МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

«Інтернет-магазин жіночого одягу»

Виконав студент групи ІПЗ-21-5
ПЕТРОВСЬКА Катерина Андріївна

Керівник роботи:
ЧИЖМОТРЯ Олена Геннадіївна

Рецензент:
ВЛАСЕНКО Олег Васильович

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»

ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»
Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«14» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **Петровська Катерина Андріївна**
Керівник роботи: **Чижмотря Олена Геннадіївна**
Тема роботи: **«Інтернет-магазин жіночого одягу»,**
затверджена Наказом закладу вищої освіти від **14.02.2025р., №61/с**
Вихідні дані для роботи: Процес автоматизації електронної комерції та організації онлайн-продажу одягу із застосуванням сучасних інформаційних технологій, зокрема розробки вебзастосунку для інтернет-магазину жіночого одягу з інтеграцією інтелектуальних функцій рекомендацій і персоналізації.
Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О.В	20.03.2025	20.03.2025
2	Чижмотря О.В	20.03.2025	20.03.2025
3	Чижмотря О.В	20.03.2025	20.03.2025

РЕФЕРАТ

Кваліфікаційна робота освітнього ступеня «бакалавр» присвячена розробці інтернет-магазину жіночого одягу. Робота складається з програмного продукту та пояснювальної записки. Текстова частина пояснювальної записки до кваліфікаційної роботи містить 62 сторінки, 37 ілюстрацій та 14 таблиць. Загальний обсяг — 79 сторінок.

Метою роботи є створення сучасного, функціонального та зручного інтернет-магазину, який дозволяє користувачам переглядати асортимент жіночого одягу, здійснювати покупки, оформлювати замовлення, а адміністраторам — керувати товарною базою.

У процесі роботи були сформульовані основні завдання для створення системи: розробка модулів для користувачів і адміністраторів, реалізація каталогу з можливістю фільтрації за категоріями, брендом, розмірами та ціною, створення системи кошика, оформлення замовлень і управління наявністю товарів.

Було проведено аналіз існуючих аналогів для визначення функціональних переваг і недоліків. Для розробки обрано клієнт-серверну архітектуру на основі Next.js з використанням серверних функцій (Server Actions), що забезпечують ефективну обробку запитів на стороні сервера. База даних PostgreSQL використовується для зберігання інформації про товари, замовлення, користувачів та інші бізнес-дані.

У роботі описано основні сценарії використання системи, логіку взаємодії між модулями, структуру бази даних, приклади реалізації ключових компонентів інтерфейсу, а також розглянуто особливості адаптивного дизайну для різних типів пристроїв.

Ключові слова: ІНТЕРНЕТ-МАГАЗИН, ЖІНОЧИЙ ОДЯГ, NEXT.JS, POSTGRESQL, SERVER ACTIONS, КАТАЛОГ ТОВАРІВ.

ABSTRACT

The Bachelor's qualification work is dedicated to the development of an online women's clothing store. The project consists of a software product and an explanatory report. The written part of the explanatory report includes 62 pages of text, 37 illustrations, and 14 tables. The total length is 79 pages.

The goal of the project is to create a modern, functional, and user-friendly online store that allows users to browse the assortment of women's clothing, make purchases, and place orders, while administrators can manage the product catalog.

During the project, the main tasks for building the system were defined: development of modules for users and administrators, implementation of a product catalog with filtering by categories, brand, size, and price, creation of a shopping cart system, order placement functionality, and inventory management.

An analysis of existing analogs was carried out to identify their functional advantages and shortcomings. The system was developed using a client-server architecture based on Next.js, with the use of server actions to ensure efficient server-side request handling. A PostgreSQL database is used to store information about products, orders, users, and other business data.

The thesis describes the main usage scenarios of the system, the logic of interaction between modules, the structure of the database, implementation examples of key interface components, and the specifics of responsive design for different device types.

Keywords: ONLINE STORE, WOMEN'S CLOTHING, NEXT.JS, POSTGRESQL, SERVER ACTIONS, PRODUCT CATALOG.

ЗМІСТ

РЕФЕРАТ ***3***

ABSTRACT 4

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 6

ВСТУП 7

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ 9

1.1. Постановка задачі 9

1.2. Аналіз аналогів програмного продукту 10

1.3. Вибір архітектури платформи для надання юридичної допомоги з ші 14

1.4. Обґрунтування вибору інструментальних засобів 15

Висновоки до першого розділу 18

РОЗДІЛ 2. ПРОЄКТУВАННЯ САЙТУ З ПОШУКУ РОБОТИ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТеЛЕКТУ 19

2.1. Визначення варіантів використання та структури системи 19

2.2 Розробка бази даних 23

2.3. Проєктування та реалізація алгоритмів роботи 29

2.4 Реалізація функціоналу 32

Висновки до другого розділу 34

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ДОДАТКУ 35

3.1 Порядок встановлення та налаштування системи 35

3.2 Структура інтерфейсу та порядок взаємодії з інтернет-магазином жіночого одягу 36

3.3 Порядок встановлення та налаштування системи 53

Висновки до третього розділу 57

ВИСНОВКИ 58

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 59

ДОДАТКИ 63

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних

SPA – single page application

SSR – server side render

ВСТУП

Актуальність теми. В умовах стрімкого розвитку інформаційних технологій та цифровізації бізнес-процесів, електронна комерція займає провідні позиції на сучасному ринку. Інтернет-магазини стали невід'ємною частиною торгівлі, пропонуючи зручність, швидкість і широкий асортимент товарів для споживачів. Однією з найбільш популярних галузей електронної комерції є продаж жіночого одягу, оскільки попит на стильні, якісні та доступні речі постійно зростає. Особливу увагу привертає створення таких платформ, які не лише задовольняють запити клієнтів, але й забезпечують конкурентоспроможність бізнесу. Таким чином, тема створення інтернет-магазину жіночого одягу є надзвичайно актуальною, оскільки відповідає сучасним потребам ринку та сприяє розвитку електронної торгівлі.

Метою даної роботи є розробка функціонального, зручного та привабливого інтернет-магазину жіночого одягу, що забезпечить користувачам комфортний досвід здійснення покупок, розширить можливості бізнесу та оптимізує процеси онлайн-продажів. Досягнення цієї мети включає реалізацію інтуїтивно зрозумілого інтерфейсу, ефективної системи пошуку та фільтрації товарів, а також впровадження механізмів для забезпечення безпеки транзакцій.

Для досягнення поставленої мети визначено наступні завдання:

1. Провести аналіз напрямків розвитку електронної комерції та сформулювати постановку задачі для створення інтернет-магазину жіночого одягу.

2. Дослідити існуючі платформи інтернет-торгівлі, визначити їх переваги та недоліки.

3. Обґрунтувати вибір архітектури веборієнтованої системи інтернет-магазину.

4. Визначити інструментальні засоби розробки, мови програмування та вимоги до апаратного забезпечення.

5. Розробити об’єктно-орієнтовану структуру системи та визначити варіанти її використання з урахуванням потреб цільової аудиторії.

6. Спроєктувати базу даних для зберігання інформації про товари, користувачів, замовлення та транзакції.

7. Розробити та реалізувати функціонал пошуку, фільтрації товарів, кошика, оформлення замовлення та обробки платежів.

8. Реалізувати вебдодаток інтернет-магазину жіночого одягу з адаптивним інтерфейсом.

9. Розробити та описати інтерфейс користувача, а також порядок взаємодії з системою.

10. Провести тестування функціональних модулів системи на відповідність вимогам зручності, продуктивності та безпеки.

Об’єктом дослідження є електронна комерція як галузь, зокрема процеси розробки, впровадження та функціонування інтернет-магазинів.

Предметом дослідження є програмні рішення, інструменти та технології, які застосовуються для створення інтернет-магазину жіночого одягу, а також підходи до забезпечення його ефективної роботи, відповідності потребам цільової аудиторії та ринковим вимогам.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1. Постановка задачі

На сьогоднішній день інтернет-магазини стали важливим інструментом для бізнесу, особливо у сфері торгівлі одягом. Власний веб-сайт для підприємця є не лише засобом залучення клієнтів, а й ключовим елементом конкурентоспроможності на ринку. Наявність функціонального та зручного інтернет-магазину жіночого одягу дозволяє компаніям розширювати аудиторію, покращувати взаємодію з клієнтами, підвищувати рівень продажів і забезпечувати постійну присутність на ринку.

Метою створення інтернет-магазину жіночого одягу є розробка платформи, яка надає користувачам можливість легкої навігації, швидкого пошуку товарів, зручного оформлення замовлень і отримання актуальної інформації про продукцію. Окрім того, система повинна забезпечувати високий рівень безпеки даних та відповідати сучасним вимогам дизайну й функціональності.

Інтернет-магазин жіночого одягу зорієнтований на задоволення потреб трьох ключових груп: клієнтів, які шукають якісний і стильний одяг; підприємців, котрі прагнуть автоматизувати продажі й підвищити рентабельність; а також дизайнерів і постачальників, охочих просувати свої товари через сучасні електронні платформи. Окрім зручного користувацького інтерфейсу, важливою складовою є багатофункціональна адміністративна панель, що дає змогу керувати товарами, замовленнями, відгуками та фінансовими операціями. Вона міститиме модуль для додавання, редагування й видалення позицій каталогу, інструменти обробки замовлень зі статусами доставки, систему перегляду та модерації відгуків, а також аналітичний розділ для відстеження продажів і популярності товару.

Для оперативної роботи адміністрації заплановано створення Telegram-бота, який надсилатиме сповіщення про нові замовлення, інформуватиме про зміни статусу доставки й надаватиме інтерактивний інтерфейс для швидких

відповідей клієнтам. Крім цього, бот умітиме формувати стислі звіти про продажі за день, тиждень або місяць.

Реалізація проекту передбачає кілька етапів. Спершу відбудеться проектування дизайну: розробка привабливого вигляду головної сторінки, сторінок каталогу, кошика та оформлення замовлення з адаптацією під різні пристрої. Далі — проектування та розробка архітектури: створення серверної частини для обробки замовлень і зберігання даних, клієнтської частини з акцентом на швидкість та інтерактивність, а також інтеграція адмін-панелі й Telegram-бота. На третьому етапі будуть реалізовані ключові функціональні компоненти: реєстрація й авторизація користувачів, управління каталогом, інтеграція платіжних систем, модулі адміністрування замовлень і товарів та сам бот для адміністраторів. Завершальним кроком стане тестування: перевірка працездатності основних функцій, виправлення помилок і оптимізація продуктивності всієї системи.

1.2. Аналіз аналогів програмного продукту

Перед розробкою інтернет-магазину жіночого одягу важливо проаналізувати існуючі аналоги, які вже функціонують на ринку, для визначення їхніх сильних і слабких сторін. Це дозволить врахувати найкращі практики, уникнути можливих недоліків та створити конкурентоспроможний продукт.

ASOS

ASOS (рис. 1.1) – одна з найвідоміших світових платформ електронної торгівлі, яка пропонує великий вибір одягу, взуття та аксесуарів. Сайт орієнтований на міжнародний ринок і має велику аудиторію користувачів [1].

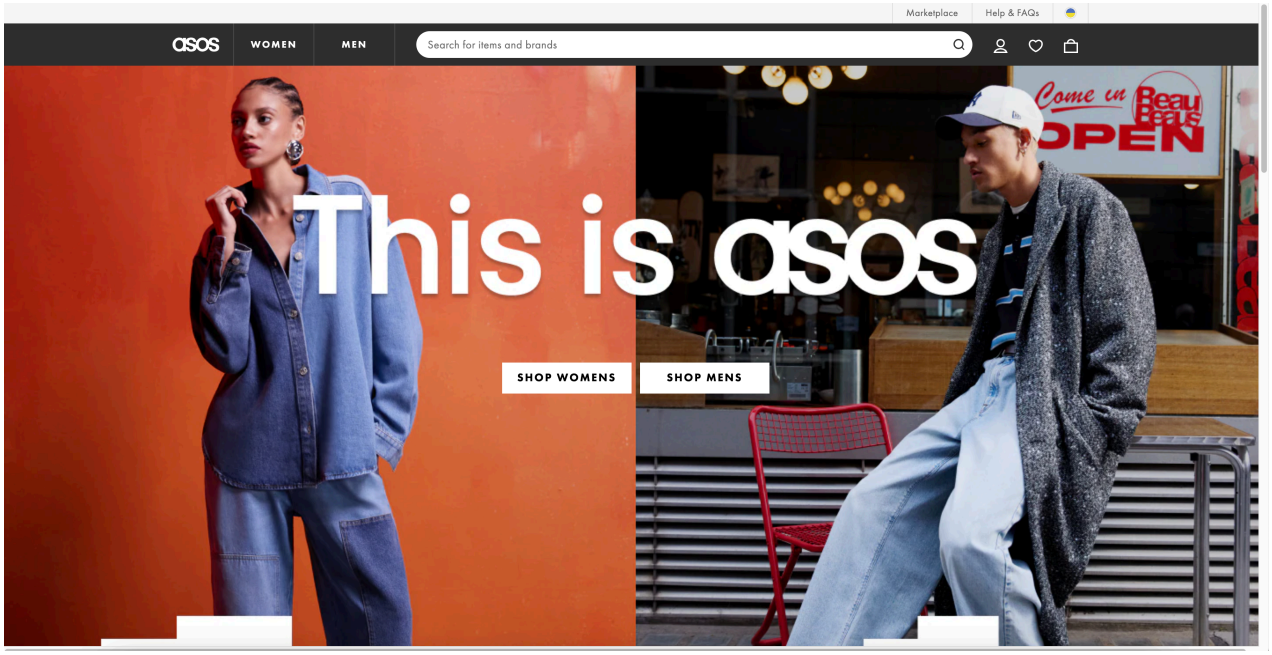


Рис. 1.1 Сайт ASOS

Серед основних переваг інтернет-магазину варто відзначити широкий асортимент товарів, зокрема власні колекції ASOS; високий рівень персоналізації завдяки системі рекомендацій і формуванню готових образів; інтеграцію мультимедійного контенту з відеооглядами, що допомагає краще оцінити характеристики виробів; а також зручну й прозору процедуру повернення та обміну продукції. Водночас інтернет-магазин має й недоліки: відсутність повної локалізації, адже сайт не підтримує українську мову, що ускладнює користування для локальної аудиторії; під час розпродажів виникає значне навантаження на сервери, через що може знижуватися продуктивність; окрім того, інтеграція з локальними платіжними системами наразі залишається обмеженою.

Zara

Zara (рис. 1.2) – це офіційний інтернет-магазин одного з найпопулярніших брендів швидкої моди. Платформа спрямована на продаж актуального одягу для чоловіків, жінок та дітей [2].

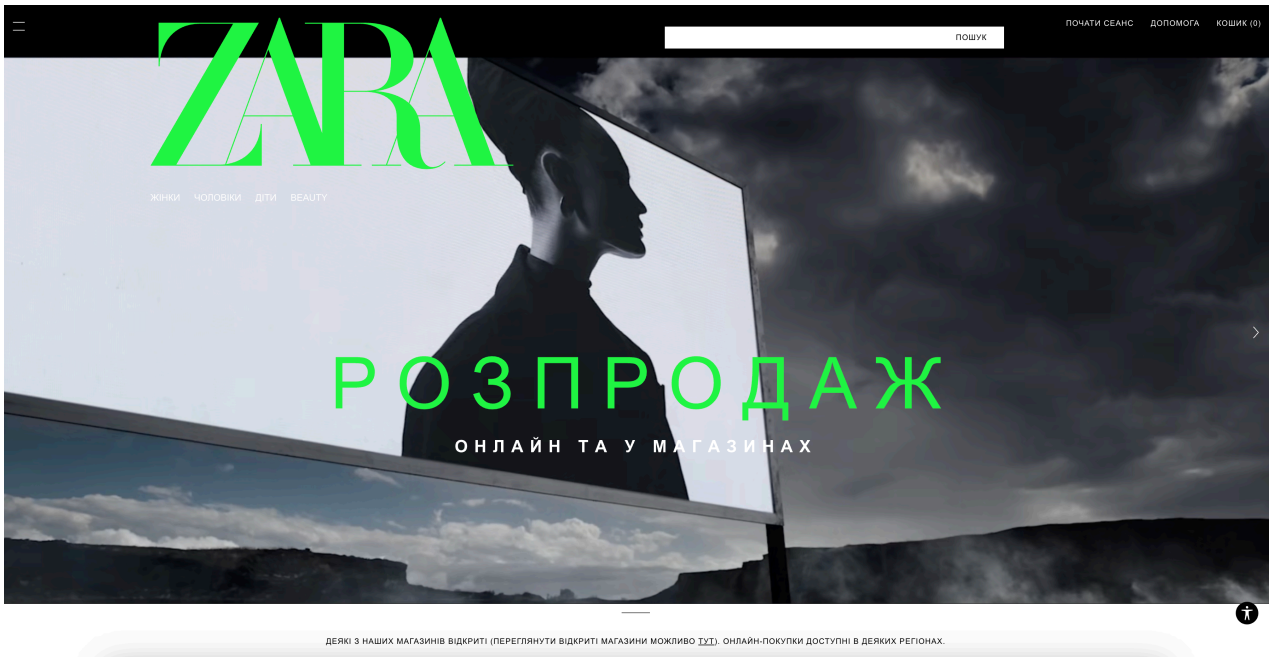


Рис. 1.2 Сайт Zara

Серед ключових переваг інтернет-магазину варто виокремити мінімалістичний і стильний дизайн, який точно віддзеркалює концепцію бренду; зручну можливість миттєво перевіряти наявність обраного товару у фізичних точках продажу; а також регулярне оновлення асортименту, що постійно тримає пропозицію у руслі найактуальніших модних трендів. Утім, магазин має й низку недоліків: особистий кабінет поки пропонує лише базовий функціонал, тож користувачі не можуть зберігати історію покупок чи списки бажань; відсутня розширена система фільтрації, що ускладнює пошук у великих каталогах; до того ж низка можливостей доступна тільки у мобільному додатку, що обмежує тих, хто віддає перевагу веб-версії.

Kasta

Kasta (рис. 1.3) – це українська платформа електронної торгівлі, яка пропонує товари різних категорій: одяг, взуття, аксесуари, косметика, товари для дому тощо. Сайт орієнтований на локальний ринок, пропонуючи акційні пропозиції та швидку доставку [3].

