонентів інтерфейсу

Лістинг Б.1 AuthProvider (components/auth-provider.tsx)

```
‘use client’

import { createContext, useContext, useState, useEffect, ReactNode } from ‘react’

type UserRole = ‘customer’ | ‘manager’ | ‘admin’

interface User {
  id: number
  email: string
  name: string
  role: UserRole
}

interface AuthContextType {
  user: User | null
  loading: boolean
  login: (email: string, password: string) => Promise<boolean>
  register: (name: string, email: string, password: string) => Promise<boolean>
  logout: () => Promise<void>
}

const AuthContext = createContext<AuthContextType | undefined>(undefined)
export function AuthProvider({ children }: { children: ReactNode }) {
```

```
const [user, setUser] = useState<User | null>(null)
const [loading, setLoading] = useState(true)

// Перевірка аутентифікації при завантаженні
useEffect(() => {
  checkAuth()
}, [])

const checkAuth = async () => {
  try {
    const res = await fetch('/api/auth')
    if (res.ok) {
      const data = await res.json()
      setUser(data.user)
    }
  } catch (error) {
    console.error('Auth check failed:', error)
  } finally {
    setLoading(false)
  }
}

const login = async (email: string, password: string): Promise<boolean> => {
  try {
    const res = await fetch('/api/auth', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ action: 'login', email, password })
    })
    if (res.ok) {
      const data = await res.json()
```

```
      setUser(data.user)
      return true
    }
    return false
  } catch {
    return false
  }
}

const register = async (
  name: string,
  email: string,
  password: string
): Promise<boolean> => {
  try {
    const res = await fetch('/api/auth', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ action: 'register', name, email, password })
    })

    if (res.ok) {
      const data = await res.json()
      setUser(data.user)
      return true
    }
    return false
  } catch {
    return false
  }
}
```

```
  const logout = async () => {
    await fetch('/api/auth', { method: 'DELETE' })
    setUser(null)
  }

  return (
    <AuthContext.Provider value={{ user, loading, login, register, logout }}>
      {children}
    </AuthContext.Provider>
  )
}

export function useAuth() {
  const context = useContext(AuthContext)
  if (!context) {
    throw new Error('useAuth must be used within AuthProvider')
  }
  return context
}
```

Лістинг Б.2 CartProvider (components/cart-provider.tsx)

```
'use client'

import { createContext, useContext, useState, useEffect, ReactNode } from 'react'
import { useAuth } from './auth-provider'

interface CartItem {
```

```
  id: number
  product_id: number
  name: string
  slug: string
  price: number
  old_price: number | null
  image: string
  quantity: number
  stock: number
}

interface CartContextType {
  items: CartItem[]
  total: number
  itemCount: number
  loading: boolean
  addToCart: (productId: number, quantity?: number) => Promise<void>
  removeFromCart: (productId: number) => Promise<void>
  updateQuantity: (productId: number, quantity: number) => Promise<void>
  clearCart: () => Promise<void>
}

const CartContext = createContext<CartContextType | undefined>(undefined)

export function CartProvider({ children }: { children: ReactNode }) {
  const { user } = useAuth()
  const [items, setItems] = useState<CartItem[]>([])
  const [loading, setLoading] = useState(false)

  // Завантаження кошика при зміні користувача
  useEffect(() => {
```

```
  if (user) {
    fetchCart()
  } else {
    setItems([])
  }
}, [user])

const fetchCart = async () => {
  setLoading(true)
  try {
    const res = await fetch('/api/cart')
    const data = await res.json()
    setItems(data.items || [])
  } catch (error) {
    console.error('Failed to fetch cart:', error)
  } finally {
    setLoading(false)
  }
}

const addToCart = async (productId: number, quantity = 1) => {
  const res = await fetch('/api/cart', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ productId, quantity })
  })

  if (res.ok) {
    await fetchCart()
  }
}
```

```
const removeFromCart = async (productId: number) => {
  const res = await fetch('/api/cart', {
    method: 'DELETE',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ productId })
  })

  if (res.ok) {
    await fetchCart()
  }
}

const updateQuantity = async (productId: number, quantity: number) => {
  if (quantity <= 0) {
    await removeFromCart(productId)
    return
  }

  const res = await fetch('/api/cart', {
    method: 'PUT',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ productId, quantity })
  })

  if (res.ok) {
    await fetchCart()
  }
}

const clearCart = async () => {
```

```
    const res = await fetch('/api/cart', { method: 'DELETE' })
    if (res.ok) {
      setItems([])
    }
  }

  const total = items.reduce(
    (sum, item) => sum + item.price * item.quantity,
    0
  )

  const itemCount = items.reduce(
    (sum, item) => sum + item.quantity,
    0
  )

  return (
    <CartContext.Provider value={{
      items, total, itemCount, loading,
      addToCart, removeFromCart, updateQuantity, clearCart
    }}>
      {children}
    </CartContext.Provider>
  )
}

export function useCart() {
  const context = useContext(CartContext)
  if (!context) {
    throw new Error('useCart must be used within CartProvider')
  }
```

```
  return context
}
```