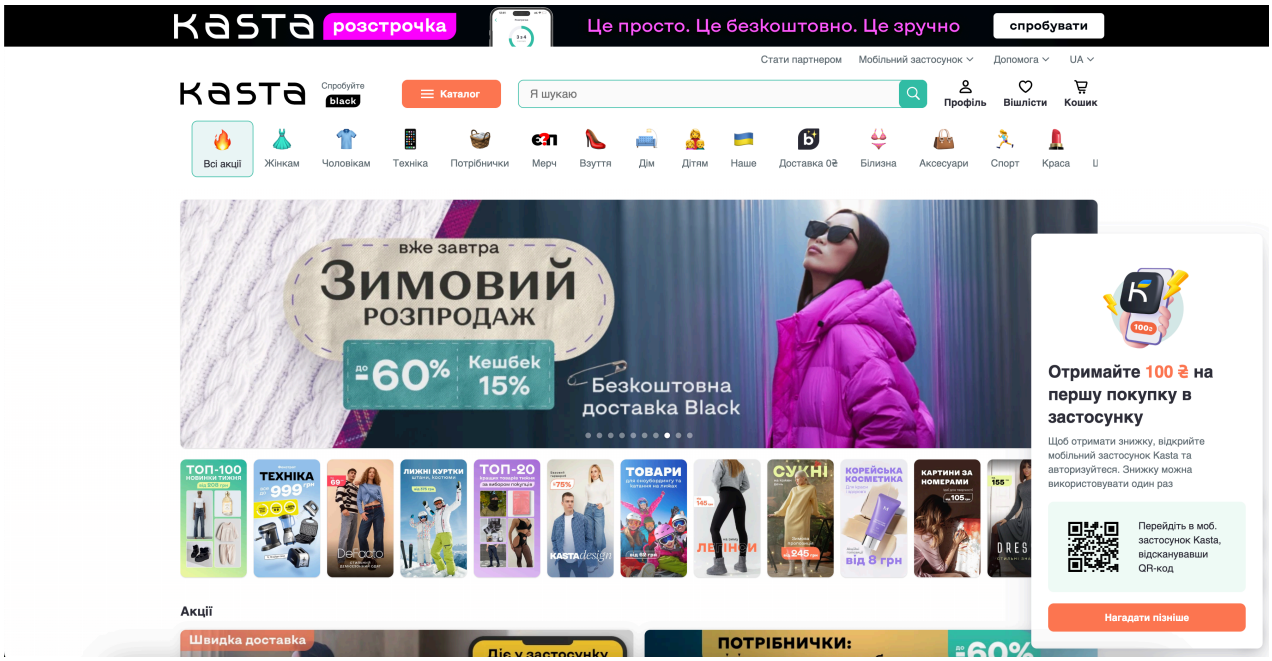


Рис. 1.3 Сайт Kasta

Серед беззаперечних переваг інтернет-магазину варто насамперед відзначити повну локалізацію: сайт доступний українською мовою, тож навігація й оформлення замовлень не викликають труднощів у місцевих користувачів. До плюсів також належить широка інтеграція платіжних систем — окрім банківських карток і післяплати тут працюють Apple Pay та Google Pay, що робить процес оплати максимально зручним. Гнучка система знижок із щоденними акціями до 70 % постійно підтримує інтерес покупців, а особистий кабінет дає змогу переглядати історію замовлень, зберігати вподобані товари й відстежувати доставку. Окремою перевагою є мобільний додаток, який забезпечує швидкий доступ до магазину з будь-якого пристрою, та оперативна доставка: завдяки локальній інфраструктурі товари надходять у короткі строки, зокрема через поштомати. Водночас інтернет-магазин має й недоліки як програмний продукт. Інтерфейс головної сторінки дещо перевантажений інформацією, що може ускладнити пошук потрібного товару для нових відвідувачів. Мультимедійна презентація позицій обмежується переважно фотографіями, тоді як відеоогляди та інтерактивний контент майже відсутні. Значний акцент на знижках іноді формує враження, ніби низька ціна компенсує компроміс у якості, а у пікові періоди — наприклад, під час масштабних

розпродажів — платформа може працювати повільніше, що негативно впливає на користувацький досвід.

Таким чином, аналізуючи ASOS, Zara та Kasta, можна зробити висновок, що жоден із наявних на ринку продуктів не поєднує всі ключові переваги одночасно. Ваш інтернет-магазин успішно вирішує проблему локалізації та адаптації інтерфейсу під українську аудиторію, на відміну від ASOS, і пропонує розширену веб-версію фільтрації та особистий кабінет із історією покупок, на відміну від Zara. Як і Kasta, ви інтегруєте як локальні, так і міжнародні платіжні сервіси, але додатково підсилюєте користувацький досвід AI-генерацією описів та банерів для візуальної презентації товарів. Технічна архітектура на базі Next.js 15 із React Server Components і Prisma забезпечує високу швидкодію й масштабованість, перевершуючи статичні рішення конкурентів. Завдяки продуманій ролевій моделі та зручній адмін-панелі ваш продукт максимально враховує як потреби клієнтів, так і завдання бізнес-адміністрування.

1.3. Вибір архітектури платформи для надання юридичної допомоги з ші

Для розробки інтернет-магазину жіночого одягу було обрано клієнт-серверну архітектуру, яка забезпечує ефективний розподіл ролей між клієнтською та серверною частинами. Клієнтська частина відповідає за веб-інтерфейс, з яким взаємодіють користувачі, та надсилає запити на сервер для отримання та обробки даних. Сервер виконує бізнес-логіку, валідує запити, обробляє дані та зберігає їх у базі даних.

Інтернет-магазин побудований з використанням технології Next.js, яка підтримує сучасний підхід до обробки даних через Server Actions. Такий підхід дозволяє виконувати серверні операції напряму із клієнтської частини, забезпечуючи простоту взаємодії та ефективність обробки даних.

Клієнтська частина реалізує відображення каталогу товарів, кошика, сторінок оформлення замовлення та особистого кабінету; збирає дані користувача — від вибраних позицій і особистої інформації для доставки до залишених відгуків — і надсилає запити на сервер для отримання або оновлення

потрібних відомостей, наприклад щодо поточного статусу замовлення чи опублікованих відгуків.

Серверна частина, у свою чергу, обробляє будь-які клієнтські запити (пошук товарів, створення замовлень, оновлення даних профілю), виконує бізнес-логіку — зокрема рахує підсумкову вартість замовлення з урахуванням знижок і доставки — а також взаємодіє з базою даних, щоб зберігати, оновлювати й видаляти інформацію про користувачів, товари, замовлення та відгуки.

Server Actions [4] у Next.js [5] дозволяють уникнути надмірної залежності від API-запитів, оскільки функціональність викликається безпосередньо в серверному середовищі, інтегруючи бізнес-логіку з клієнтськими зверненнями. Це зменшує кількість проміжних запитів і підвищує загальну продуктивність системи.

Застосування клієнт-серверної архітектури разом із Server Actions гарантує одразу кілька принципових переваг. По-перше, ефективність: взаємодія між клієнтом і сервером оптимізована настільки, що обробка даних відбувається швидко й без зайвих затримок. По-друге, масштабованість: чітке розділення логіки дає змогу легко підтримувати й розширювати функціонал, додаючи нові можливості без радикальних змін у вже працюючих модулях. Нарешті, безпека: чутлива інформація обробляється виключно на сервері, тож ризик її перехоплення мінімізується. У такій архітектурі сервер відіграє ключову роль у реалізації бізнес-логіки та збереженні даних, тоді як клієнтська частина лишається основним інструментом взаємодії з користувачем, забезпечуючи зручний і зрозумілий доступ до можливостей інтернет-магазину.

1.4. Обґрунтування вибору інструментальних засобів

Для створення інтернет-магазину жіночого одягу використано зв’язку Next.js 14, NextAuth, Prisma, React Query, node-telegram-bot-api та Zod [6-10] — набір інструментів, що у комплексі гарантує високу продуктивність, безпеку й зручність розробки. Next.js поєднує серверну та клієнтську логіку завдяки Server

Actions, підтримує SEO, SSR [11] і статичну генерацію сторінок, тож інтерфейс завантажується швидко та безшовно. NextAuth забезпечує просту інтеграцію аутентифікації з популярними провайдерами (Google, Facebook тощо), полегшує керування сесіями й запровадження ролей для адмінів. Prisma дає змогу ефективно взаємодіяти з базою даних через декларативний синтаксис і генерацію типів для TypeScript [12], що мінімізує кількість помилок і пришвидшує розробку. React Query оптимізує роботу з мережевими запитами, автоматично кешуючи й повторно використовуючи дані та централізовано керуючи станами завантаження і помилок. Для Telegram-бота, який слугує адміністративним інтерфейсом магазину, обрано node-telegram-bot-api — бібліотеку з простим API та підтримкою middleware, що забезпечує гнучкість і легку масштабованість. Нарешті, Zod відповідає за валідацію даних, тісно інтегрується з TypeScript і дозволяє будувати складні схеми перевірки, підвищуючи надійність і безпеку всієї системи. Сукупність цих технологій забезпечує сучасний, масштабований та стабільний інтернет-магазин, зручний і для користувачів, і для команди підтримки.

У процесі створення інтернет-магазину жіночого одягу важливим етапом стало визначення бази даних, яка забезпечить зберігання, обробку та управління даними у масштабованій, продуктивній та безпечній системі. Для цього було проведено аналіз трьох популярних СУБД: PostgreSQL, MySQL та MongoDB [13-15], з урахуванням критеріїв продуктивності, безпеки, масштабованості та функціональних можливостей.

Порівняння обсягів даних

Одним із важливих критеріїв вибору стало визначення максимального обсягу даних, які можуть зберігатися у кожній із розглянутих СУБД (табл. 1.1).

Таблиця 1.1

Порівняння обмежень на розміри даних у різних СУБД

Параметр	PostgreSQL	MySQL	MongoDB
Максимальний розмір БД	Необмежено	Необмежено	Необмежено

Продовження табл. 1.1.

Максимальний розмір таблиці/колекції	32 ТБ	256 ТБ	Необмежено
Максимальний розмір рядка/документа	1.6 ТБ	64 КБ	16 МБ

Порівняння безпеки

Для інтернет-магазину безпека даних є критично важливим фактором, оскільки система працює з конфіденційною інформацією клієнтів. У таблиці 1.2 наведено порівняння механізмів безпеки в розглянутих СУБД.

Таблиця 1.2

Порівняння механізмів безпеки в PostgreSQL, MySQL та MongoDB

Параметр	PostgreSQL	MySQL	MongoDB
Аутентифікація користувачів	+	+	+
Захист від brute-force атак	+	-	-
Вбудоване шифрування	+	+	+
Система ролей і прав доступу	+	Обмежена	+

Функціональні можливості

Для вибору оптимальної СУБД також було враховано функціональні можливості, такі як підтримка сучасних типів даних, робота з транзакціями та масштабованість (табл. 1.3).

Таблиця 1.3

Порівняльна характеристика функціональних можливостей СУБД

Параметр	PostgreSQL	MySQL	MongoDB
Тип даних	Об’єктно-реляційний	Реляційний	Документоорієнтований
Підтримка JSON	Повна	Обмежена	Повна
Масштабованість	Висока	Середня	Дуже висока
Робота з транзакціями	Так	Так	Часткова
Гнучкість структури даних	Висока	Середня	Дуже висока

Після проведення аналізу було визначено, що PostgreSQL є оптимальним вибором для інтернет-магазину жіночого одягу. Ця СУБД підтримує складні реляційні запити та напівструктуровані дані, що дозволяє реалізувати пошук, фільтрацію та сортування. Завдяки вбудованим механізмам шифрування та контролю доступу PostgreSQL забезпечує високий рівень безпеки, а підтримка транзакцій гарантує надійну обробку замовлень. Безкоштовна ліцензія та масштабованість роблять її доступним та ефективним рішенням для сучасних веб-систем.

Висновоки до першого розділу

У першому розділі було сформульовано мету та завдання розробки інтернет-магазину жіночого одягу, а також визначено ключові функціональні вимоги до системи. Проведено аналіз існуючих аналогів, що дозволило виокремити актуальні функції, зокрема каталог товарів, кошик, замовлення, профіль користувача та адмін-панель. Обґрунтовано вибір технологій для реалізації клієнтської та серверної частин, включаючи Next.js і PostgreSQL.

РОЗДІЛ 2. ПРОЄКТУВАННЯ САЙТУ З ПОШУКУ РОБОТИ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТеЛЕКТУ

2.1. Визначення варіантів використання та структури системи

Для забезпечення повноцінної роботи інтернет-магазину жіночого одягу було реалізовано низку модулів, які охоплюють управління товарним каталогом, обробку замовлень, роботу зі знижками та взаємодію з покупцями. Платформа дозволяє зберігати та оновлювати інформацію про товари, користувачів, замовлення, акції, що істотно спрощує процес здійснення покупок в онлайн-середовищі. Основний функціонал включає перегляд та фільтрацію товарів, додавання до кошика, оформлення замовлення з вибором способу доставки та оплати, а також реєстрацію й керування особистим профілем. Для адміністрації реалізовано можливості керування товарним складом, категоріями, брендами, знижками, перегляд аналітики продажів та обробку замовлень.

Типи користувачів

У системі передбачено два основних типи користувачів:

Покупець (звичайний користувач) — особа, яка має можливість переглядати асортимент товарів, додавати обрані позиції до кошика, оформляти замовлення, переглядати свою історію покупок і редагувати особисту інформацію.

Адміністратор — користувач із розширеними правами, що керує наповненням каталогу, обробляє замовлення, створює акційні пропозиції, переглядає звіти про продажі та керує користувачами.

На (рис. 2.1) зображено побудовану діаграму базових варіантів використання для кожного типу користувачів.

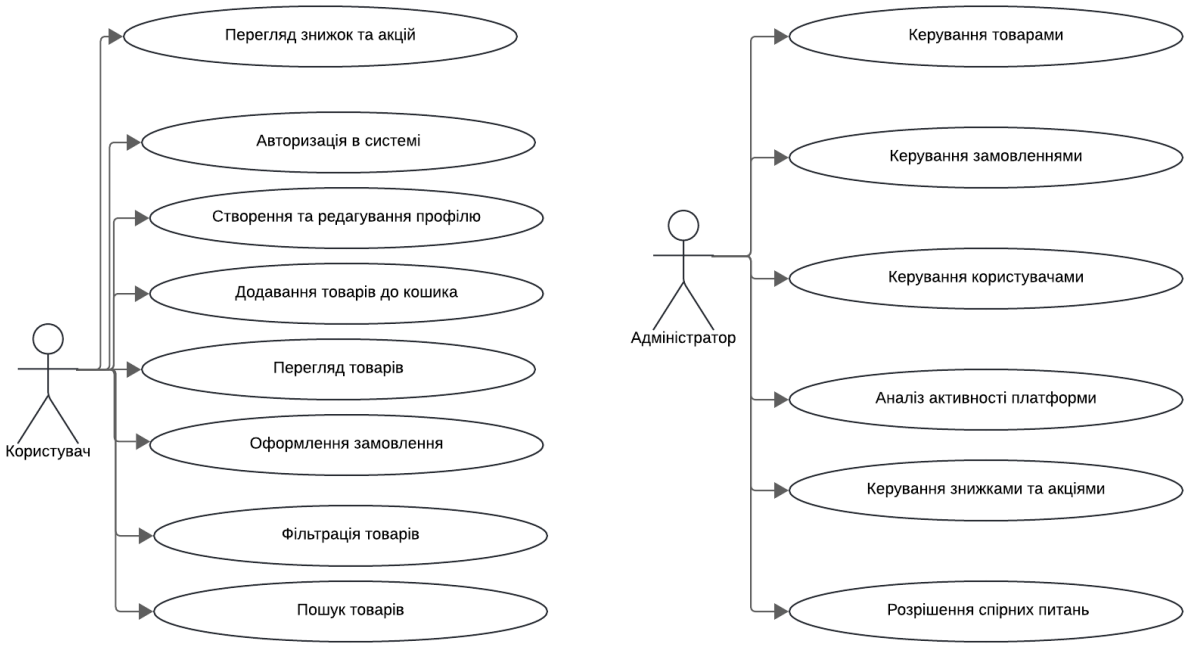


Рис 2.1. Варіанти використання сайту по пошуку роботи з використанням штучного інтелекту

Для покупця передбачено наступні варіанти використання:

- Реєстрація та авторизація в системі
- Перегляд товарів у каталозі
- Фільтрація та пошук товарів
- Перегляд детальної інформації про товар
- Додавання товарів до кошика
- Оформлення замовлення
- Перегляд історії замовлень
- Редагування персонального профілю
- Ознайомлення з акційними пропозиціями

Для адміністратора передбачено такі дії:

- Додавання, редагування та видалення товарів
- Керування категоріями та брендами
- Створення та керування знижками й акціями
- Обробка замовлень (зміна статусів, перегляд деталей)
- Перегляд статистики продажів

- Керування обліковими записами користувачів

У рамках розробки інтернет-магазину жіночого одягу було визначено низку ключових функціональних вимог, які мають забезпечити повноцінну роботу системи та задовольнити потреби як звичайних користувачів, так і адміністраторів. Передусім необхідно реалізувати механізм авторизації з підтримкою реєстрації та входу з різними ролями — звичайний користувач і адміністратор. Кожен зареєстрований клієнт має можливість керувати власними даними: редагувати ім’я, адресу доставки та контактні відомості. Система повинна надавати зручний інтерфейс для перегляду каталогу товарів із розширеними фільтрами за категоріями, брендами, розмірами, ціновими діапазонами та іншими параметрами, а також реалізувати пошук за ключовими словами і сортування результатів. Для комфортного шопінгу користувачам необхідно надати можливість додавати товари до кошика, переглядати його вміст і змінювати кількість чи видаляти позиції перед оформленням замовлення. Процес оформлення має включати вибір способу доставки та оплату через кредитні картки, PayPal або післяплату. Крім того, кожен клієнт повинен мати доступ до власної історії замовлень із деталізацією дати покупки, статусу замовлення та загальної суми. Адміністраторам же слід надати інструменти для управління знижками та акційними пропозиціями, створення та редагування товарних позицій із можливістю коригувати описи, ціни й залишки на складі, а також функціонал для обробки замовлень із можливістю змінювати їхній статус (наприклад, «в обробці», «відправлено», «доставлено»). Виконання цих вимог створить надійну, зручну та масштабовану платформу для ефективного онлайн-продажу жіночого одягу.

Нефункціональні вимоги передбачають, що час навчання користувача роботі із системою не перевищуватиме однієї години, натомість час відгуку на типові дії становитиме не більше десяти секунд. Система має бути максимально надійною та забезпечувати безперервний доступ із мінімальними простоями, а також продуктивною: вона повинна підтримувати не менше ста одночасних користувачів.

Аналіз функціональних вимог дозволив виокремити ключові сутності, необхідні для реалізації основної бізнес-логіки інтернет-магазину жіночого одягу. До них належать користувачі, товари, категорії, замовлення, кошик, адреси доставки, коментарі, список бажаного тощо.

На основі цього виділення було побудовано модель даних, яка враховує взаємозв'язки між об'єктами та їхні атрибути й стала фундаментом для розробки серверної частини застосунку з використанням ORM-бібліотеки Prisma.

На рисунку 2.2 представлено діаграму класів, що ілюструє основні сутності системи, їх атрибути та зв'язки між ними. Зокрема, виділено такі класи, як User, Product, Order, CartItem, Comment, Watchlist, Address, Category, які формують структурну основу платформи.

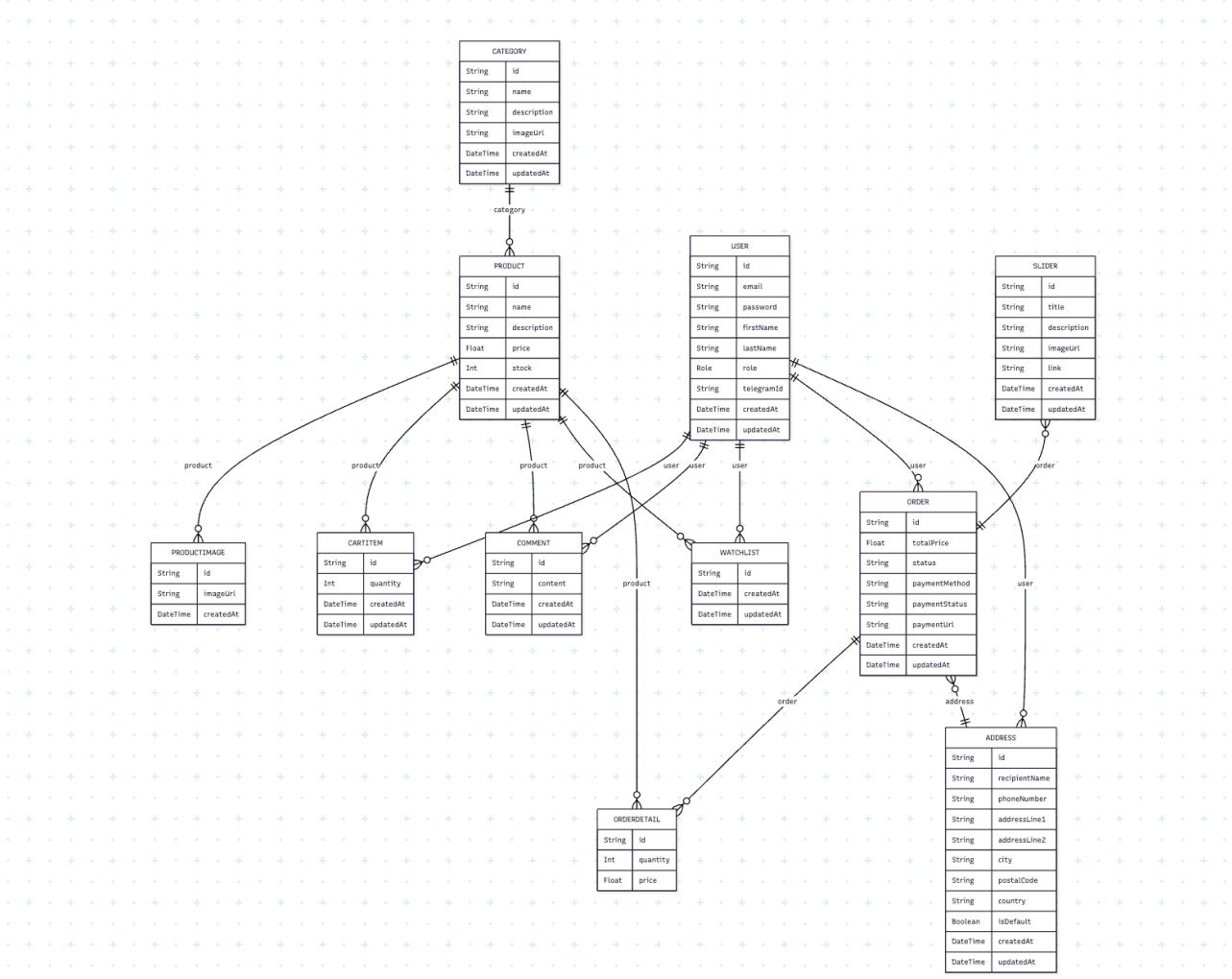


Рис. 2.2 Діаграма класів для інтернет-магазину жіночого одягу

2.2 Розробка бази даних

Одним із ключових етапів реалізації інтернет-магазину жіночого одягу є проєктування та створення реляційної бази даних, яка забезпечує збереження, цілісність та ефективний доступ до інформації про користувачів, товари, замовлення та інші об’єкти системи. База даних повинна підтримувати всі функціональні вимоги, що були визначені на етапі аналізу, та забезпечувати масштабованість системи.

Для розробки бази даних була обрана система керування базами даних PostgreSQL, відзначена високою продуктивністю, підтримкою транзакцій і складних типів даних, а опис структури виконано з використанням ORM-бібліотеки Prisma, що дозволяє декларативно задавати модель даних і автоматично генерувати запити для створення необхідних таблиць. У реалізованій моделі даних виділено такі основні сутності: User для зберігання інформації про користувачів; Address для адрес доставки, пов’язаних із замовленнями; Category — категорії товарів; Product із полями опису, ціни, залишків на складі, прив’язкою до категорій і набором зображень; ProductImage для зберігання окремих зображень товарів; CartItem, що фіксує вміст кошика користувача; Order із даними про користувача, адресу доставки, статусом і способом оплати; OrderDetail, який деталізує кожне замовлення, вказуючи перелік товарів, їхню кількість та ціну; Comment для відгуків користувачів про товари; Watchlist для списку бажаного; а також Slider для банерів та рекламних блоків на головній сторінці.

Для наочного уявлення структури даних, які використовуються у системі, на рисунку 2.3 зображено узагальнену структуру бази даних інтернет-магазину жіночого одягу. Діаграма відображає основні сутності, їх зв’язки між собою, а також логіку організації інформації в системі.

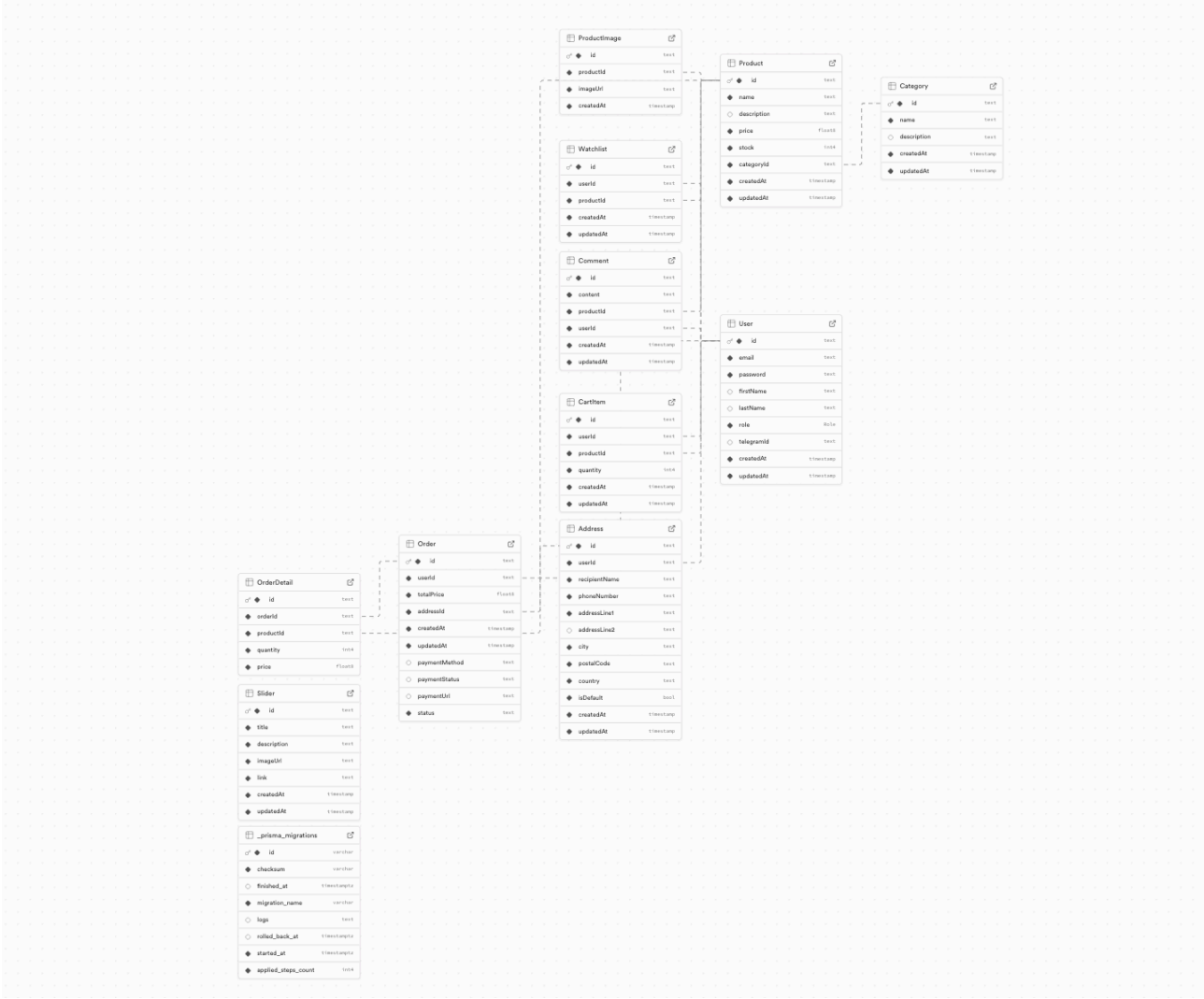


Рис. 2.3 Структура бази даних

Після загального представлення структури, у таблицях нижче наведено детальний опис кожної сутності, включаючи назви полів, типи даних, ознаку первинного ключа (PK), зовнішнього ключа (FK), а також їх призначення.

Таблиця 2.1

Структура сутності "Користувач"

Назва поля	Тип даних	PK	FK	Опис
id	String	+	–	Унікальний ідентифікатор користувача
email	String	–	–	Електронна пошта
password	String	–	–	Пароль користувача (захищений)
firstName	String	–	–	Ім’я користувача

Продовження табл. 2.1

lastName	String	–	–	Прізвище користувача
role	Enum	–	–	Роль користувача (USER, ADMIN, SUPER_ADMIN)
telegramId	String	–	–	Ідентифікатор Telegram (для адміністраторів)
createdAt	DateTime	–	–	Дата реєстрації
updatedAt	DateTime	–	–	Дата останнього оновлення профілю

Таблиця 2.2

Структура сутності "Адреса доставки"

Назва поля	Тип даних	PK	FK	Опис
id	String	+	–	Унікальний ідентифікатор адреси
userId	String	–	+	Ідентифікатор користувача, якому належить адреса
recipientName	String	–	–	Ім’я отримувача
phoneNumber	String	–	–	Номер телефону отримувача
addressLine1	String	–	–	Основна адреса
addressLine2	String	–	–	Додаткова адреса (необов’язково)
city	String	–	–	Місто
postalCode	String	–	–	Поштовий індекс
country	String	–	–	Країна (за замовчуванням — Україна)
isDefault	Boolean	–	–	Ознака основної адреси
createdAt	DateTime	–	–	Дата створення адреси
updatedAt	DateTime	–	–	Дата останнього оновлення

Таблиця 2.3

Структура сутності "Категорія товарів"

Назва поля	Тип даних	PK	FK	Опис
id	String	+	–	Унікальний ідентифікатор категорії

Продовження табл. 2.3

name	String	–	–	Назва категорії (унікальна)
description	String?	–	–	Опис категорії
imageUrl	String?	–	–	Посилання на зображення категорії
createdAt	DateTime	–	–	Дата створення
updatedAt	DateTime	–	–	Дата оновлення

Таблиця 2.4

Структура сутності "Товар"

Назва поля	Тип даних	PK	FK	Опис
id	String	+	–	Унікальний ідентифікатор товару
name	String	–	–	Назва товару
description	String?	–	–	Опис товару
price	Float	–	–	Ціна товару
stock	Int	–	–	Кількість на складі
categoryId	String	–	+	Ідентифікатор категорії, до якої належить товар
createdAt	DateTime	–	–	Дата створення товару
updatedAt	DateTime	–	–	Дата останнього оновлення

Таблиця 2.5

Структура сутності "Зображення товару"

Назва поля	Тип даних	PK	FK	Опис
id	String	+	–	Унікальний ідентифікатор зображення
productId	String	–	+	Ідентифікатор товару
imageUrl	String	–	–	URL зображення товару

Таблиця 2.6

Структура сутності "Замовлення"

Назва поля	Тип даних	PK	FK	Опис
id	String	+	–	Унікальний ідентифікатор запису в кошику
userId	String	–	+	Ідентифікатор користувача

Продовження табл. 2.6

productId	String	–	+	Ідентифікатор товару
quantity	Int	–	–	Кількість одиниць товару у кошику
createdAt	DateTime	–	–	Дата створення запису
updatedAt	DateTime	–	–	Дата оновлення запису

Таблиця 2.7

Структура сутності "Позиція замовлення"

Назва поля	Тип даних	PK	FK	Опис
id	String	+	–	Унікальний ідентифікатор замовлення
userId	String	–	+	Ідентифікатор користувача
addressId	String	–	+	Ідентифікатор адреси доставки
totalPrice	Float	–	–	Загальна сума замовлення
status	String	–	–	Статус замовлення (наприклад, PENDING, COMPLETED)
paymentMethod	String?	–	–	Метод оплати (готівка, картка тощо)
paymentStatus	String?	–	–	Статус оплати
paymentUrl	String?	–	–	Посилання для онлайн-оплати
createdAt	DateTime	–	–	Дата створення замовлення
updatedAt	DateTime	–	–	Дата оновлення

Таблиця 2.8

Назва поля	Тип даних	PK	FK	Опис
id	String	+	–	Ідентифікатор запису позиції у замовленні
orderId	String	–	+	Ідентифікатор замовлення
productId	String	–	+	Ідентифікатор товару
quantity	Int	–	–	Кількість одиниць товару
price	Float	–	–	Ціна одиниці товару

Таблиця 2.9

Структура сутності "Коментар до товару"

Назва поля	Тип даних	PK	FK	Опис
id	String	+	–	Ідентифікатор коментаря
content	String	–	–	Текст коментаря
productId	String	–	+	Ідентифікатор товару
userId	String	–	+	Ідентифікатор користувача
createdAt	DateTime	–	–	Дата створення коментаря
updatedAt	DateTime	–	–	Дата останнього редагування

Таблиця 2.10

Структура сутності "Список бажаного"

Назва поля	Тип даних	PK	FK	Опис
id	String	+	–	Унікальний ідентифікатор запису
userId	String	–	+	Ідентифікатор користувача
productId	String	–	+	Ідентифікатор товару
createdAt	DateTime	–	–	Дата додавання до списку бажаного
updatedAt	DateTime	–	–	Дата останнього оновлення

Таблиця 2.11

Структура сутності "Слайдер/банер"

Назва поля	Тип даних	PK	FK	Опис
id	String	+	–	Унікальний ідентифікатор слайдера
title	String	–	–	Заголовок банера
description	String	–	–	Короткий опис
imageUrl	String	–	–	Зображення банера
link	String	–	–	Посилання, на яке веде банер
createdAt	DateTime	–	–	Дата створення
updatedAt	DateTime	–	–	Дата оновлення

Використовуючи такі таблиці, як User, Product, Order, CartItem та Address, було сформовано логіку зберігання основної інформації про покупців, асортимент товарів, структуру замовлень та адреси доставки. Рольову модель побудовано на основі поля role, що дозволяє обмежувати або надавати доступ до адміністративних функцій відповідно до типу користувача. Завдяки таблицям Comment та Watchlist реалізовано можливості для залишення відгуків на товари та формування списку бажаного. Таблиці Category та Slider забезпечують гнучку класифікацію товарів і підтримку маркетингового контенту у вигляді банерів, що відображаються на головній сторінці. Така структура дозволяє масштабувати систему, підтримувати персоналізацію інтерфейсу та впроваджувати інтелектуальні рекомендації в майбутньому.

2.3. Проєктування та реалізація алгоритмів роботи

Для реалізації вебзастосунку інтернет-магазину жіночого одягу була обрана клієнт-серверна архітектура на основі сучасного фреймворку Next.js 15. Такий підхід забезпечує оптимальний розподіл навантаження між клієнтом і сервером, підтримку серверного рендерингу (SSR), статичної генерації сторінок (SSG), а також найновіших можливостей рендерингу в режимі React Server Components (RSC).

Ключовою особливістю реалізації системи стало використання Server Actions — вбудованого механізму Next.js, який дозволяє викликати серверну логіку напряму з компонентів інтерфейсу. Це дозволяє значно спростити структуру API-запитів та підвищити безпеку передачі чутливих даних.

Весь функціонал реалізовано з урахуванням ролей користувачів. Після входу в систему, користувач отримує роль (USER або ADMIN), і на основі цього формується доступ до окремих частин функціоналу. Загальний алгоритм взаємодії користувача з системою подано на рисунку 2.4.

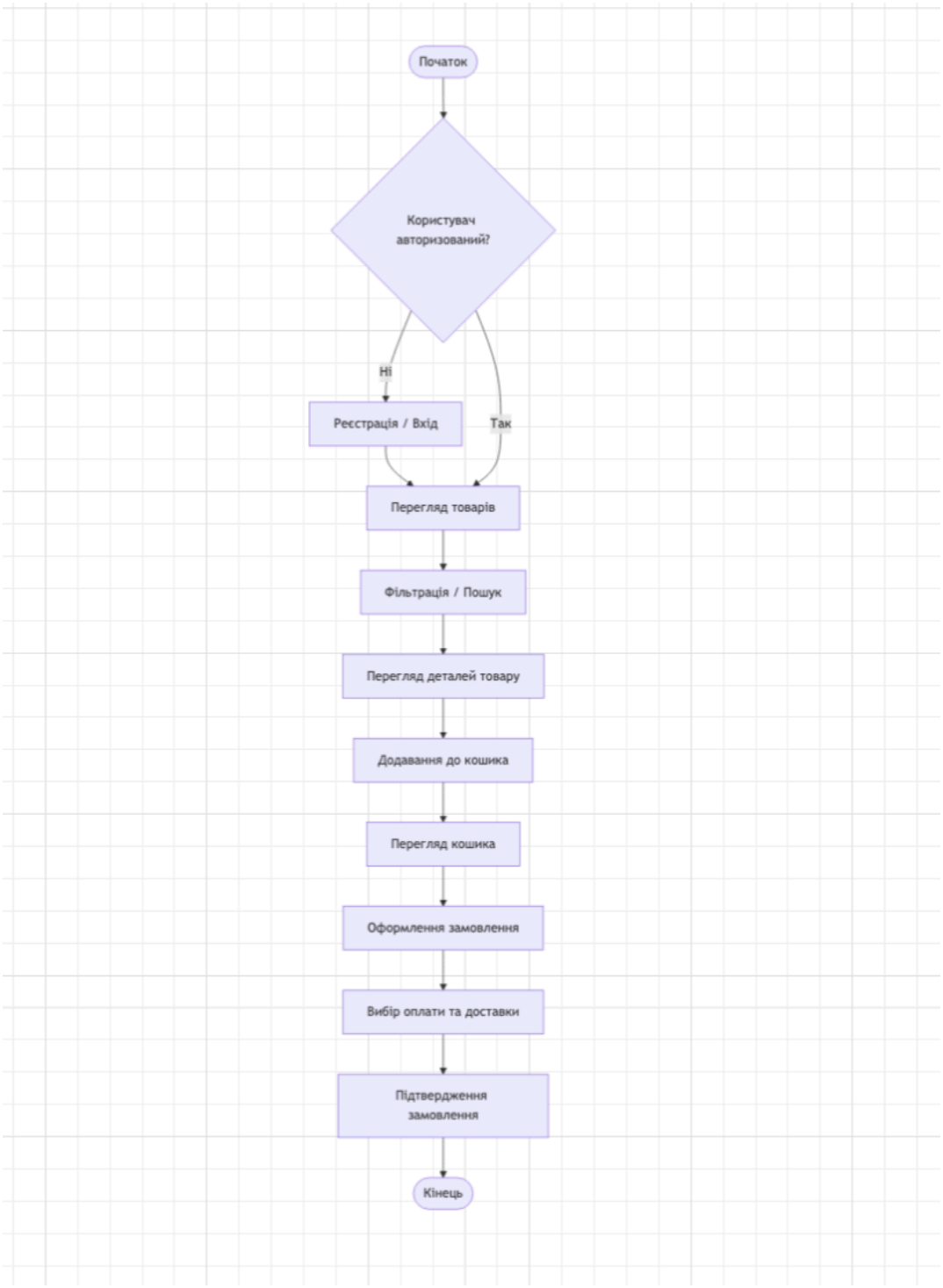


Рис. 2.4 Загальний алгоритм роботи інтернет-магазину жіночого одягу

Для детальнішого розуміння логіки взаємодії між основними компонентами системи було складено діаграму послідовності, яка описує процес оформлення замовлення. У ній відображено ключові дії користувача – від додавання товару до кошика до підтвердження замовлення, а також взаємодію з модулями товарів, кошика, замовлень і адміністрування.

Ця діаграма дозволяє простежити порядок виконання запитів, передачу даних між об’єктами системи та логіку зміни статусу замовлення. Її побудова є важливим етапом проєктування, що сприяє точнішій реалізації бізнес-логіки застосунку та мінімізації помилок у процесі розробки.