Лістинг Б.3 ProductCard (components/ProductCard.tsx)

```
‘use client’

import { useState } from ‘react’
import Link from ‘next/link’
import Image from ‘next/image’
import { Heart, ShoppingCart, Check } from ‘lucide-react’
import { Button } from ‘@/components/ui/button’
import { Card, CardContent } from ‘@/components/ui/card’
import { Badge } from ‘@/components/ui/badge’
import { useAuth } from ‘./auth-provider’
import { useCart } from ‘./cart-provider’
import { useWishlist } from ‘./wishlist-provider’
import { cn } from ‘@/lib/utils’

interface Product {
  id: number
  name: string
  slug: string
  price: number
  old_price: number | null
  image: string
  rating: number
  reviews_count: number
  stock: number
}
```

```
interface ProductCardProps {
  product: Product
  className?: string
}

export function ProductCard({ product, className }: ProductCardProps) {
  const { user } = useAuth()
  const { addToCart, items } = useCart()
  const { isInWishlist, addToWishlist, removeFromWishlist } = useWishlist()
  const [isAdding, setIsAdding] = useState(false)
  const [justAdded, setJustAdded] = useState(false)

  const discount = product.old_price
    ? Math.round(((product.old_price - product.price) / product.old_price) * 100)
    : 0

  const isInCart = items.some(item => item.product_id === product.id)
  const inWishlist = isInWishlist(product.id)

  const handleAddToCart = async (e: React.MouseEvent) => {
    e.preventDefault()
    e.stopPropagation()

    if (!user) {
      window.location.href = '/login'
      return
    }

    setIsAdding(true)
    await addToCart(product.id)
```

```
  setIsAdding(false)
  setJustAdded(true)
  setTimeout(() => setJustAdded(false), 2000)
 }

 const handleToggleWishlist = async (e: React.MouseEvent) => {
  e.preventDefault()
  e.stopPropagation()

  if (!user) {
   window.location.href = '/login'
   return
  }

  if (inWishlist) {
   await removeFromWishlist(product.id)
  } else {
   await addToWishlist(product.id)
  }
 }

 return (
  <Card className={cn('group overflow-hidden', className)}>
   <Link href={`/product/${product.slug}`}>
    <div className="relative aspect-square bg-secondary/30">
     {/* Зображення товару */}
     <Image
      src={product.image || '/placeholder.png'}
      alt={product.name}
      fill
      className="object-cover transition-transform group-hover:scale-105"
```

```
  />

  {/* Знижка */}
  {discount > 0 && (
    <Badge className=«absolute top-2 left-2 bg-destructive»>
      -{discount}%
    </Badge>
  )}

  {/* Кнопка wishlist */}
  <Button
    variant=«secondary»
    size=«icon»
    className={cn(
      ‘absolute top-2 right-2 opacity-0 group-hover:opacity-100’,
      inWishlist && ‘opacity-100 bg-destructive/10 text-destructive’
    )}
    onClick={handleToggleWishlist}
  >
    <Heart className={cn(‘h-4 w-4’, inWishlist && ‘fill-current’)} />
  </Button>
</div>

<CardContent className=«p-4»>
  {/* Назва товару */}
  <h3 className=«font-medium line-clamp-2 mb-2 min-h-[2.5rem]»>
    {product.name}
  </h3>

  {/* Рейтинг */}
  <div className=«flex items-center gap-1 mb-2»>
```

```
  {[...Array(5)].map((_, i) => (
   <span
    key={i}
    className={cn(
     'text-sm',
     i < Math.round(product.rating)
      ? 'text-yellow-500'
      : 'text-muted'
    )}
   >
    ★
   </span>
  ))}
  <span className="text-xs text-muted-foreground ml-1">
   ({product.reviews_count})
  </span>
 </div>

 {/* Ціна */}
 <div className="flex items-center gap-2 mb-3">
  <span className="text-lg font-bold text-primary">
   {product.price.toLocaleString()} ₴
  </span>
  {product.old_price && (
   <span className="text-sm text-muted-foreground line-through">
    {product.old_price.toLocaleString()} ₴
   </span>
  )}
 </div>
 {/* Кнопка додавання в кошик */}
```

```
        <Button
          className=«w-full»
          variant={isInCart || justAdded ? ‘secondary’ : ‘default’}
          disabled={isAdding || product.stock === 0}
          onClick={handleAddToCart}
        >
          {product.stock === 0 ? (
            ‘Немає в наявності’
          ) : isInCart || justAdded ? (
            <>
              <Check className=«mr-2 h-4 w-4» />
              В кошику
            </>
          ) : (
            <>
              <ShoppingCart className=«mr-2 h-4 w-4» />
              Купити
            </>
          )}
        </Button>
      </CardContent>
    </Link>
  </Card>
)}
```

ДОДАТОК В

Приклад синхронізації з SWR

```
import useSWR from ‘swr’

const fetcher = (url: string) => fetch(url).then(res => res.json())

export function useProducts(filters: ProductFilters) {
  const queryString = new URLSearchParams(filters as any).toString()

  const { data, error, isLoading, mutate } = useSWR(
    `/api/products?${queryString}`,
    fetcher,
    {
      revalidateOnFocus: false,
      revalidateOnReconnect: true,
      dedupingInterval: 5000
    }
  )

  return {
    products: data?.products || [],
    pagination: data?.pagination,
    isLoading,
    isError: error,
    refresh: mutate
  }
```

```
}

// Використання в компоненті
function CatalogPage() {
  const { products, pagination, isLoading, refresh } = useProducts({
    category: 'electronics',
    sort: 'price_asc',
    page: 1
  })

  // Після оновлення товару в адмін-панелі
  const handleProductUpdate = async () => {
    await updateProduct(productData)
    refresh() // Оновлення списку
  }

  // ...
}
```