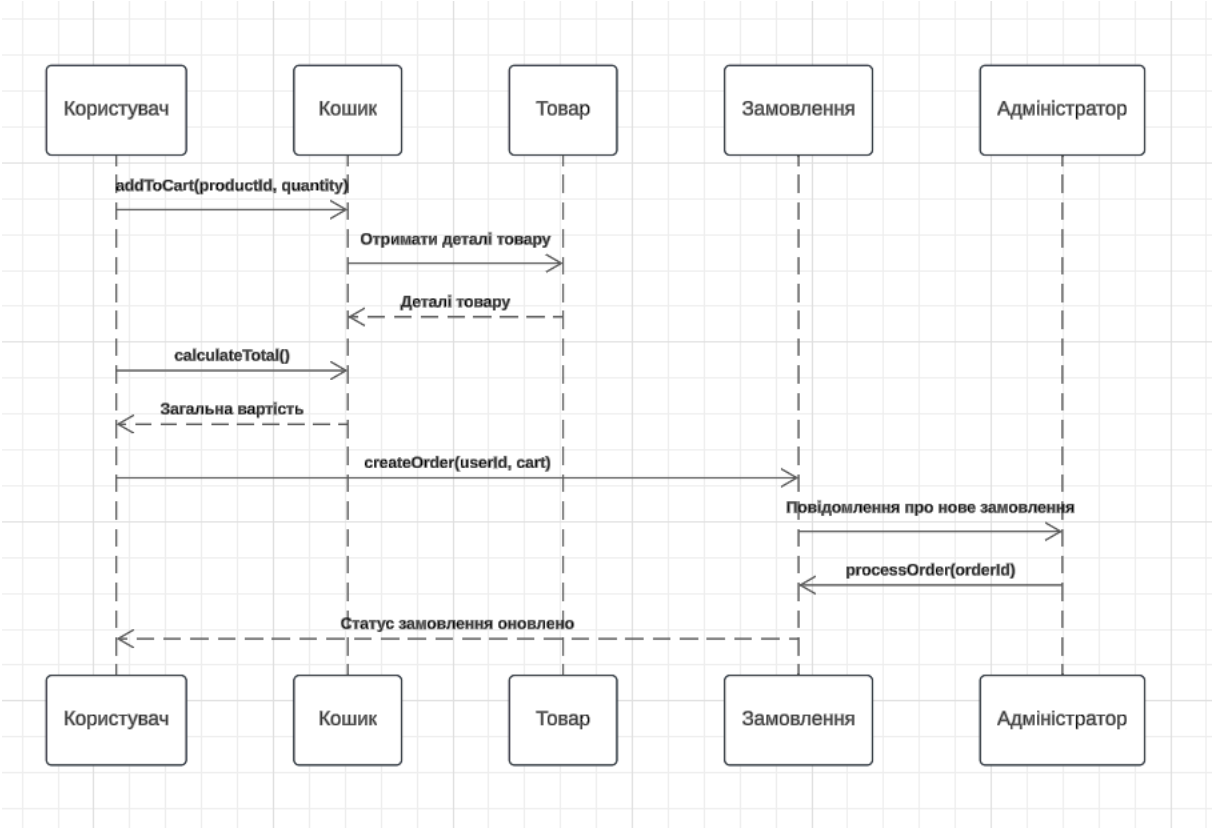


Рис. 2.5 Діаграма послідовності: оформлення замовлення

На діаграмі послідовності, зображеній на рисунку 2.5, представлено логіку процесу оформлення замовлення в інтернет-магазині жіночого одягу. Взаємодія відбувається між п’ятьма основними об’єктами: користувачем, модулем кошика, модулем товарів, підсистемою замовлень та адміністратором.

Процес розпочинається з того, що користувач додає товар до кошика. У відповідь система ініціює запит до модуля товарів з метою отримання необхідної інформації про обраний товар, зокрема його назву, опис, вартість та наявність. Після успішного отримання цих даних вони повертаються до кошика, де формується його актуальний вміст.

На наступному етапі виконується розрахунок загальної вартості всіх товарів, що були додані до кошика. Отримана сума відображається користувачу

для підтвердження замовлення. Після цього система створює нове замовлення, яке містить інформацію про користувача та список товарів з кошика. Після успішного створення замовлення відповідне повідомлення автоматично надсилається адміністратору для подальшої обробки.

Адміністратор, отримавши інформацію про нове замовлення, переходить до його перегляду, перевірки даних і зміни статусу відповідно до внутрішніх процедур (наприклад, підтвердження, обробка, відправлення). Після завершення обробки, оновлений статус замовлення передається назад у систему, і користувач отримує підтвердження про поточний стан замовлення.

2.4 Реалізація функціоналу

Реалізація інтернет-магазину жіночого одягу виконана за допомогою фреймворку Next.js 15, який поєднує функціональність серверного та клієнтського рендерингу, що дозволило досягти високої швидкодії та гнучкості у розробці. Архітектура застосунку базується на розділенні логіки між API-роутами, Server Actions та React Server Components.

Дані зберігаються у реляційній базі даних PostgreSQL, для взаємодії з якою використовується ORM Prisma. Такий підхід спрощує роботу з моделями, забезпечує типізацію та підтримку транзакцій, що особливо важливо для операцій із замовленнями, кошиком та оплатою.

Аутентифікація користувачів реалізована за допомогою бібліотеки NextAuth.js, яка дозволяє підтримувати як традиційний вхід з email/паролем, так і OAuth-провайдери. Крім того, використовується збереження сесій і обмеження доступу до окремих сторінок залежно від ролі користувача. Ролі задаються на рівні бази даних через Prisma та передаються у токен доступу.

Фінансові операції (оплата замовлень) реалізовані через інтеграцію з платіжною системою Stripe. Після підтвердження вмісту кошика, користувач перенаправляється на сторінку оплати, сформовану через API Stripe [16]. По завершенні транзакції дані повертаються на сервер, де обробляються за допомогою вебхуків і зберігаються в базі даних.

Для збереження та обробки даних у проєкті використовується ORM Prisma, що забезпечує зручну роботу з реляційною базою даних PostgreSQL. Усі запити проходять обов'язкову валідацію вхідних даних і перевірку ролей користувачів, а оновлення кешу за допомогою функції revalidatePath гарантує своєчасне відображення актуальної інформації на сторінках. Серверна логіка реалізує повний цикл CRUD-операцій для товарів: створення, отримання, оновлення та видалення, при цьому для кожної дії передбачено централізовану перевірку прав доступу та завантаження зображень у вигляді масиву URL-адрес. Аналітичний і компактний підхід мінімізує дублювання коду та забезпечує чітке розмежування функціональних обов'язків; детальний приклад реалізації цього модуля наведено у Додатку А.

Окрему увагу в системі було приділено реалізації механізму аутентифікації та авторизації користувачів. Для цього було використано бібліотеку NextAuth.js, яка дозволяє гнучко керувати сесіями, підтримує різні типи автентифікаційних провайдерів і легко інтегрується з Next.js-архітектурою. У даному проєкті використано провайдер облікових даних (CredentialsProvider), що реалізує класичну форму авторизації через email та пароль. Після успішного входу відбувається створення JWT-токена, у якому зберігається інформація про користувача, включаючи його роль, що дозволяє ефективно контролювати доступ до ресурсів застосунку.

Конфігурація NextAuth визначає логіку провайдера облікових даних із полями email та password для форми входу та містить функцію authorize, яка викликає внутрішню дію login для перевірки облікових даних через API і, у разі успіху, повертає об'єкт із ключовими даними користувача (id, email, роль) для додавання до JWT-токена та сесії. Параметри сесії налаштовані таким чином, щоб забезпечити баланс між зручністю користувача та безпекою даних, а роль, закодована в токені, використовується в middleware та API-роутах для чіткого розмежування прав доступу до різних частин застосунку. Повна реалізація наведена в Додатку А.

На сайті реалізовано повноцінну інтеграцію з платіжною платформою Stripe для обробки онлайн-оплат під час оформлення замовлення. Після підтвердження вмісту кошика клієнт автоматично перенаправляється на вбудовану форму оплати, реалізовану через компонент PaymentElement, що забезпечує єдиний безпечний інтерфейс для введення платіжних даних.

На сервері при кожному новому замовленні створюється об’єкт PaymentIntent із вказаною сумою та валютою, що ініціює платіжну сесію. У відповідь клієнт отримує clientSecret, який використовується для підтвердження транзакції на стороні клієнта. Весь процес супроводжується ретельною обробкою можливих помилок — від некоректних деталей картки до невдалої авторизації платежу. Після успішного підтвердження дані про оплату передаються назад на бекенд, де за необхідності зберігаються у базі даних, а також обробляються через Stripe API для подальшого звітування та аналітики.

Використання PaymentElement гарантує адаптивність інтерфейсу й автоматичне оновлення безпеки відповідно до вимог платіжного провайдера, що значно спрощує розробку та зменшує ризик помилок. Такий підхід дозволяє реалізувати весь цикл обробки оплати в декілька кроків із мінімальним залученням користувача та повною інтеграцією зі Stripe.

Висновки до другого розділу

У другому розділі було спроєктовано архітектуру інтернет-магазину з урахуванням ролей користувачів і взаємодії між модулями. Розроблено структуру бази даних на основі бізнес-вимог, що забезпечує збереження інформації про товари, замовлення, користувачів та адміністративні дії. Також описано ключові компоненти коду, відповідальні за обробку замовлень, повідомлень і динамічну генерацію контенту.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ДОДАТКУ

3.1 Порядок встановлення та налаштування системи

Діаграма розгортання виступає як візуальне уявлення конфігурації вузлів обробки та компонентів системи під час їх встановлення. Це один із типів структурних діаграм, який дозволяє відобразити фізичне розміщення програмних компонентів у середовищі розгортання, а також взаємозв'язки між ними. На рисунку 3.1 представлено діаграму розгортання для інтернет-магазину жіночого одягу, розробленого з використанням фреймворку Next.js, бази даних MongoDB та платіжного шлюзу Stripe. Система реалізована за архітектурою клієнт-сервер, з чітким поділом на фронтенд, бекенд, базу даних і зовнішній сервіс оплати.

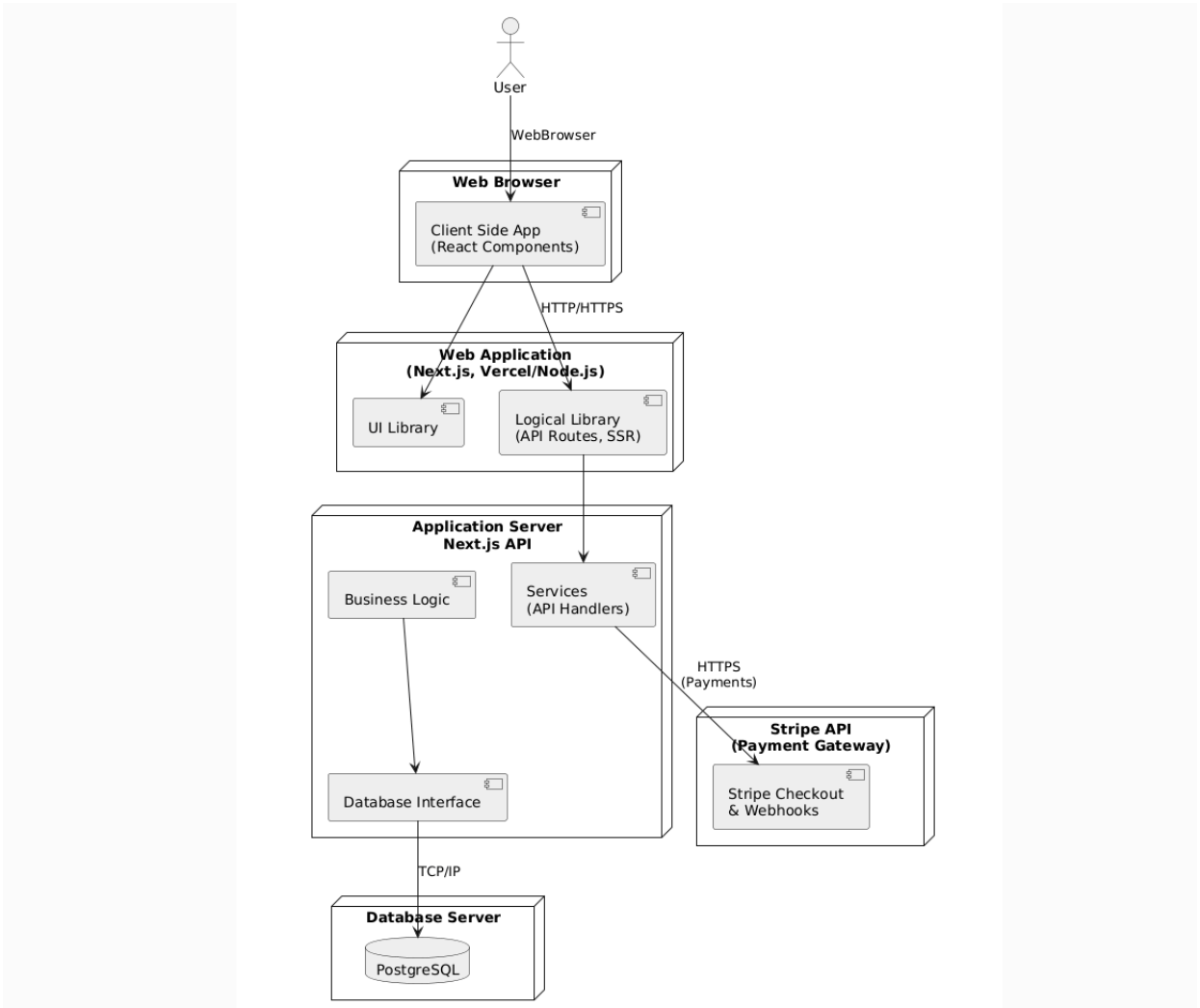


Рис. 3.1 Діаграма послідовності: оформлення замовлення

Процес розгортання системи починається з клієнтської частини, яку становить веб-браузер: тут без встановлення будь-якого додаткового ПЗ відображаються React-компоненти, які на льоту звертаються до сервера через API. Інтерфейс і логіка додатку реалізовані на платформі типу Vercel за допомогою Next.js, що забезпечує сервер-сайд рендеринг, маршрутизацію та власні API-маршрути для авторизації, перегляду товарів і оформлення замовлень. Уся бізнес-логіка обробляється на рівні серверного застосунку, який отримує HTTP-запити, виконує контролери API, перевіряє наявність товарів у наявності й взаємодіє з базою даних через ORM Prisma. Сервер бази даних PostgreSQL працює окремо та зберігає інформацію про користувачів, товари, замовлення й інші сутності, а обмін з ним відбувається через TCP/IP. Для обробки платежів система інтегрується з зовнішнім сервісом Stripe: сервер надсилає HTTPS-запити для створення й підтвердження транзакцій, а також обробляє webhook-події, які надходять від Stripe після зміни статусу платежу.

3.2 Структура інтерфейсу та порядок взаємодії з інтернет-магазином жіночого одягу

Продемонструємо інтерфейс системи на прикладі основного потоку використання. Після відкриття сайту користувач потрапляє на головну сторінку інтернет-магазину (рис. 3.2), де представлено список товарів, фільтри за категоріями, акційні пропозиції та банери. Звідси він може перейти до перегляду

товару, додати його в кошик або продовжити пошук.

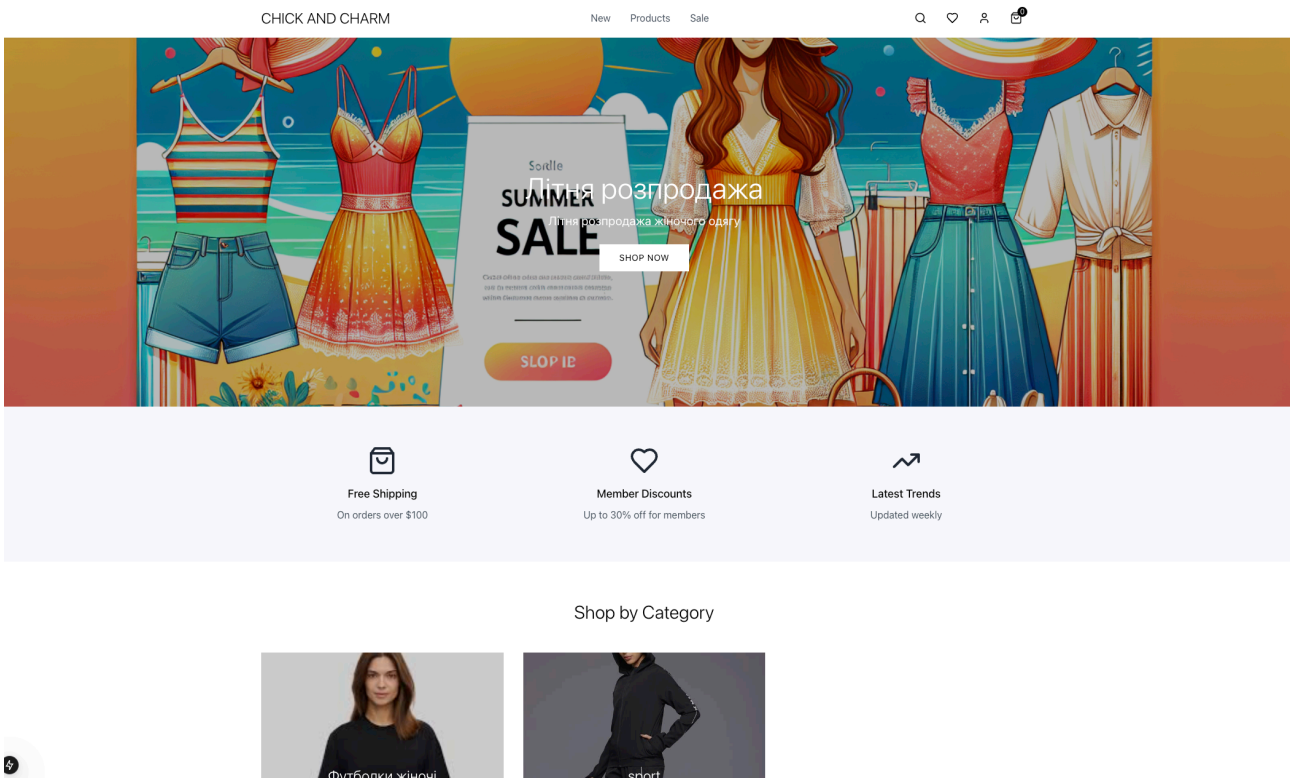


Рис. 3.2 Головна сторінка

Для початку взаємодії з інтернет-магазином користувач має можливість авторизуватися, натиснувши кнопку «Вхід» у верхній частині сайту. Після цього відкривається сторінка авторизації, на якій користувач вводить свої облікові дані (електронну пошту та пароль) для доступу до персонального кабінету. Інтерфейс цієї сторінки наведено на рисунку 3.3.

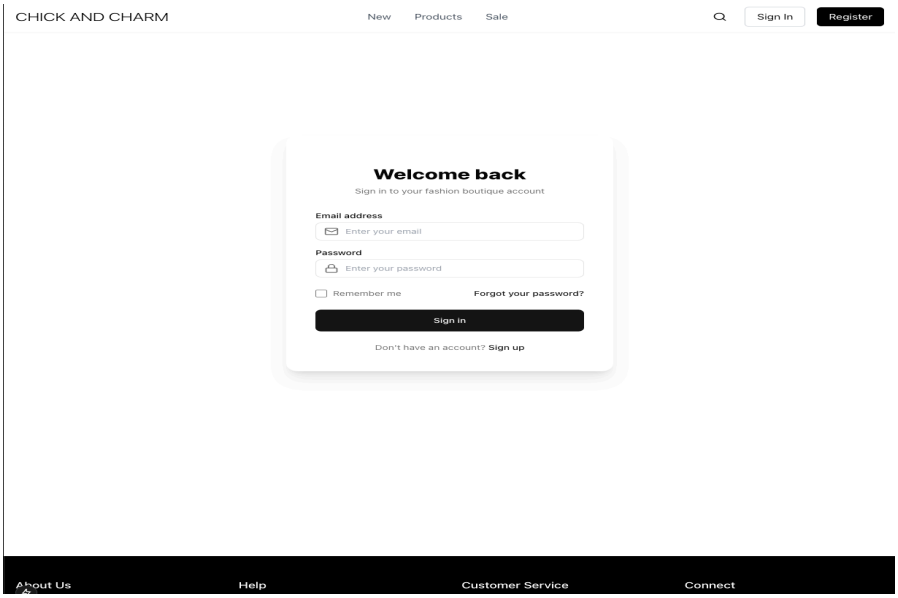


Рис. 3.3 Сторінка входу

Якщо користувач ще не має облікового запису, він може натиснути кнопку «Register» у правому верхньому куті сторінки. Після цього відкривається форма реєстрації, де потрібно ввести ім’я, прізвище, електронну пошту та пароль. У разі, якщо акаунт уже створений, можна скористатися кнопкою «Sign In» для входу. Інтерфейс сторінки реєстрації показано на рисунку 3.4.

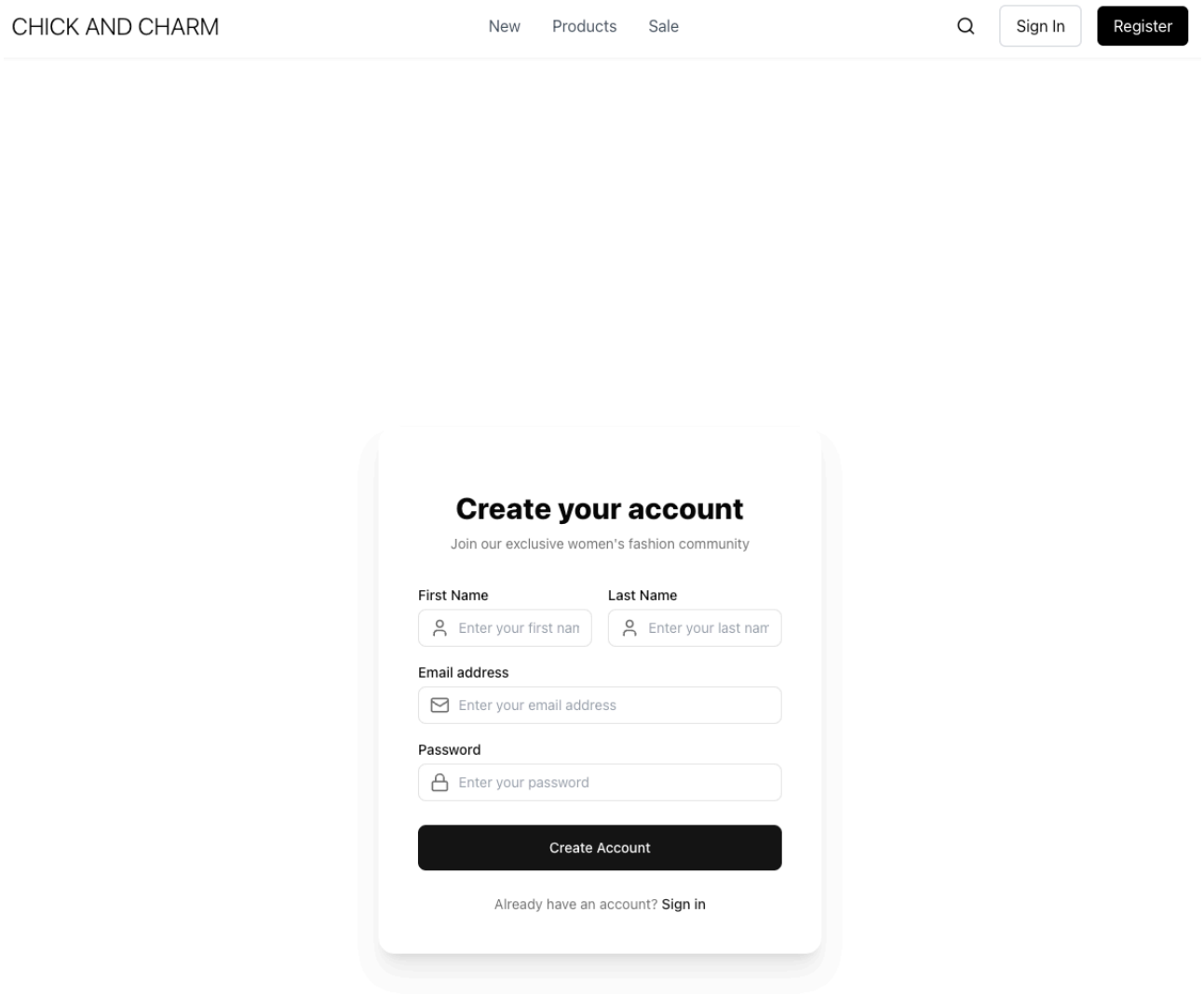


Рис. 3.4 Сторінка реєстрації

Окрім можливості реєстрації чи входу, користувач також може ознайомитися з асортиментом товарів без авторизації. На сторінці колекції відображається список доступних продуктів із фотографіями, назвами, описами, ціною та залишком на складі. Присутні зручні фільтри за категоріями та діапазоном цін. Приклад інтерфейсу цієї сторінки наведено на рисунку 3.5.

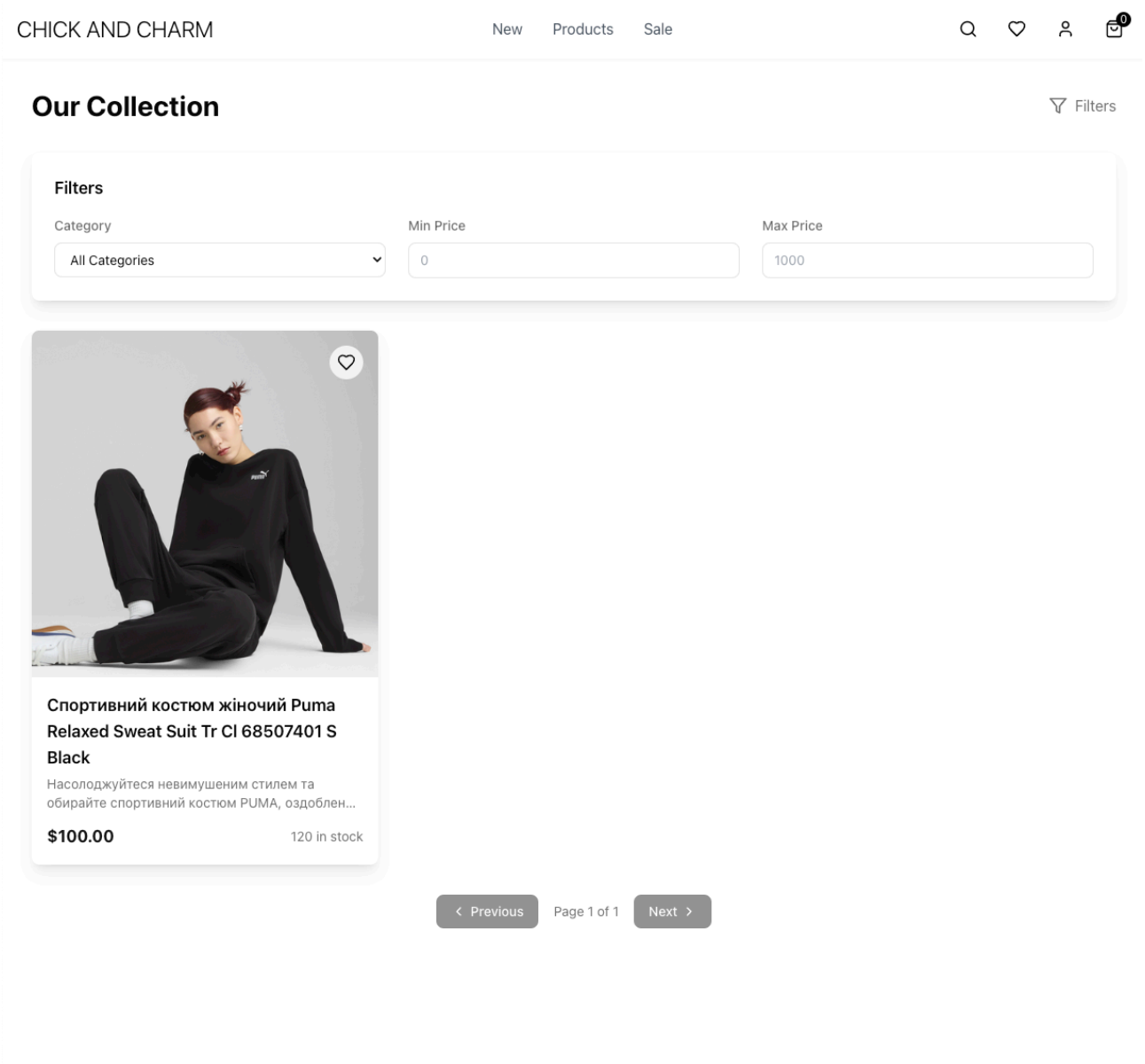


Рис. 3.5 Сторінка перегляду товарів.

Після авторизації користувач отримує доступ до персонального кабінету, де може переглядати та редагувати власні дані. У профілі доступна можливість оновлення контактної інформації, зміни пароля, а також керування збереженими адресами доставки. Крім того, користувач має змогу переглядати список обраних товарів та відстежувати історію замовлень. Інтерфейс особистого кабінету реалізовано у вигляді зручної панелі з вкладками, що забезпечує швидку навігацію між розділами. На рисунках 3.6 - 3.9 наведено приклад відображення сторінки профілю авторизованого користувача.

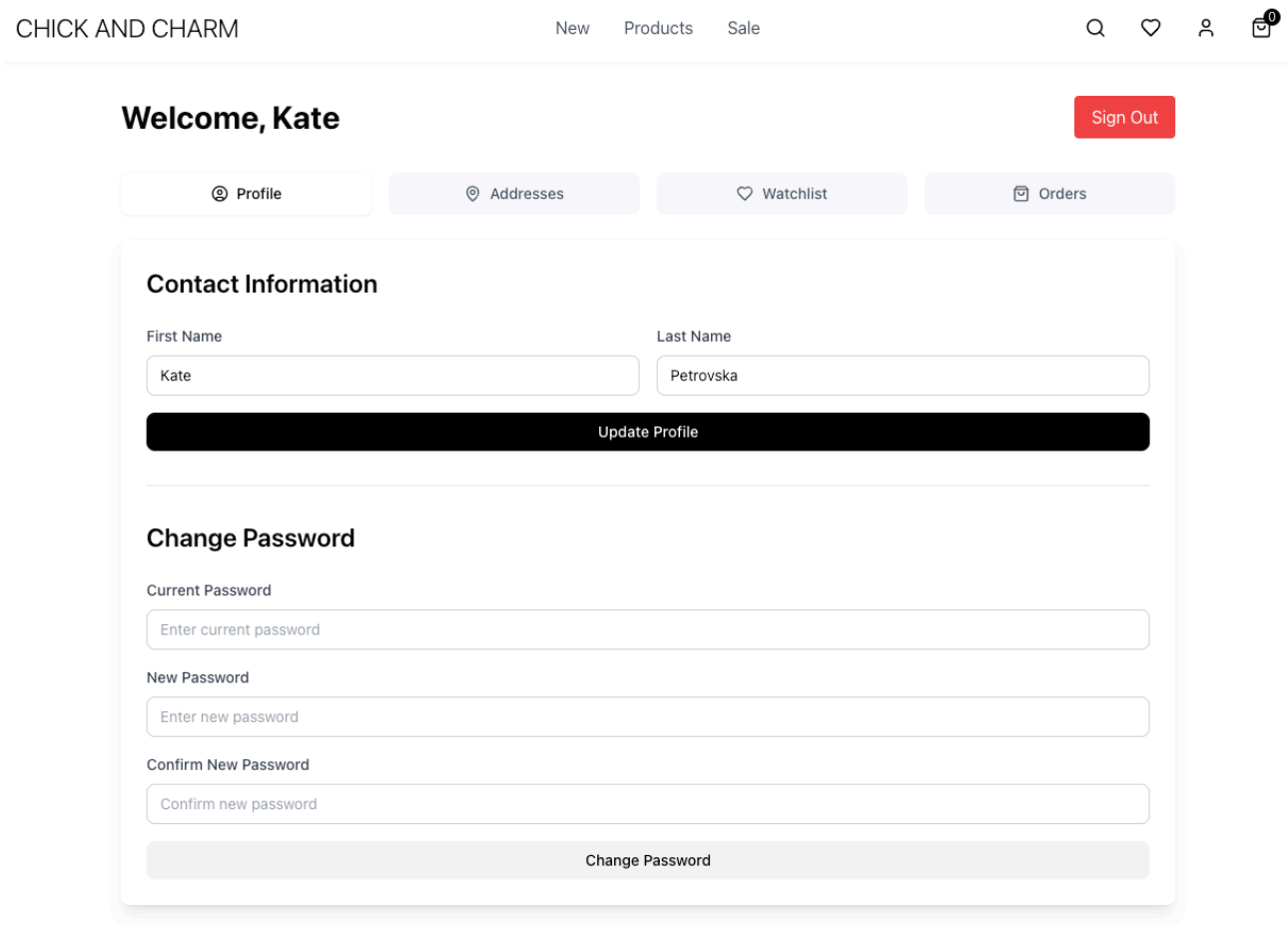


Рис. 3.6 Вкладка «Профіль» в особистому кабінеті користувача

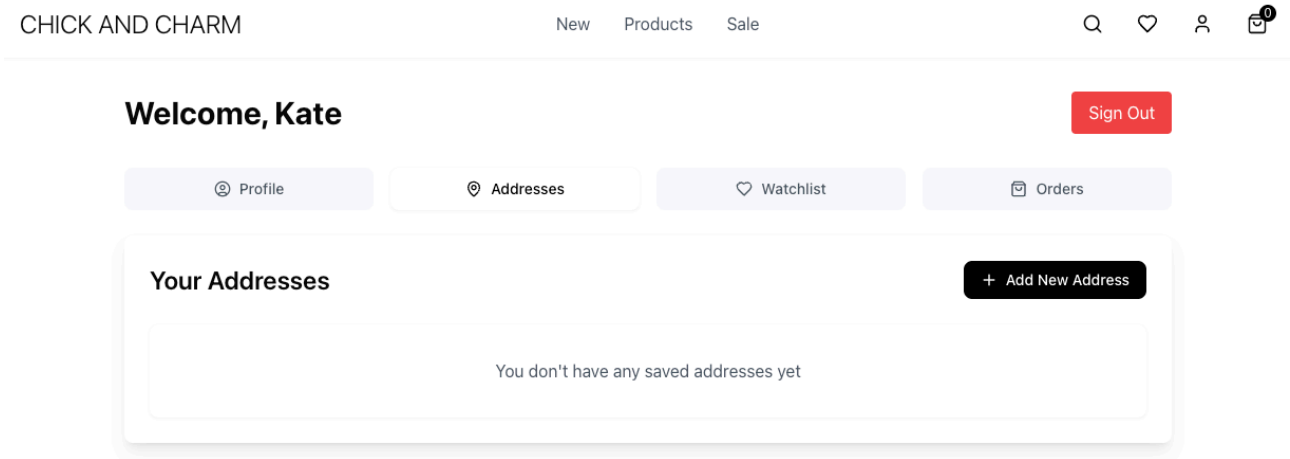


Рис. 3.7 Вкладка «Адреси» для керування місцями доставки

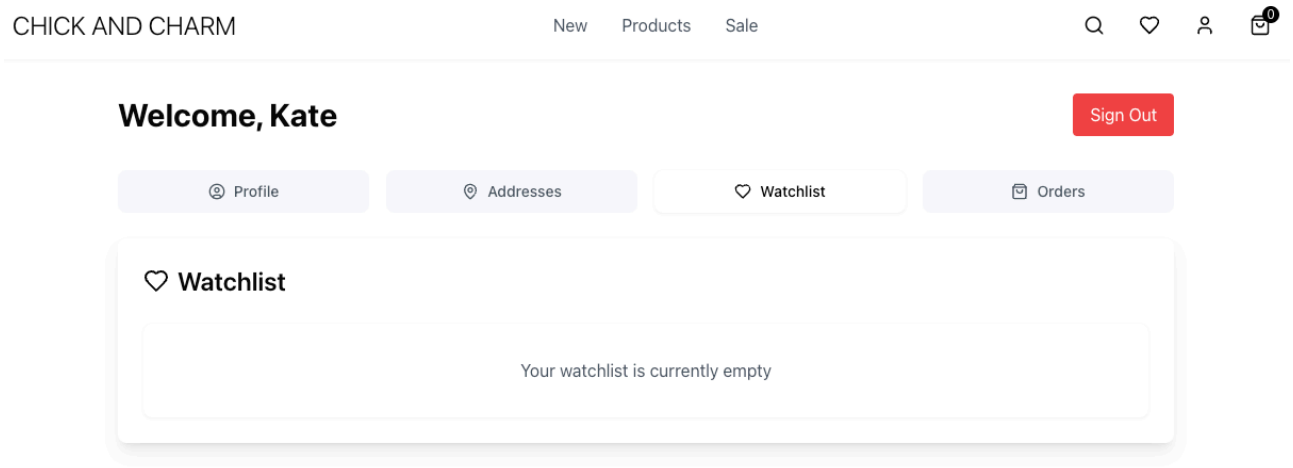


Рис. 3.8 Вкладка «Обране» з переліком збережених товарів

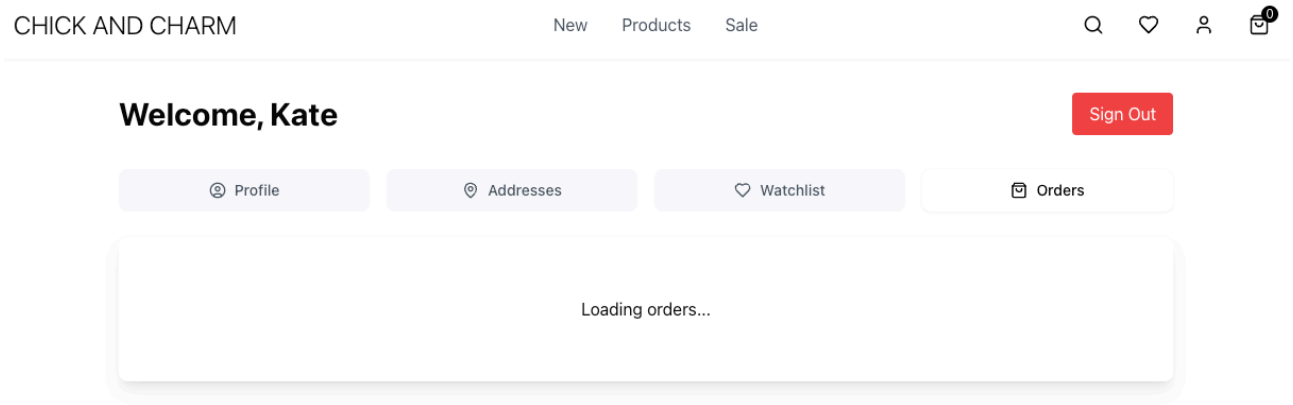


Рис. 3.9 Вкладка «Замовлення» з інформацією про здійснені покупки

Авторизований користувач має можливість додавати товари до списку уподобаного, натискаючи іконку у вигляді серця на картці товару (рис. 3.10). Це дозволяє зберегти вибрані позиції для подальшого перегляду або покупки без необхідності їх повторного пошуку. Така функціональність підвищує зручність користування сайтом та персоналізацію досвіду.