ДОДАТОК Г

Вигляд розробленої інформаційної системи

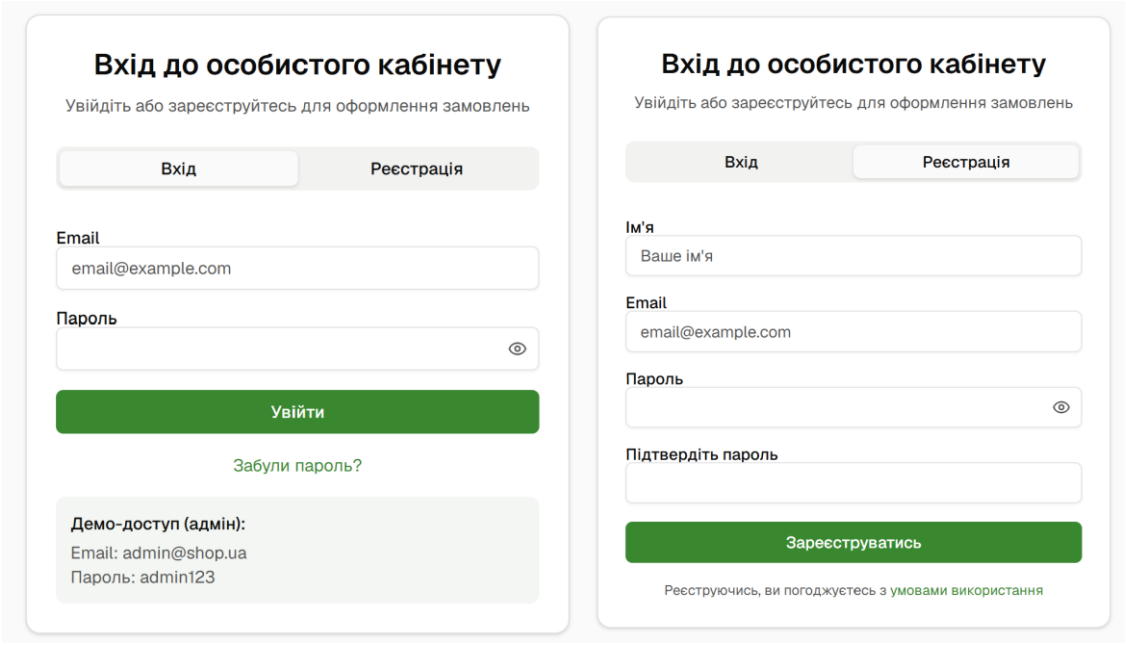


Рисунок Г.1 – Сторінка входу та реєстрації

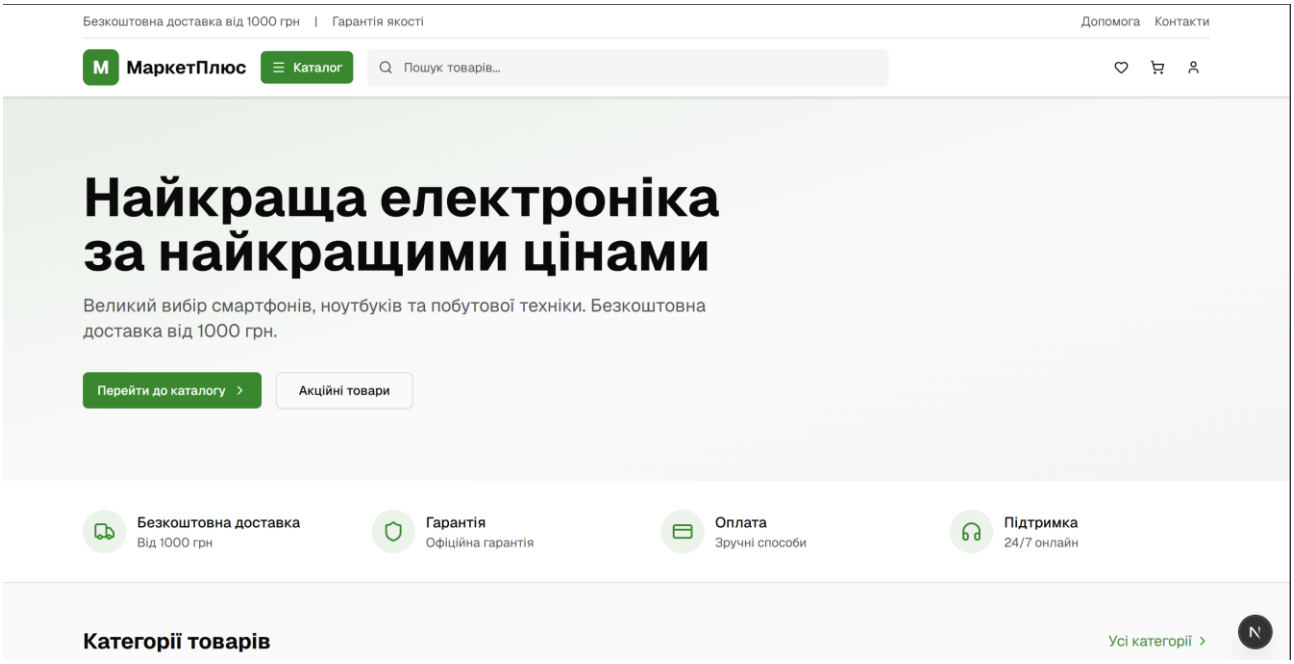
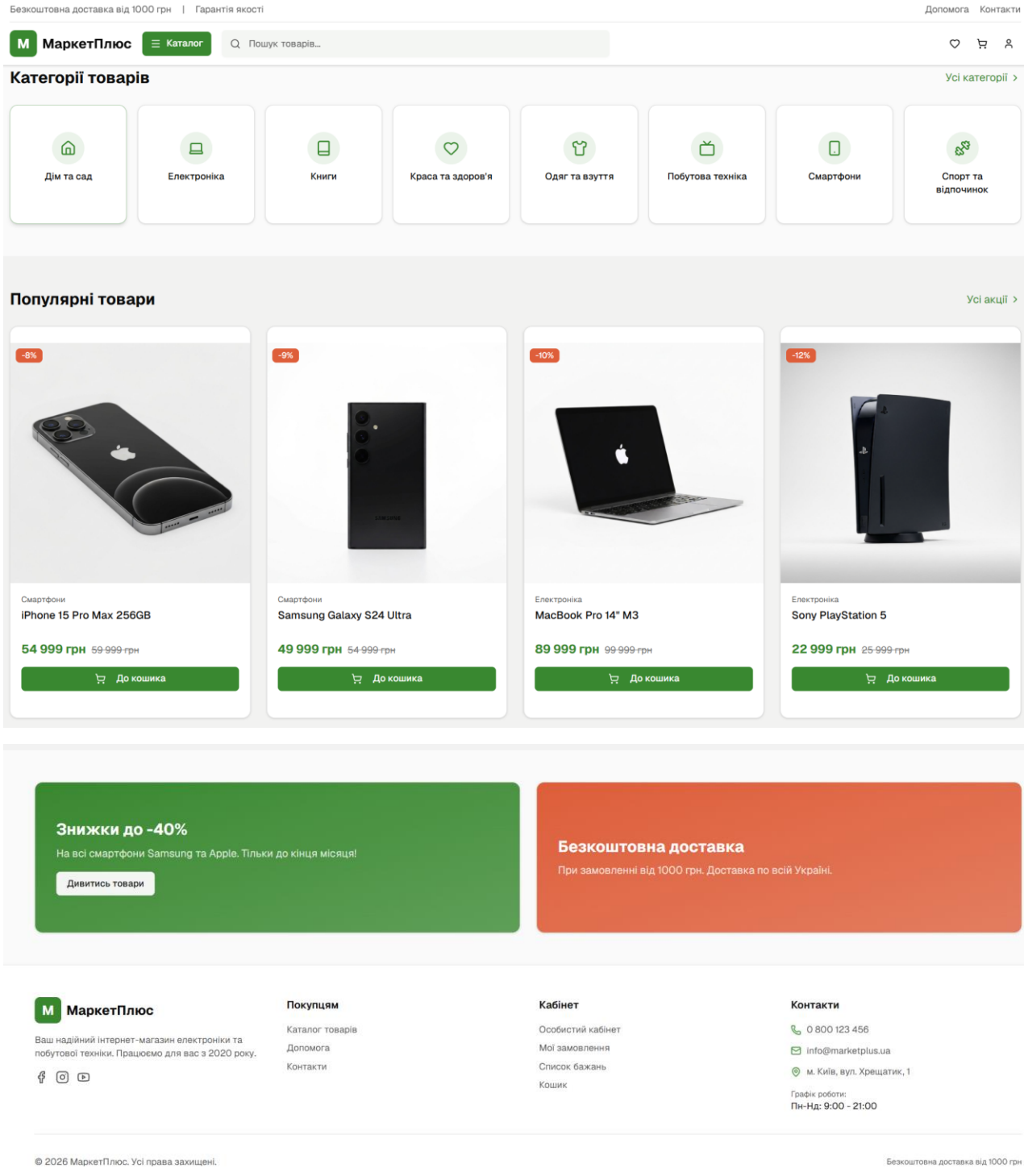
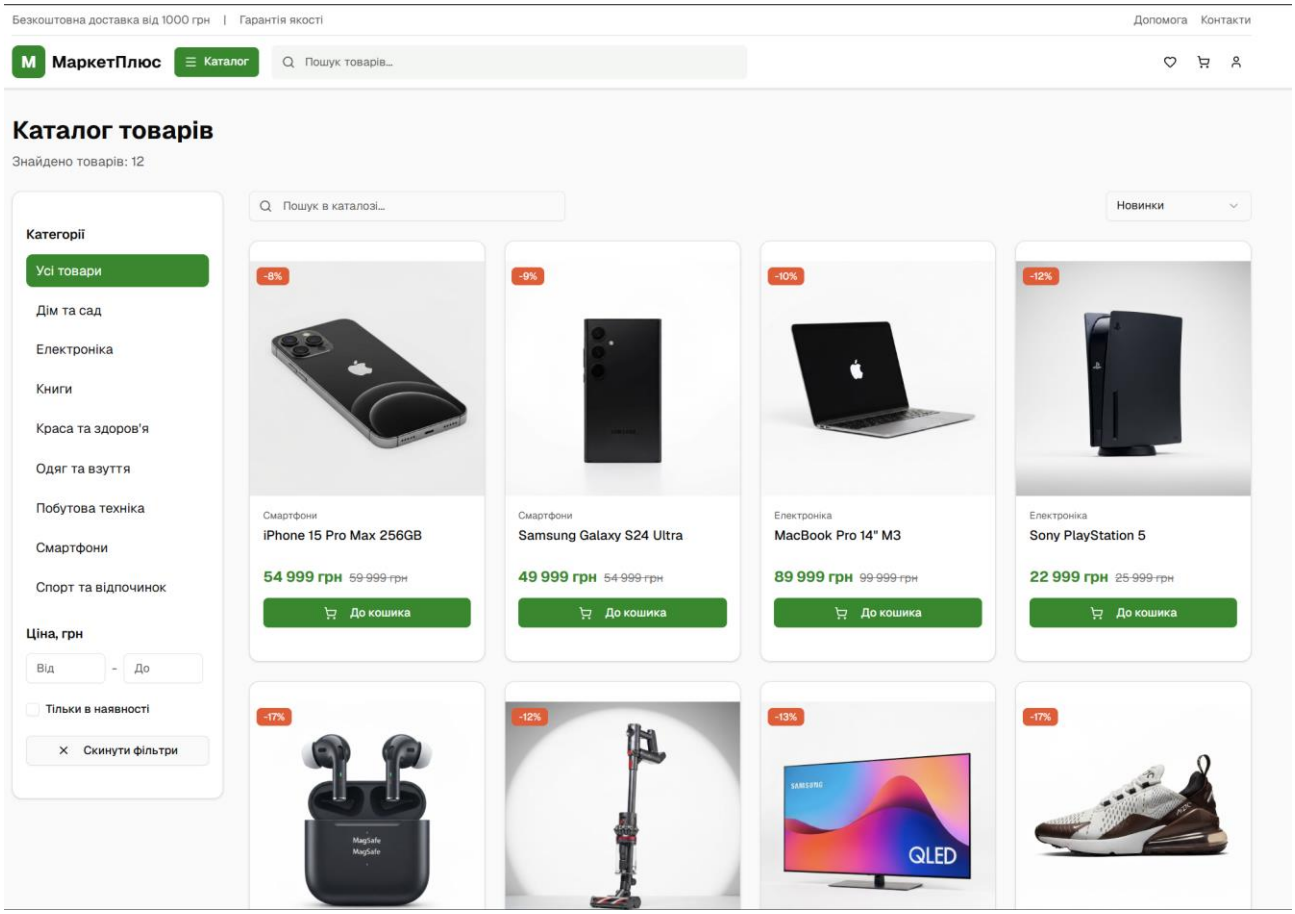