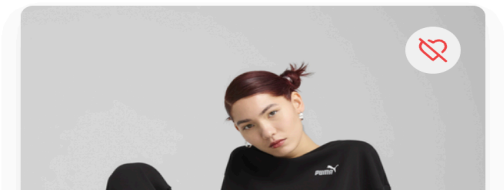


Рис. 3.10 Додавання товару до списку обраного авторизованим користувачем

При натисканні на картку товару користувач переходить на окрему сторінку з детальною інформацією про обраний продукт (рис. 3.11). Тут відображається велике зображення товару, його повна назва, опис, ціна, кількість на складі, а також доступні дії — додати до кошика або до списку уподобаного. Нижче розміщений блок для залишення коментарів, доступний авторизованим користувачам. Така структура дозволяє швидко ознайомитися з товаром та прийняти рішення щодо покупки.

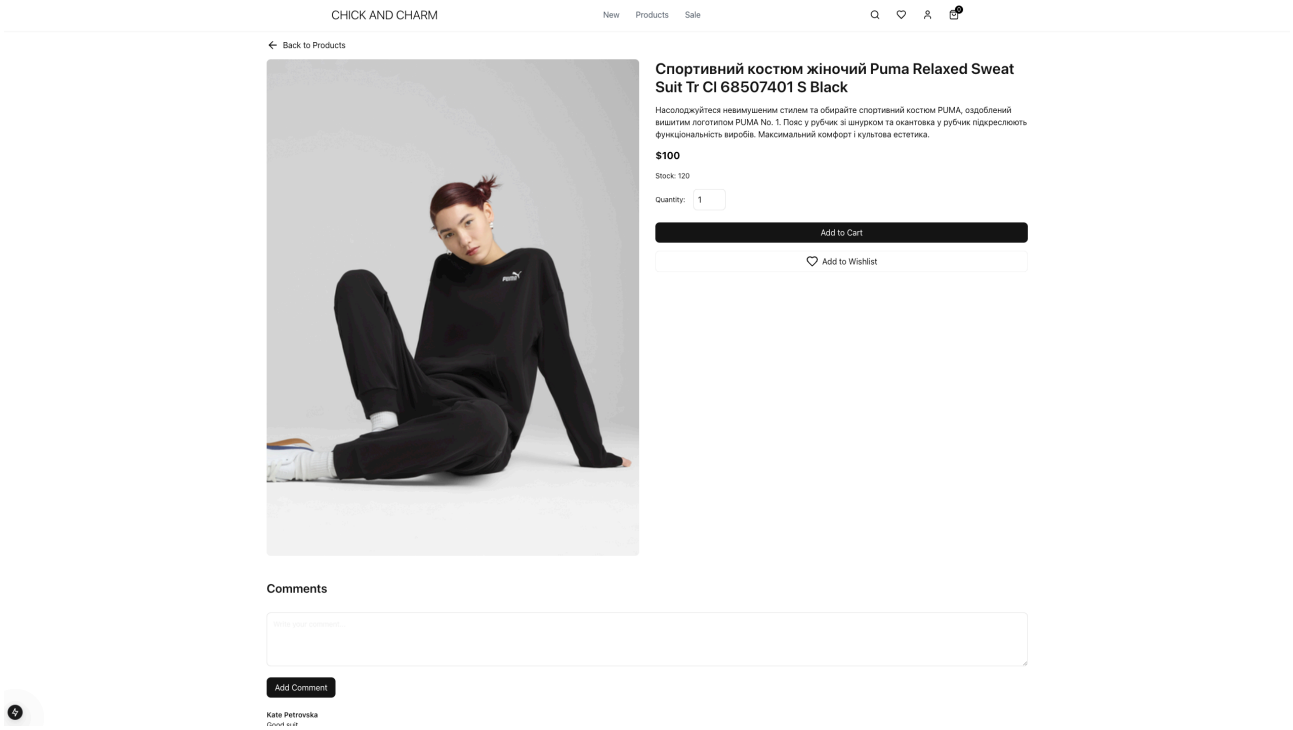


Рис. 3.11 Сторінка окремого товару з можливістю додавання до кошика або списку бажаного

Після натискання кнопки «Add to Cart» товар успішно додається до кошика. На екрані з’являється відповідне повідомлення, що підтверджує додавання (рис. 3.12). Після цього користувач може перейти до сторінки кошика, де відображаються всі обрані товари для оформлення замовлення. Це дозволяє зручно керувати покупками, змінювати кількість товарів або видаляти їх перед оплатою.

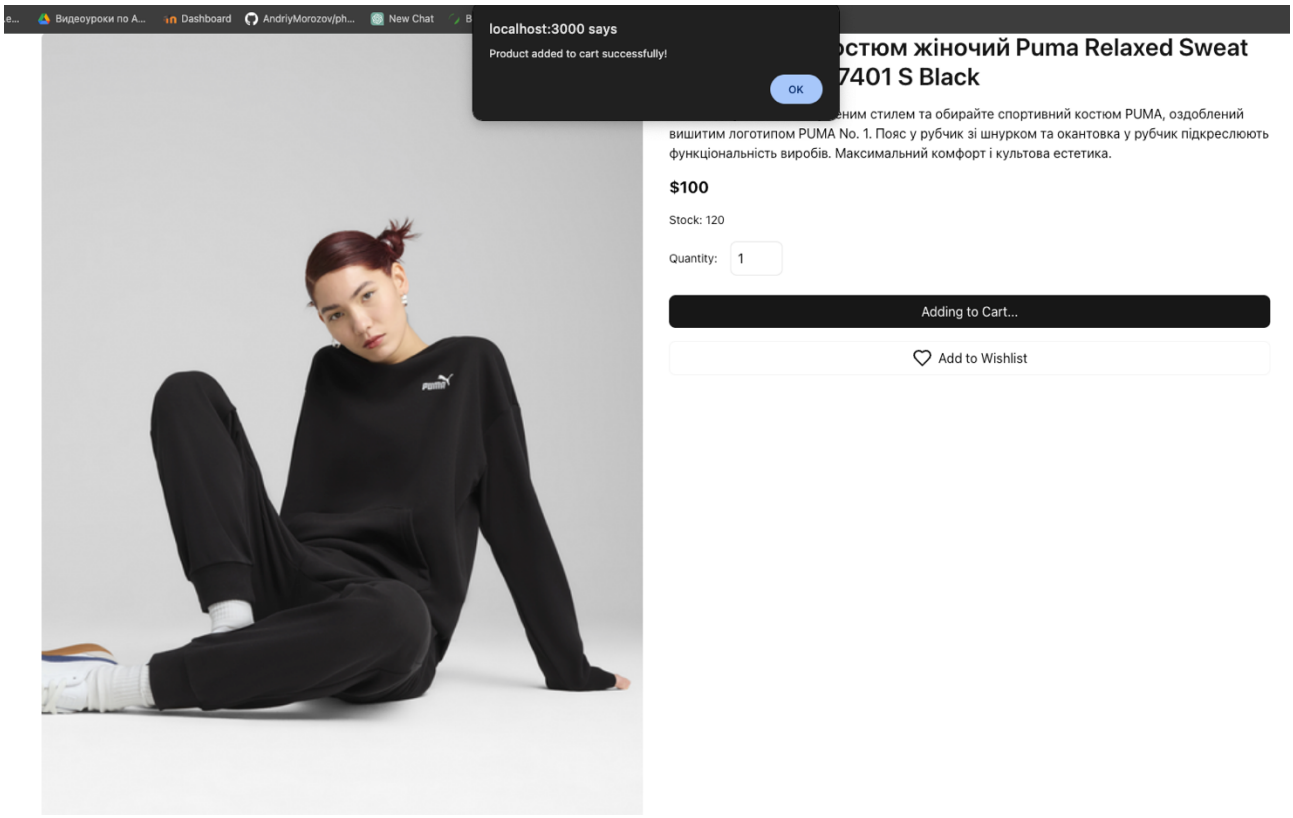


Рис. 3.12 Повідомлення про успішне додавання товару до кошика

Після додавання товару користувач може перейти до сторінки кошика, де відображається вся обрана продукція. Тут є можливість змінити кількість одиниць, видалити товар або повернутися до покупок. У правій частині екрана розміщено блок із підсумком замовлення, де вказано загальну суму, вартість доставки (яка розраховується при оформленні) та кнопка переходу до оформлення покупки (рис. 3.13).

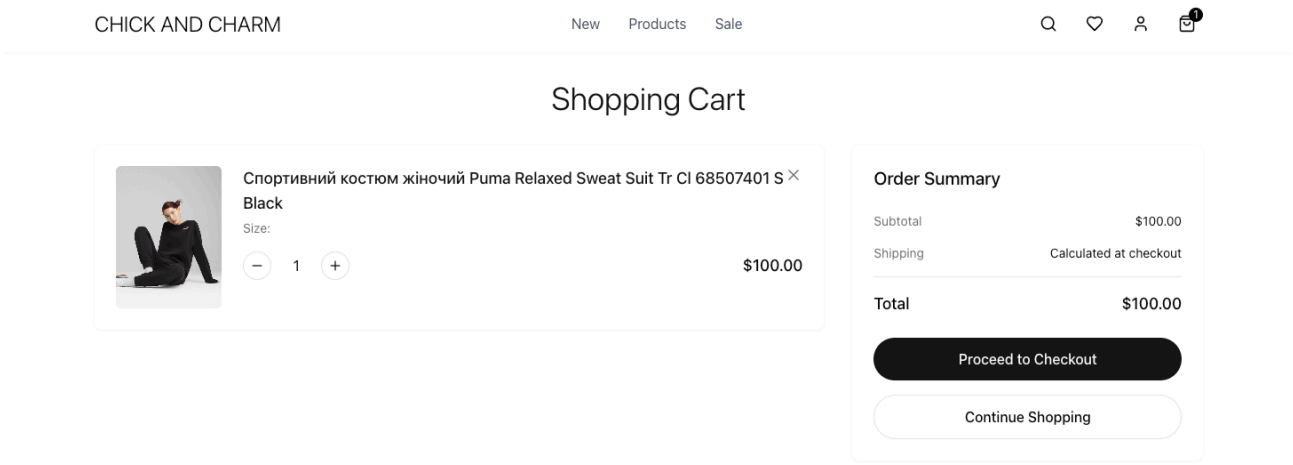


Рис. 3.13 Сторінка кошика з переліком доданих товарів і підсумком замовлення

На етапі оформлення замовлення користувач бачить стислий огляд вмісту кошика, зазначає адресу доставки та обирає зручний спосіб оплати. Доступні два варіанти: оплата при отриманні або оплата банківською карткою. У другому випадку активується форма введення реквізитів, що забезпечується через захищений платіжний шлюз. Після заповнення всіх полів користувач може підтвердити замовлення натисканням кнопки «Place Order» або «Pay», залежно від обраного способу оплати (рис. 3.14 – 3.15).

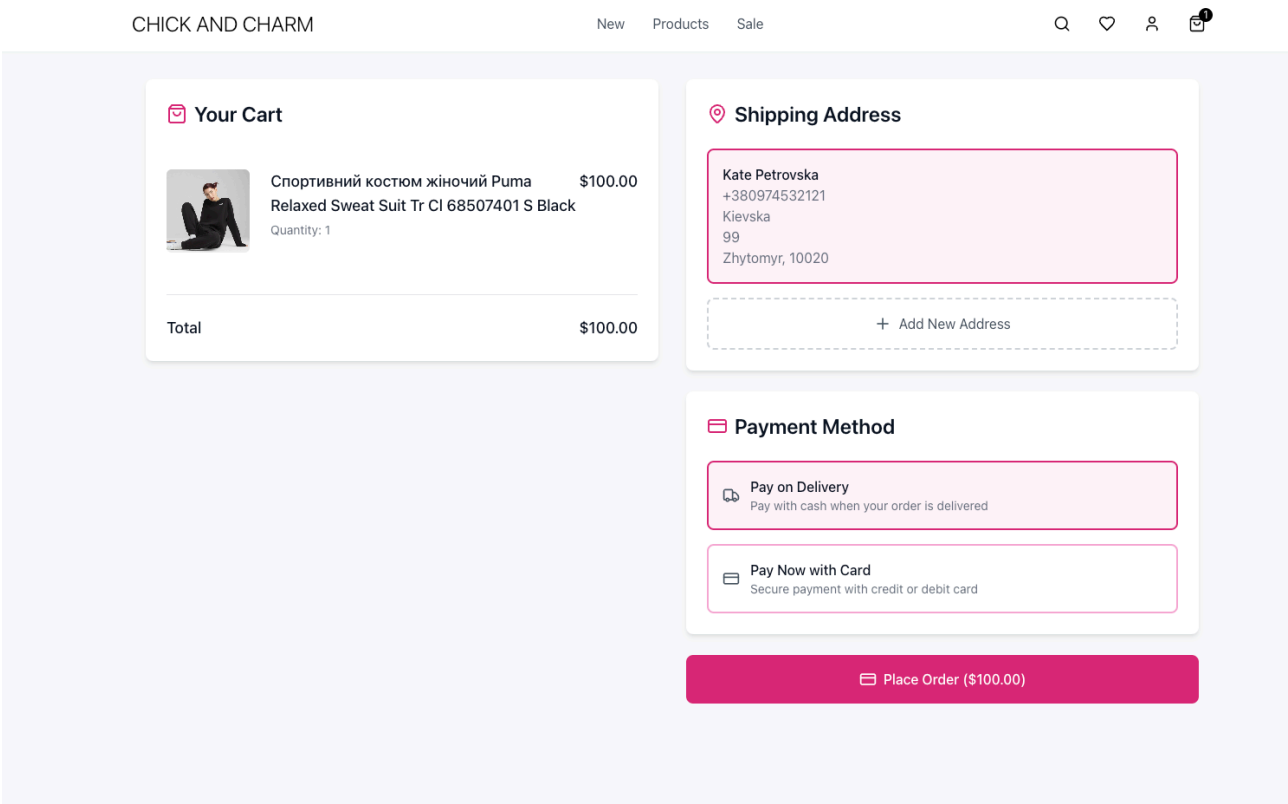


Рис. 3.14 Оформлення замовлення з вибором адреси доставки та способу оплати

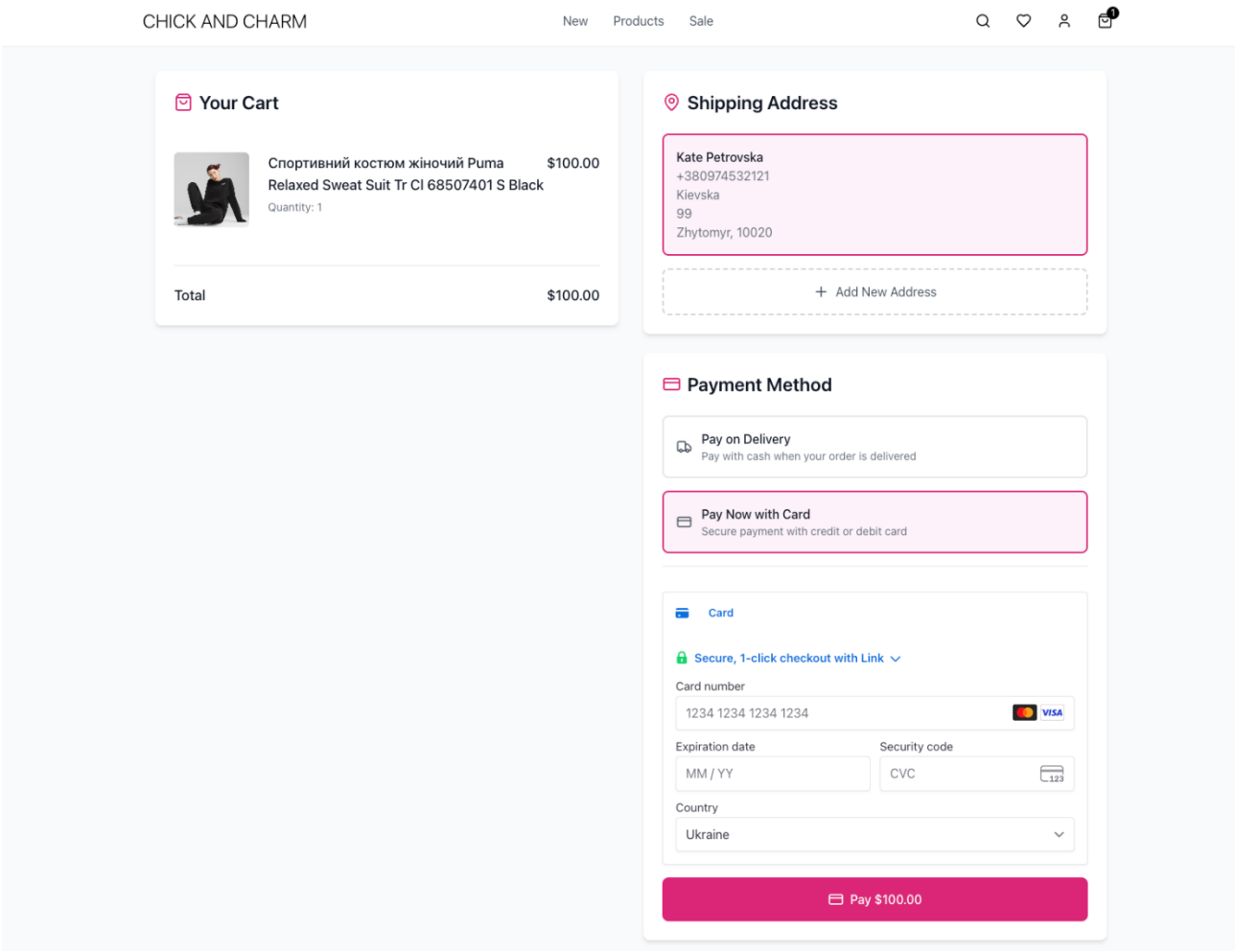


Рис. 3.15 Оплата банківською карткою через інтегрований платіжний шлюз

Після підтвердження замовлення користувач отримує повідомлення про його успішне створення. У разі оплати карткою додатково відображається повідомлення про завершення транзакції. Це означає, що покупка оброблена, і замовлення передано до виконання. Така інтерактивна взаємодія підвищує довіру до процесу покупки та створює комфортні умови для користувача (рис. 3.16 - 3.17).