Рисунок Г.2 – Головна сторінка

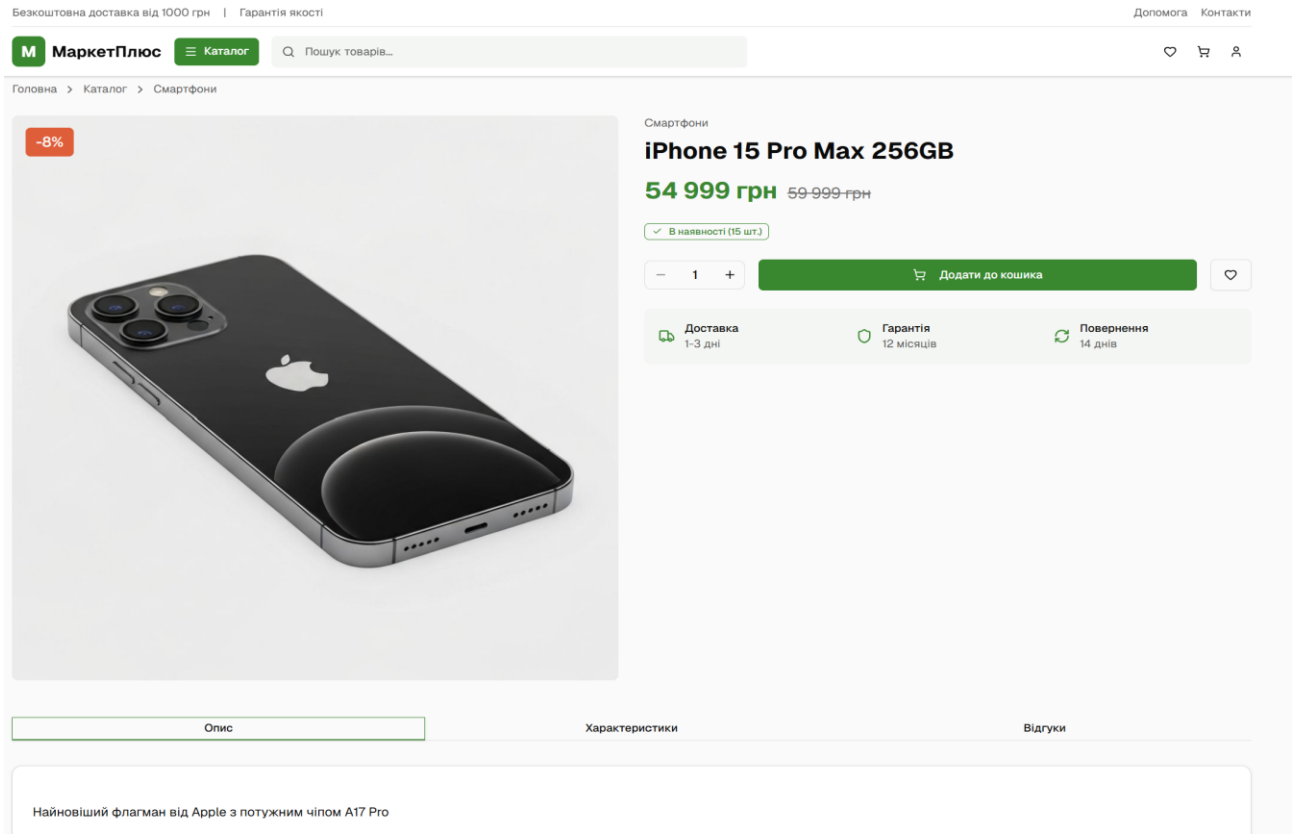
Г.3 – Продовження головної сторінки



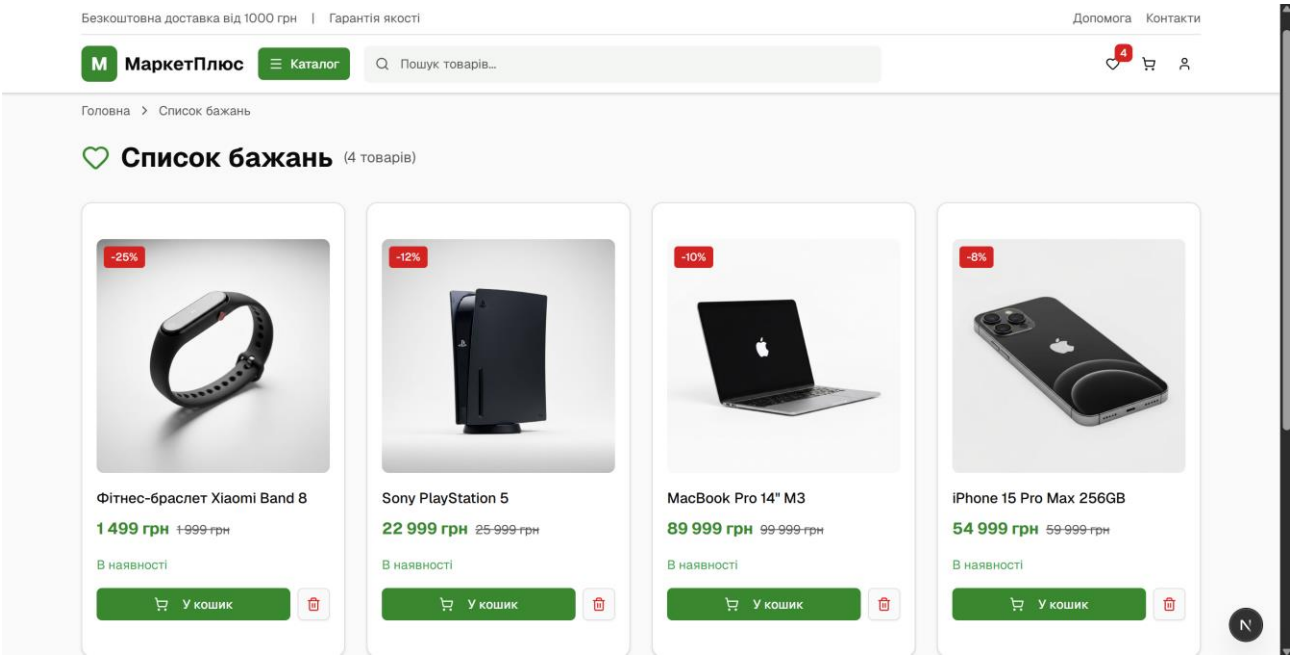