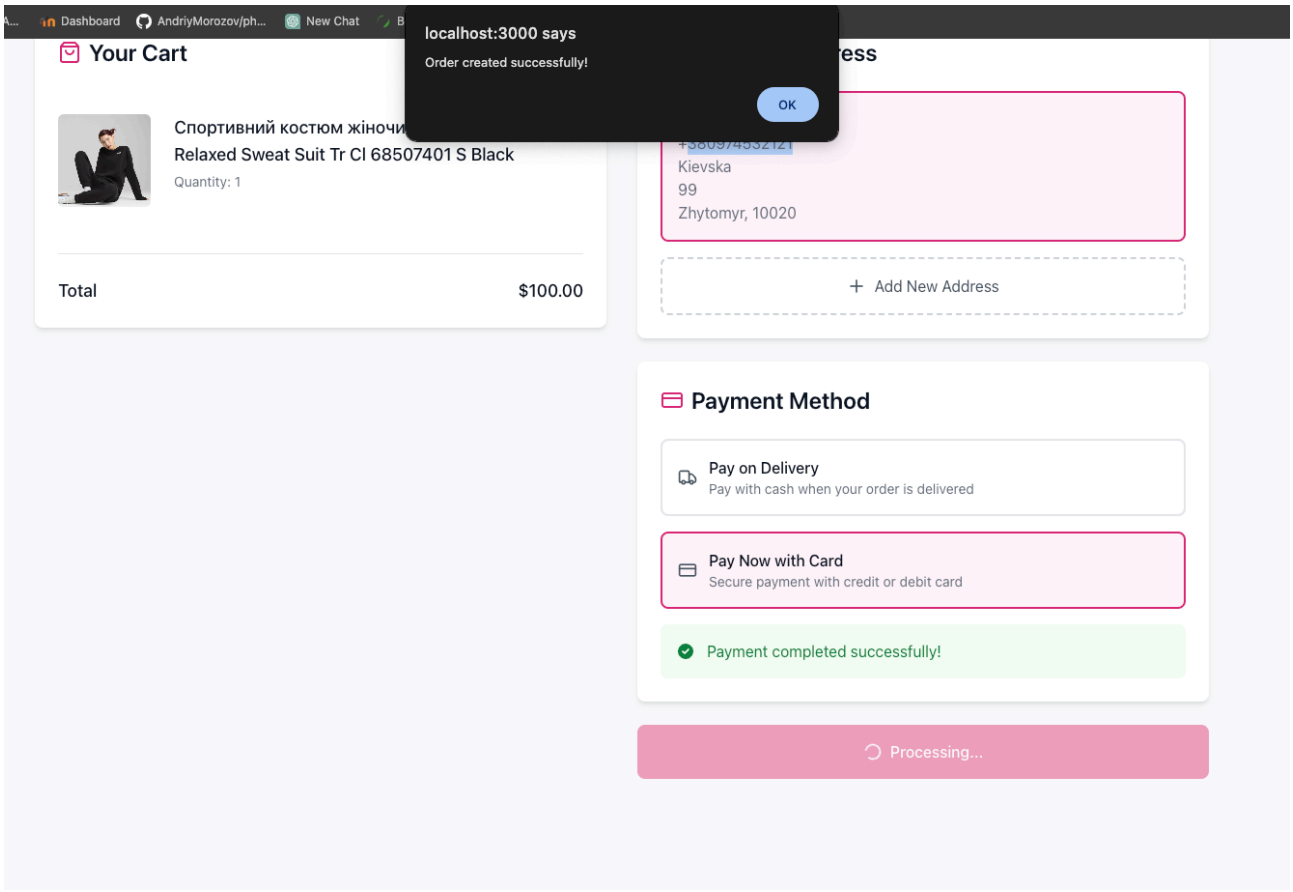


Рис. 3.16 Повідомлення про успішне створення замовлення після натискання кнопки оформлення

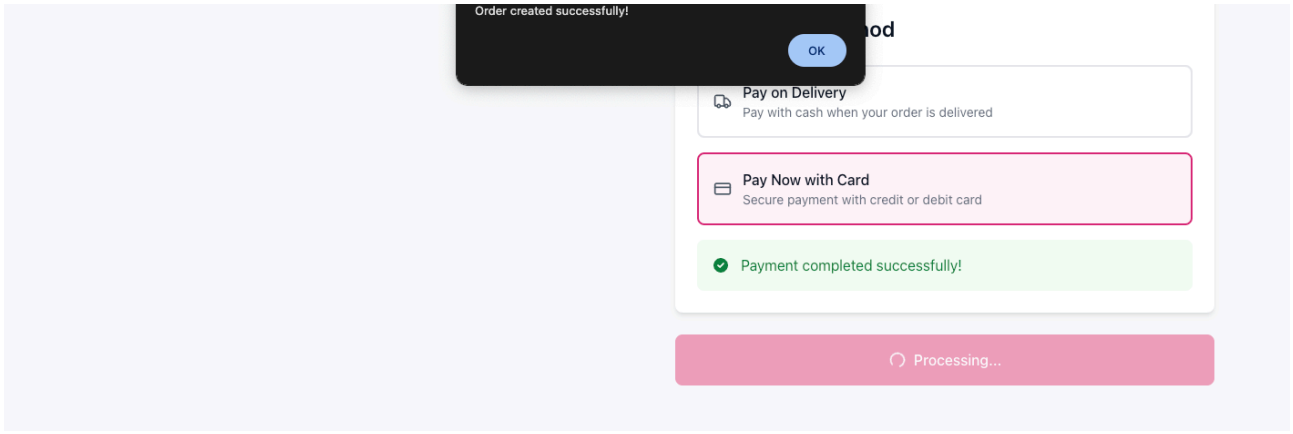


Рис. 3.17 Повідомлення про успішну оплату при виборі методу оплати банківською карткою

Після успішного оформлення замовлення користувач може переглядати інформацію про нього у розділі «Orders Management». Тут відображається унікальний номер замовлення, загальна сума, адреса доставки, перелік замовлених товарів та поточний статус (наприклад, Pending — в очікуванні

обробки) (рис. 3.18). Це дозволяє користувачу відстежувати стан свого замовлення без необхідності звертатися до служби підтримки.

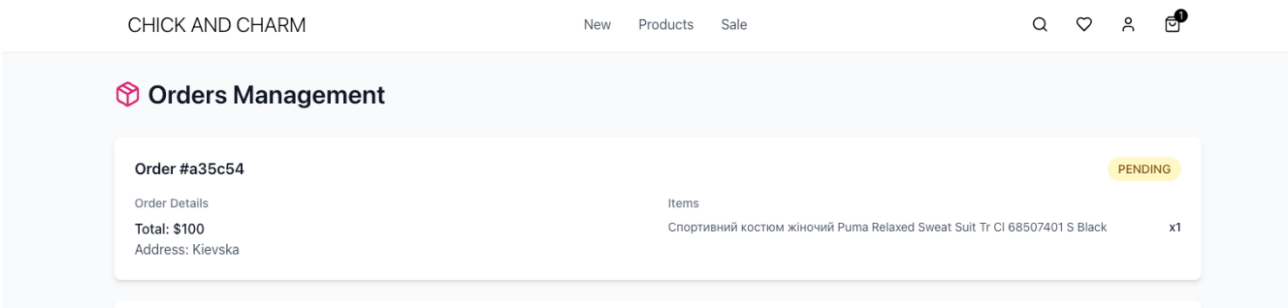


Рис. 3.18 Розділ управління замовленнями з відображенням статусу та деталей покупки

У разі, якщо користувач має роль адміністратора, йому відкривається доступ до адмін-панелі, де зібрано основні метрики та функції керування магазином (рис. 3.19). Тут можна переглядати кількість зареєстрованих користувачів, доданих товарів, здійснених замовлень і загальний дохід. У лівому боковому меню доступні вкладки для керування користувачами, товарами, замовленнями та візуальними компонентами магазину (слайдерами). Це забезпечує повноцінне адміністрування без виходу з системи.

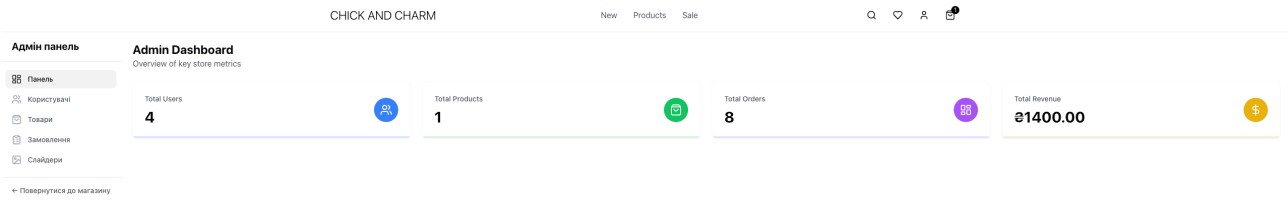


Рис. 3.19 Адмін-панель з основними показниками та можливостями керування інтернет-магазином

Користувач із правами супер-адміністратора має розширені можливості управління системою. Зокрема, він може змінювати ролі інших користувачів, додавати або призначати адміністраторів, а також керувати сповіщеннями (рис. 3.20). Однією з важливих функцій є генерація Telegram-посилання, що дозволяє підключити адміністратора до спеціального Telegram-бота, який надсилає автоматичні повідомлення про нові замовлення (рис. 3.21 – 3.22). Це значно спрощує оперативне реагування та моніторинг активності в магазині.

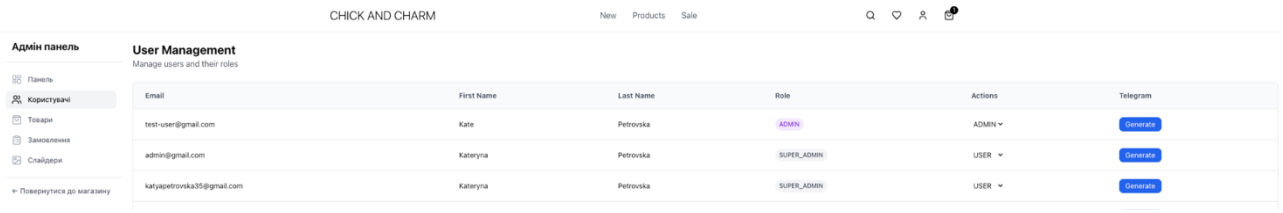


Рис. 3.20 Розділ керування користувачами з можливістю зміни ролей і генерації Telegram-посилання

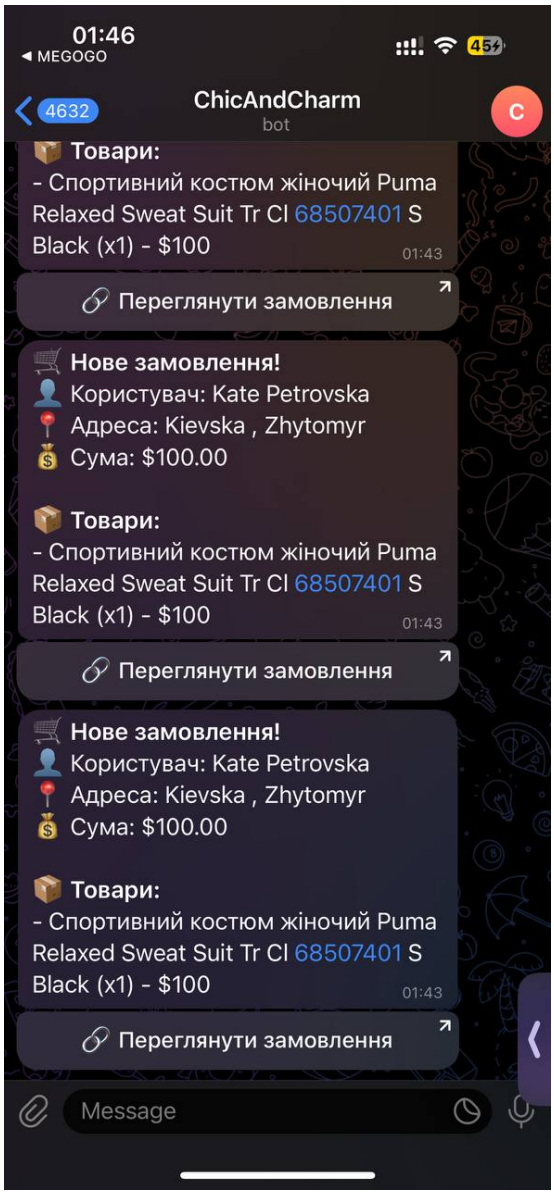


Рис. 3.21 Повідомлення про нове замовлення в Telegram-боті, що надходить адміністратору

Рис. 3.22 Згенероване посилання на Telegram-бота для підключення адміністратора

У розділі "Товари" адмін-панелі адміністратор має змогу додавати нові продукти, редагувати або видаляти наявні. В таблиці відображаються основні характеристики кожного товару: назва, ціна, наявність на складі, а також кнопки для редагування чи видалення (рис. 3.23). Це дозволяє оперативно оновлювати асортимент магазину, підтримувати актуальність цін і доступності товарів для користувачів.

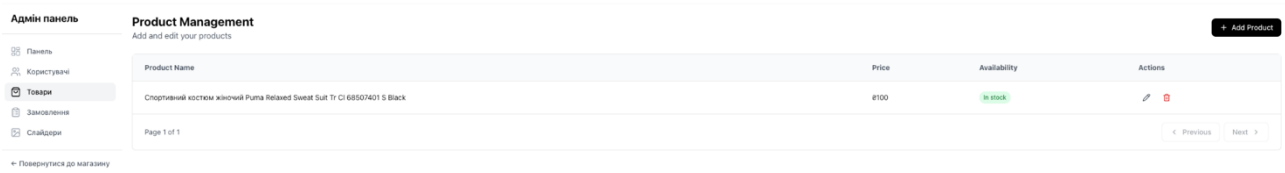


Рис. 3.23 Сторінка керування товарами в адмін-панелі інтернет-магазину

Під час створення нового товару адміністратор може заповнити назву, опис, ціну, кількість, категорію та завантажити фотографії. Для зручності додано можливість генерації опису товару на основі зображення за допомогою вбудованого модуля штучного інтелекту. Це дозволяє значно пришвидшити процес додавання нових позицій до каталогу та забезпечує якісні, інформативні описи без додаткових витрат часу. Досить лише завантажити фото та натиснути кнопку «Generate», після чого в текстовому полі автоматично з’являється згенерований опис (рис. 3.24 – 3.26).

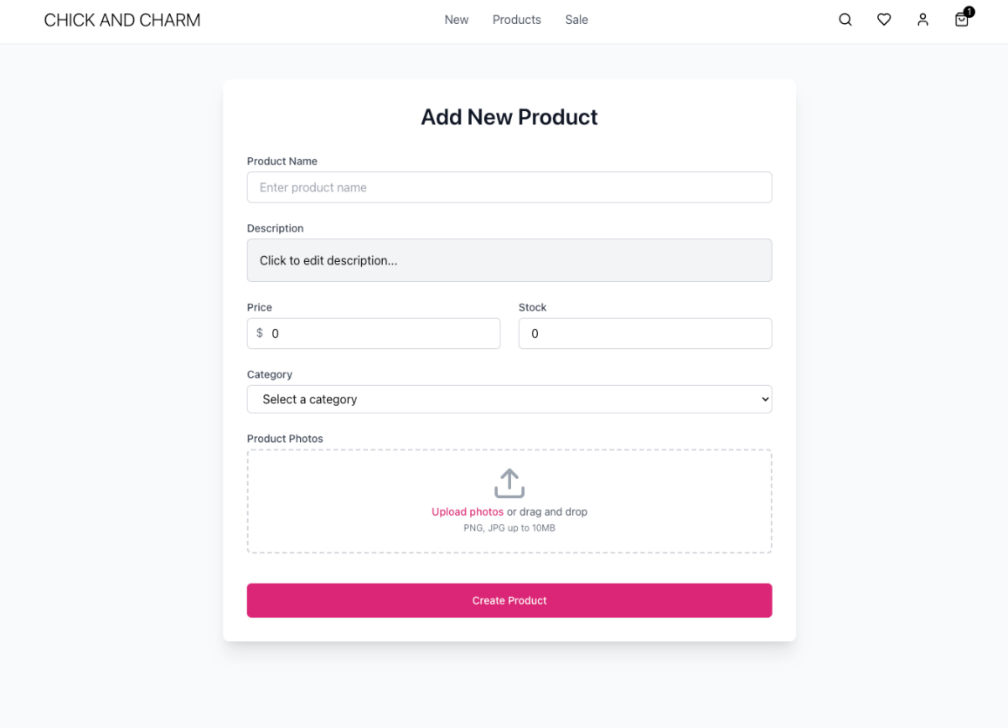


Рис. 3.24 Форма створення нового товару в адміністративній панелі

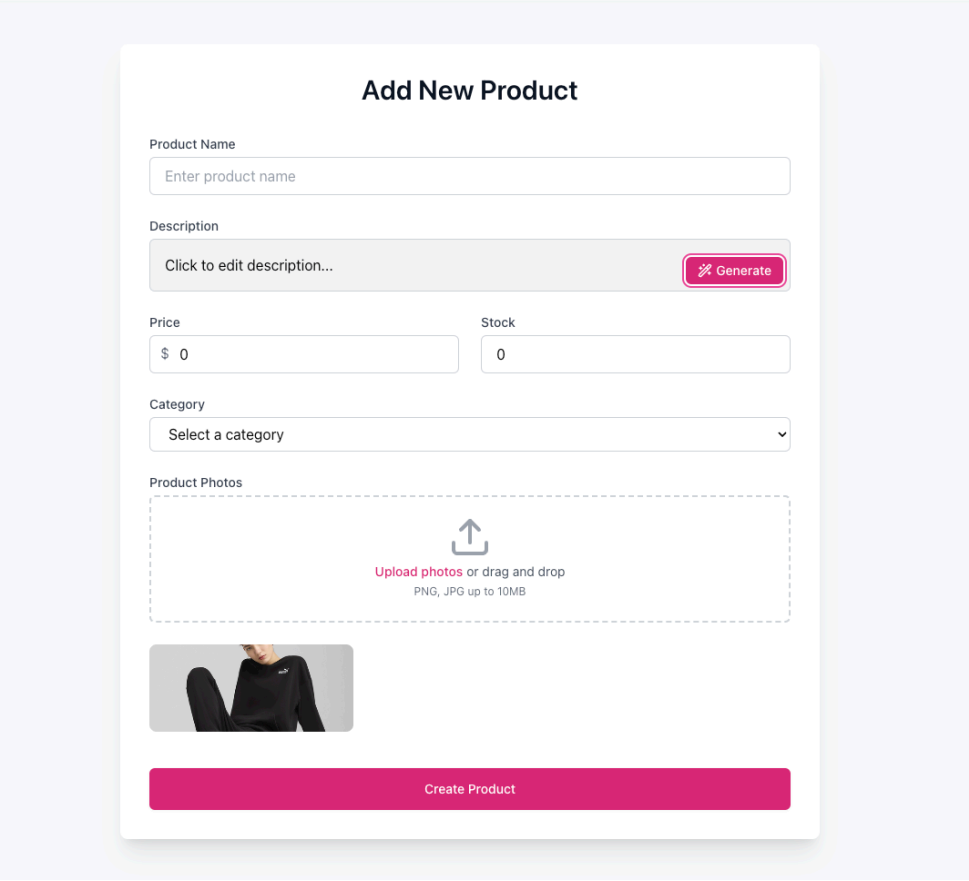


Рис. 3.25 Завантажене зображення товару перед генерацією опису

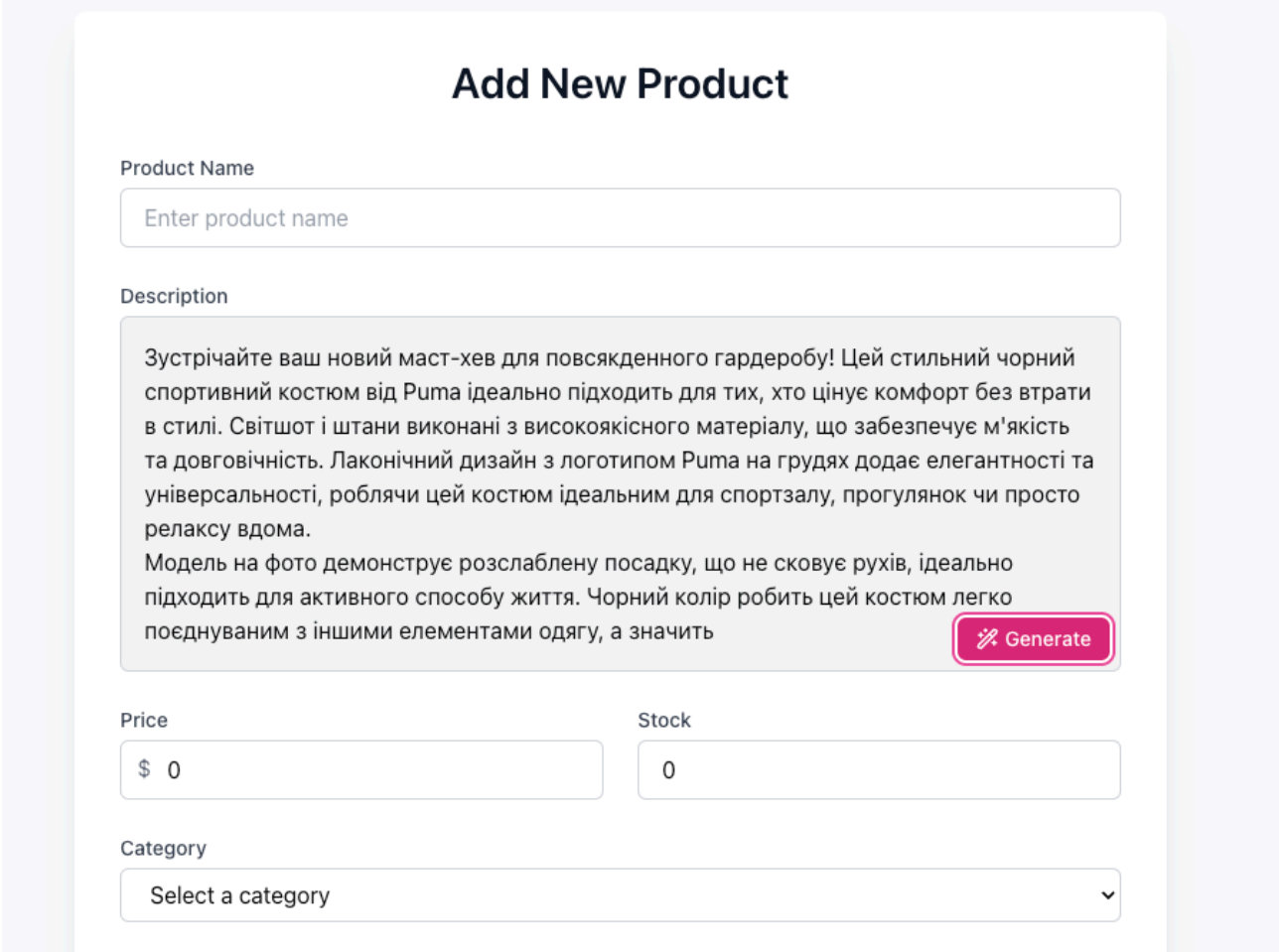


Рис. 3.26 Автоматично згенерований опис товару за допомогою ШІ

В адміністративній панелі також передбачений інструмент керування слайдерами, які відображаються на головній сторінці магазину для промоакцій або рекламних кампаній. Адміністратор може створити новий слайд, заповнивши назву, опис, посилання та зображення. Останнє можна як завантажити вручну, так і згенерувати автоматично за допомогою штучного інтелекту на основі текстового опису. Це дозволяє швидко створювати візуально привабливі банери без залучення сторонніх дизайнерів (рис. 3.27 – 3.29).