Г.4 – Сторінка каталогу



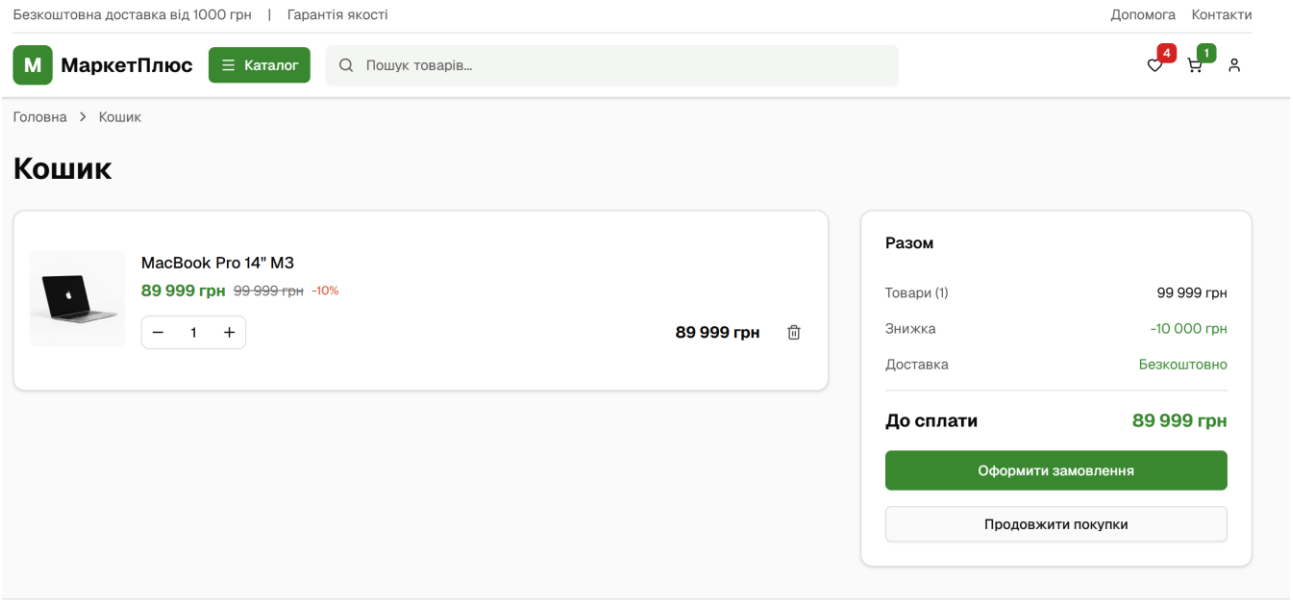
Г.5 – Сторінка опису товару



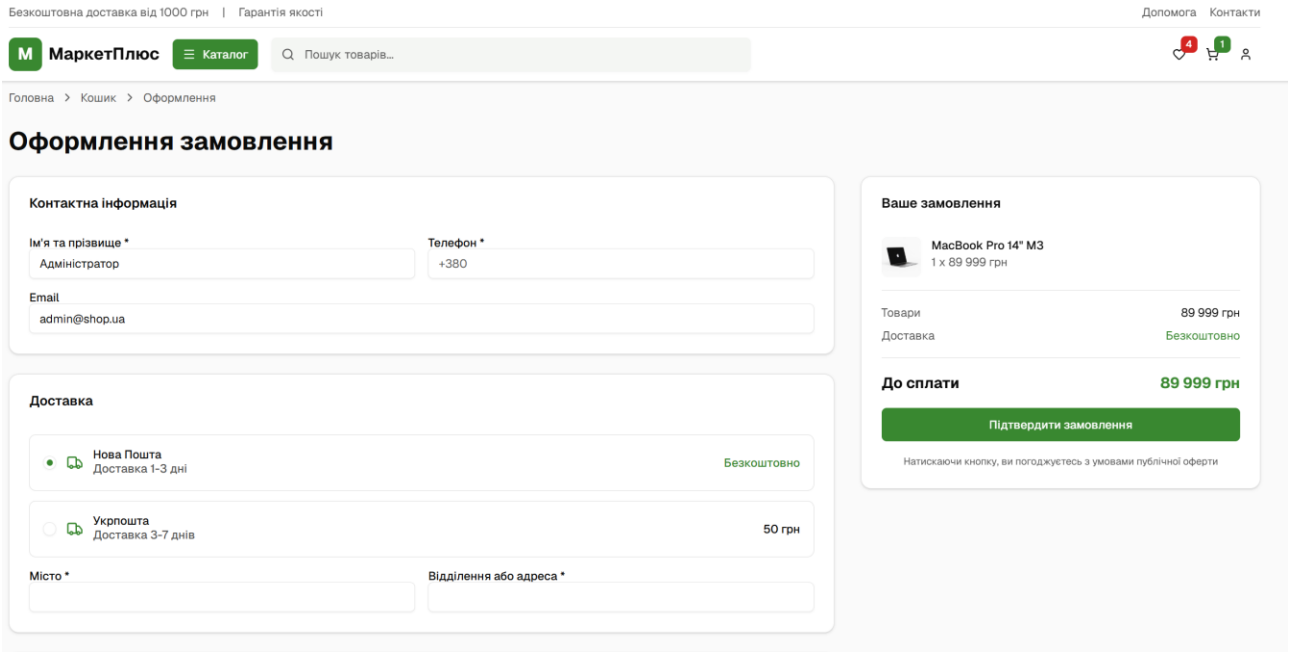
Г.6 – Сторінка списку бажань



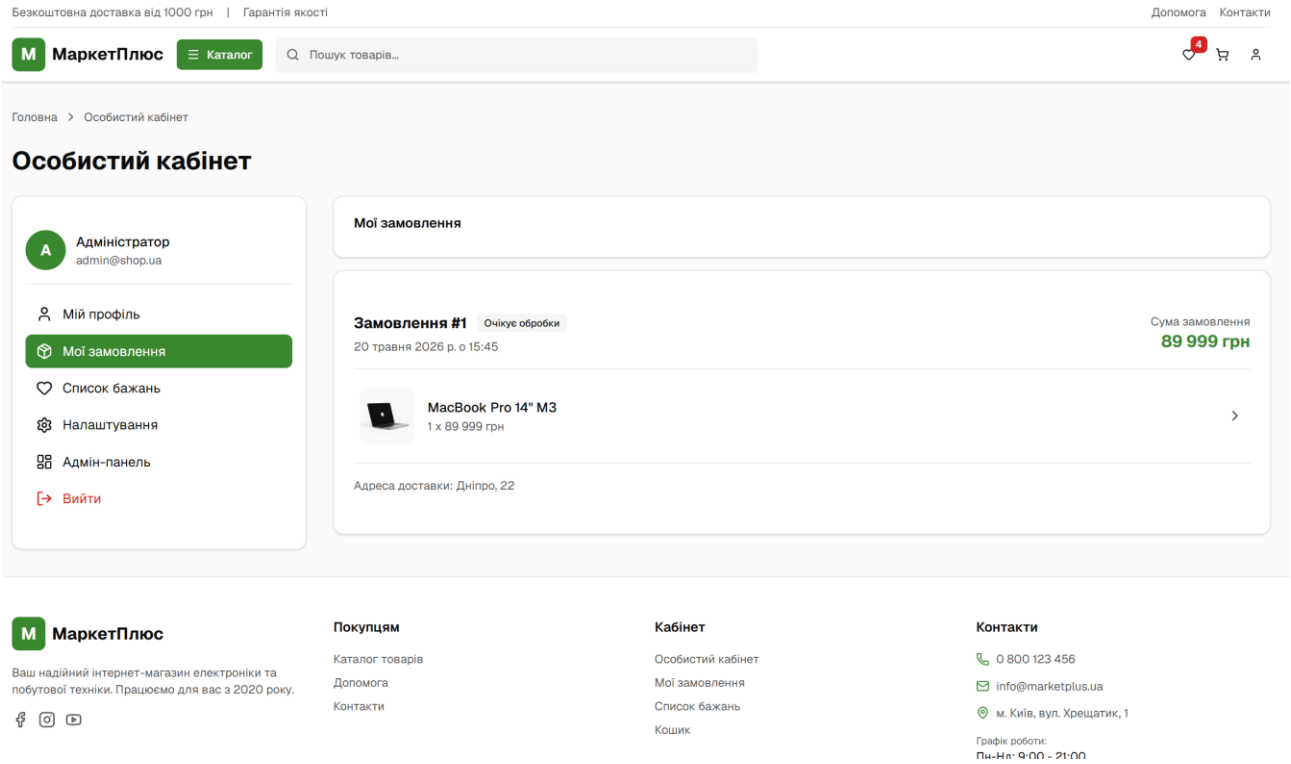
Г.7 – Сторінка кошику



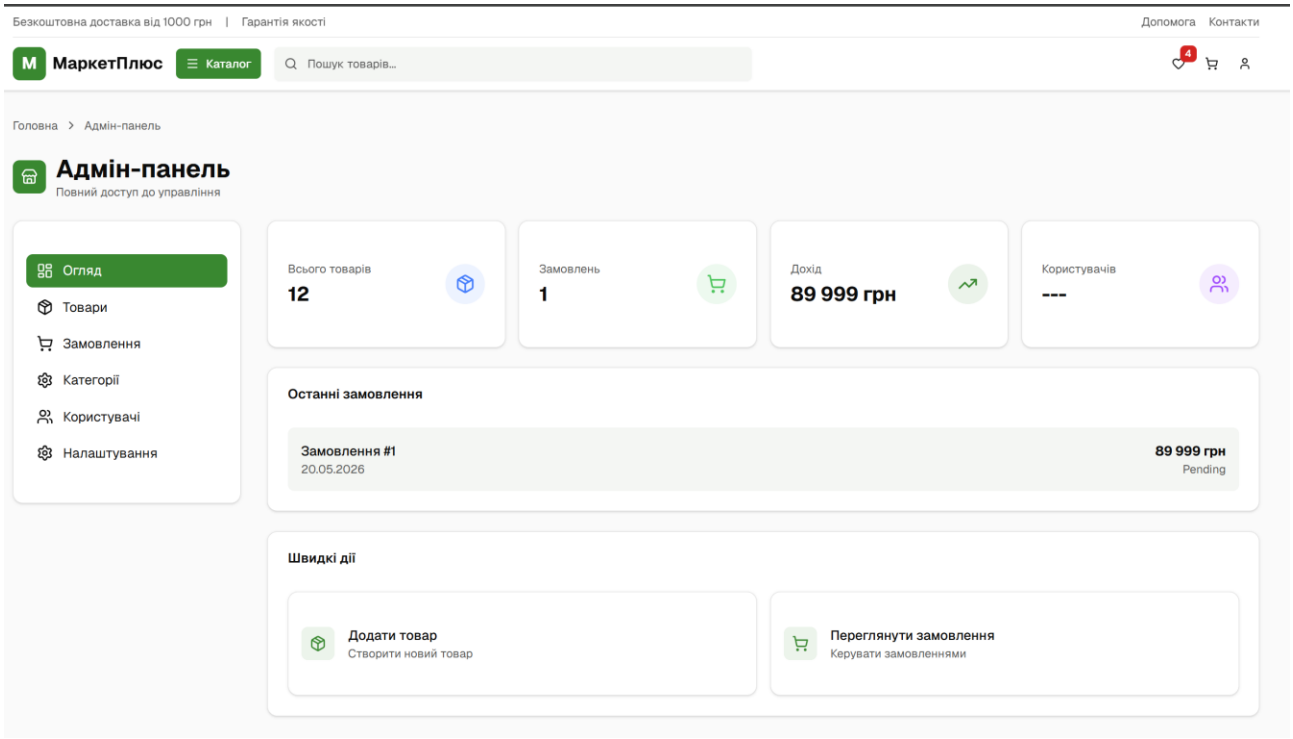
Г.8 – Сторінка оформлення замовлення



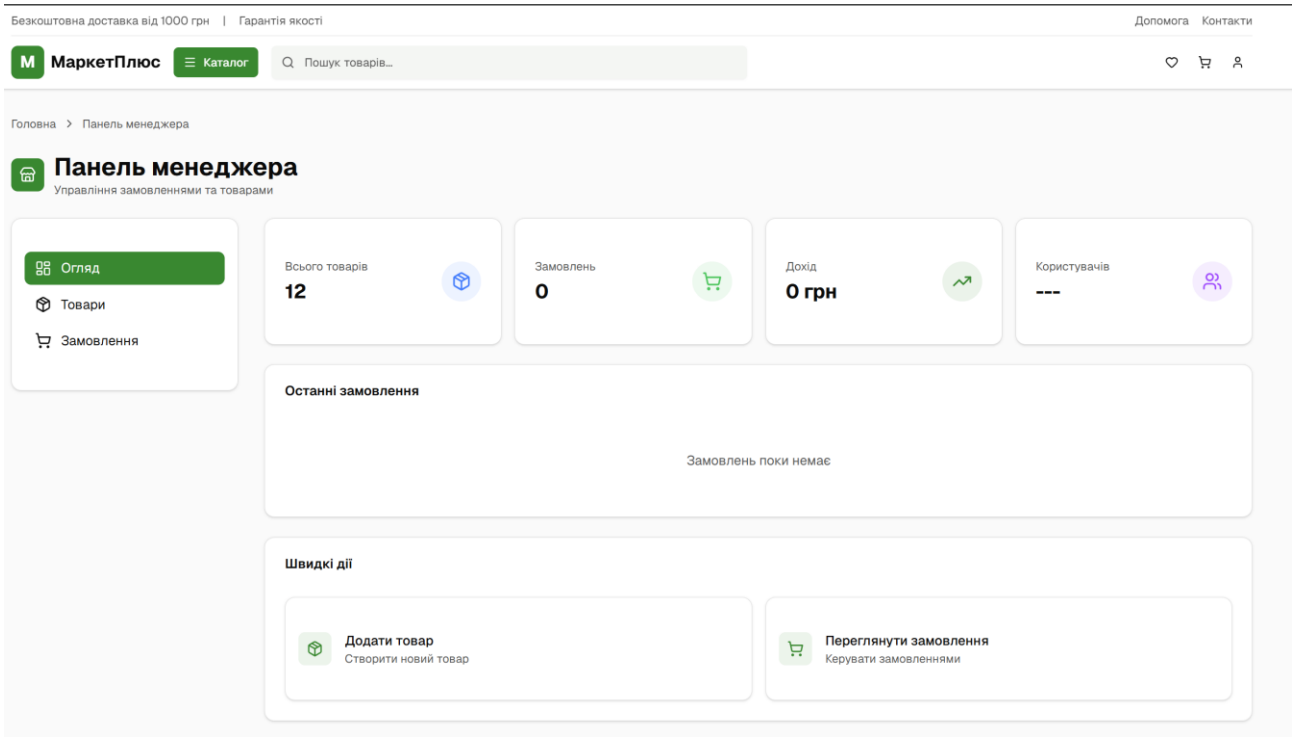
Г.9 – Сторінка особистого кабінету



Г.10 – Сторінка адмін-панелі



Г.11 – Сторінка панелі менеджера



Г.12 – Сторінка контактів