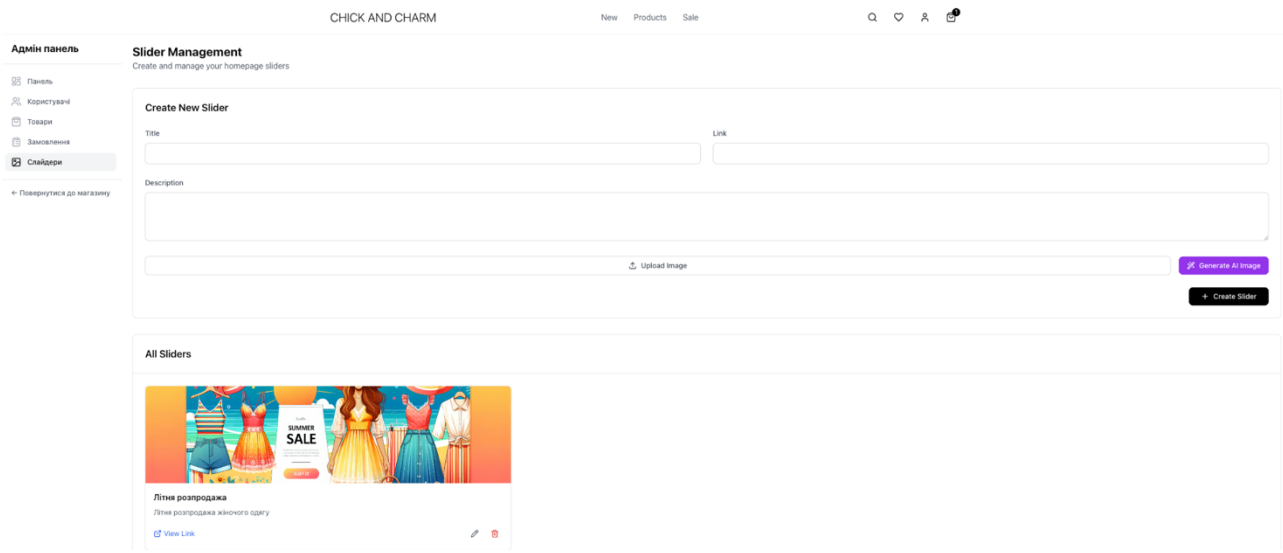


Рис. 3.27 Управління слайдерами: створення нового банера

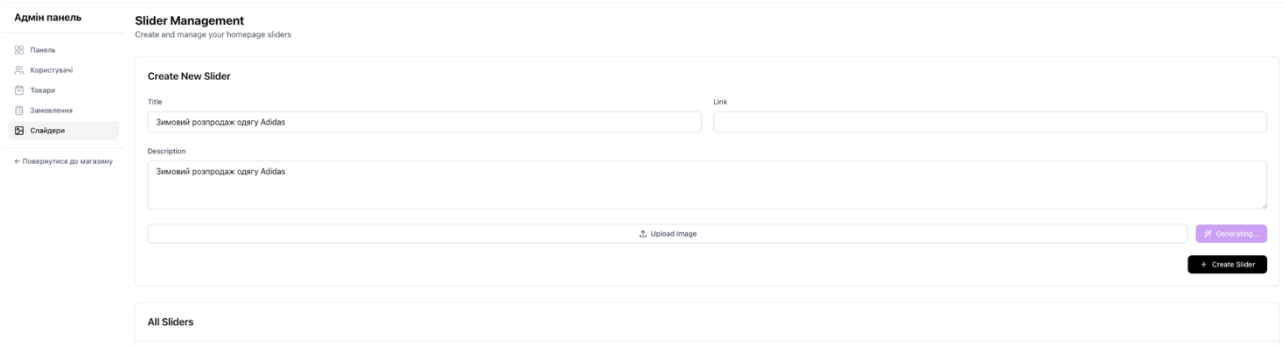


Рис. 3.28 Введення тексту для генерації банера

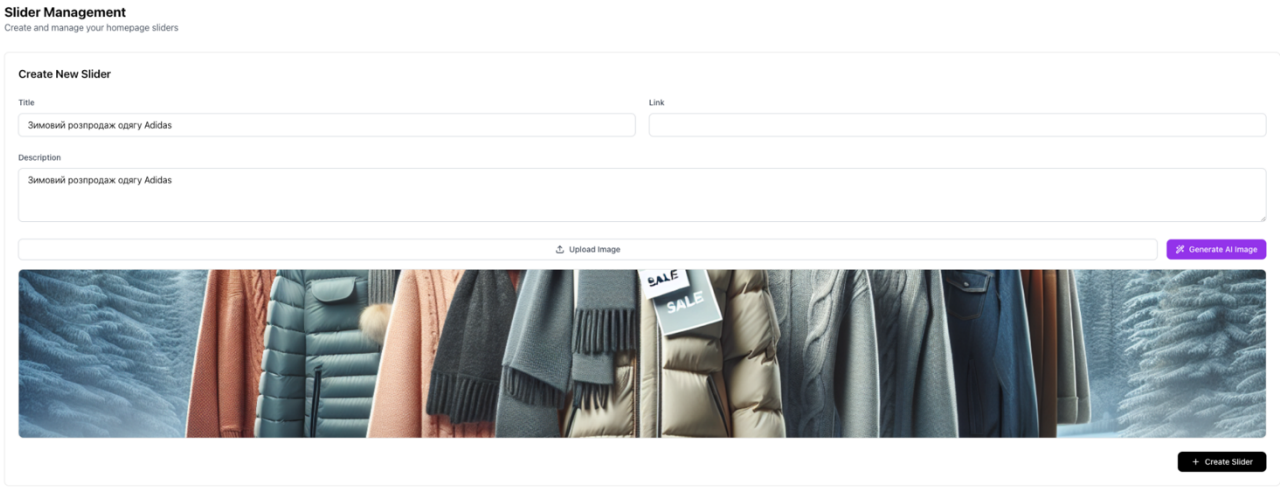


Рис. 3.29 Результат генерації слайдера за допомогою ШІ

3.3 Порядок встановлення та налаштування системи

Розробка будь-якої вебсистеми, зокрема інтернет-магазину, супроводжується виникненням певних похибок у логіці або реалізації окремих модулів. Це пов’язано з неможливістю на етапі проєктування передбачити усі можливі сценарії використання системи кінцевим користувачем. Тому процес тестування є невіддільним етапом у створенні якісного та надійного програмного забезпечення. Тестування дозволяє оцінити коректність виконання бізнес-логіки, перевірити відповідність фактичних результатів очікуваним, а також переконатися у стабільності роботи застосунку в різних умовах. Особливу увагу під час перевірки було приділено коректності обробки запитів, точності введення і виводу даних, функціональності кошика, реєстрації, оформлення замовлення та управління адміністративною панеллю. Основною метою тестування було виявлення дефектів, які можуть негативно вплинути на досвід користувача або функціонування системи в цілому, а також забезпечення можливості їх оперативного усунення до моменту запуску платформи у виробниче середовище. Для цього було складено набір тест-кейсів — структурованих сценаріїв, які описують умови, дії користувача та очікувану поведінку системи. Ці тест-кейси допомогли перевірити ключові функціональні модулі та виявити потенційні помилки. Приклад тестових сценаріїв наведено у таблиці 3.1.

Таблиця 3.1

Перелік тест-кейсів для перевірки працездатності інтернет-магазину

Test Case Description	Test Steps	Test Data	Expected Results	Actual Results	Pass/ Fail
Реєстрація нового користувача	Заповнити форму реєстрації та натиснути "Create Account".	Ім’я: Kate, Email: kate@example.com, Password: 12345678	Користувача успішно зареєстровано, відкривається профіль	Користувача успішно зареєстровано, редірект на профіль	Pass

Продовження табл. 3.1

Авторизація існуючого користувача	Відкрити сторінку входу, ввести email та пароль, натиснути "Sign In"	Email: kate@example.com, Password: 12345678	Вхід успішний, користувач бачить головну сторінку з товарами	Вхід успішний, користувач потрапляє на головну	Pass
Перегляд товарів без авторизації	Відкрити головну сторінку	—	Користувач бачить список товарів і фільтри	Товари відображаються, фільтрація працює	Pass
Перехід на сторінку конкретного товару	Натиснути на картку товару	—	Відкривається сторінка з описом товару, ціною, кнопкою "Add to Cart"	Сторінка відкривається, вся інформація доступна	Pass
Додавання товару до кошика	Обрати товар, натиснути кнопку "Add to Cart"	Товар: Puma костюм	Товар додається до кошика, з'являється повідомлення про успішне додавання	Товар додано до кошика, повідомлення з'явилося	Pass

Закінчення таблиці 3.1

Перегляд вмісту кошика	Відкрити сторінку кошика	—	Відображається список товарів у кошику з цінами та підсумком	У кошику відображається доданий товар	Pass
Оформлення замовлення (оплата готівкою)	Перейти в кошик, ввести адресу доставки, обрати метод оплати "Pay on Delivery", натиснути "Place Order"	Адреса: вул. Київська 99, Житомир	Замовлення створено, статус "Pending", повідомлення про успішне оформлення	Замовлення з’являється в адмін-панелі	Pass
Оформлення замовлення (оплата карткою)	Заповнити дані адреси та картки, обрати "Pay with Card", підтвердити платіж	Дані тестової картки Stripe	Оплата проходить успішно, з’являється повідомлення "Payment successful"	Оплата пройшла, замовлення створено	Pass
Додавання нового продукту (адмін)	Авторизуватись як адміністратор, перейти до розділу "Products",	Назва, опис, фото, ціна, категорія	Продукт з’являється у списку товарів	Продукт додано успішно	Pass

	натиснути "Add Product", заповнити форму, натиснути "Create"				
Перегляд замовлень в адмін-панелі	Відкрити розділ "Замовлення" у панелі адміністратора	—	Виводиться таблиця всіх замовлень зі статусами	Усі замовлення відображаються	Pass
Отримання повідомлення в Telegram-боті	Створити нове замовлення, перевірити повідомлення у Telegram	Telegram user з'єднаний	Адміну приходить повідомлення з деталями замовлення	Повідомлення отримано, містить всі дані	Pass
Генерація промослайду через AI	Увійти як адміністратор, перейти до "Слайдери", ввести опис слайду, натиснути "Generate AI Image", зберегти результат	Опис: Зимовий розпродаж Adidas	Генерується зображення, додається до слайдера на головній сторінці	Слайдер з'явився у списку, AI-зображення згенеровано	Pass

Висновки до третього розділу

У третьому розділі було представлено діаграму розгортання інтернет-магазину жіночого одягу та інтерфейс користувача з прикладами взаємодії. Описано функціональні можливості для звичайних користувачів, адміністраторів і супер-адміністраторів. Проведене тестування підтвердило стабільну роботу всіх основних компонентів системи.

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи було спроєктовано та реалізовано сучасний інтернет-магазин жіночого одягу, що відповідає актуальним вимогам ринку електронної комерції. На основі аналізу предметної області, вивчення потреб цільової аудиторії та порівняння з існуючими рішеннями було визначено ключові функціональні та технічні характеристики системи.

У ході роботи створено повноцінну структуру бази даних, реалізовано інтерфейс з адаптивним дизайном, розроблено механізми реєстрації, авторизації, управління кошиком, оформлення замовлень, а також панель адміністрування для управління товарами, категоріями, слайдерами та користувачами. Забезпечено підтримку пошуку, фільтрації, коментарів, списку бажаного та безпечної обробки платіжних операцій.

Платформа успішно пройшла тестування, під час якого перевірено її стабільність, функціональність і зручність для користувачів. Усі виявлені недоліки були усунуті, що дозволило досягти високої якості реалізації.

Розроблений інтернет-магазин демонструє практичну реалізацію сучасного вебзастосунку для роздрібної торгівлі одягом, сприяє цифровізації торгівлі, розширенню бізнес-можливостей та підвищенню рівня клієнтського сервісу.

Отриманий результат може бути використаний як готове рішення або основа для подальшого вдосконалення з урахуванням нових вимог ринку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ASOS — офіційний сайт міжнародного інтернет-магазину одягу та аксесуарів [Електронний ресурс]. – Режим доступу: https://www.asos.com
2. ZARA — офіційний сайт бренду одягу ZARA [Електронний ресурс]. – Режим доступу: https://www.zara.com
3. KASTA — український інтернет-магазин модного одягу та взуття [Електронний ресурс]. – Режим доступу: https://kasta.ua
4. NextJS SSR [Електронний ресурс] - Режим доступу до ресурсу: https://nextjs.org/docs/pages/building-your-application/rendering/server-side-rendering
5. NextJS [Електронний ресурс] - Режим доступу до ресурсу: https://nextjs.org
6. NextAuth.js [Електронний ресурс] - Режим доступу до ресурсу: https://next-auth.js.org/
7. Prisma [Електронний ресурс]. – Режим доступу: https://www.prisma.io
8. Tanstack query [Електронний ресурс] - Режим доступу до ресурсу: https://tanstack.com/query/latest/
9. node-telegram-bot-api [Електронний ресурс]. – Режим доступу: https://github.com/yagop/node-telegram-bot-api
10. Zod [Електронний ресурс]. – Режим доступу: https://zod.dev
11. NextJS SSG [Електронний ресурс] - Режим доступу до ресурсу: https://nextjs.org/docs/pages/building-your-application/rendering/static-site-generation
12. TypeScript [Електронний ресурс] - Режим доступу до ресурсу: https://www.typescriptlang.org/
13. NextJS CSR [Електронний ресурс] - Режим доступу до ресурсу: https://nextjs.org/docs/pages/building-your-application/rendering/client-side-rendering
14. PostgreSQL- [Електронний ресурс] - Режим доступу до ресурсу: https://www.postgresql.org/

15. MySql [Електронний ресурс] - Режим доступу до ресурсу: https://www.mysql.com/

16. MongoDB [Електронний ресурс] - Режим доступу до ресурсу: https://www.mongodb.com/

17. Strapi's documentation [Електронний ресурс] – Режимдоступудоресурсу: https://docs.strapi.io/

18. ReactJS [Електронний ресурс] - Режим доступу до ресурсу: https://react.dev/reference/react

19. Tailwind CSS [Електронний ресурс] - Режим доступу до ресурсу: https://tailwindcss.com/docs

20. NextJS Serever actions - [Електронний ресурс] - Режим доступу до ресурсу: https://nextjs.org/docs/app/building-your-application/data-fetching/server-actions-and-mutations

21. Tanstack query_[Електронний ресурс] - Режим доступу до ресурсу: https://tanstack.com/query/latest/

22. Vercel AI SDK [Електронний ресурс] - Режим доступу до ресурсу: https://ai-sdk.dev/docs/introduction

23. Joana Almeida. Node js vs. PHP: An In-Depth Comparison Guide for Web Development – DistantJob [Електронний ресурс] / Joana Almeida – 9 червня 2022 – Режим доступу до ресурсу: https://distantjob.com/blog/nodejs-vsphp/.

24. What is middleware? [Електронний ресурс] – Режим доступу доресурсу:https://www.redhat.com/en/topics/middleware/what-is-middleware

25. Cloudinary for Node.js [Електронний ресурс] - Режим доступу до ресурсу: https://cloudinary.com/documentation/node_integration

26. Building your own hooks [Електронний ресурс] - Режим доступу до ресурсу: https://legacy.reactjs.org/docs/hooks-custom.html

27. Документація JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript

28. Write a simple Telegram Bot in Next.js [Електронний ресурс]–Режим доступу до ресурсу: https://blog.devgenius.io/write-a-simpletelegram-bot-in-next-js-b49379e77163

29. Telegram Bot API Documentation [Електронний ресурс]– Режим доступу до ресурсу: https://core.telegram.org/bots/api

30. Prisma Documentation - Офіційна документація Prisma [Електронний ресурс]. – Режим доступу до ресурсу: https://www.prisma.io/docs/

31. JOURNAL I. ReactJS: A Comprehensive Analysis of its features, Performance, and Suitability for Modern Web Development. INTERANTIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT. 2023. Vol. 07, no. 09. P. 1. URL: https://doi.org/10.55041/ijsrem25667.

32. Clean Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/codenx/clean-architecture-4c23dfa4c698

33. Роберт М. Чиста архітектура: Мистецтво створення програмного забезпечення. Роберт М. [Видання друге] – Харків: Фабула, 2023 – 368 с

34. Learning React: Modern Patterns for Developing React Apps, 2nd edition / Alex Banks, Eve Porcello, 2020 – 307 p.

35. React Explained: Your Step-by-Step Guide to React, 1st edition / Zac Gordon, 2019 – 366 p.

36. NextAuth.js Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://next-auth.js.org/getting-started/introduction

37. Many-to-many relations in Prisma [Електронний ресурс] – Режим доступу до ресурсу: https://www.prisma.io/docs/orm/prisma-schema/datamodel/relations/many-to-many-relations

38. TypeScript for JavaScript - [Електронний ресурс] - Режим доступу до ресурсу: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

39. Трофименко О. Г. Веб-технології та веб-дизайн : навч. посібник / О. Г. Трофименко, О. Б. Козін, О. В. Задерейко, О. Є. Плачінда. – Одеса : Фенікс, 2019. – 284 с.

40. Тест-кейс – один із ключових інструментів тестувальників ПЗ. [Електронний ресурс] – Режим доступу до ресурсу: https://foxminded.ua/test-keis/.

41. Документація HTML5 [Електронний ресурс] – Режим доступу до ресурсу: https://webplatform.github.io/docs/html/html5/

ДОДАТКИ

Лістинг authOption.ts:

```
import { login } from "@/actions/auth";
import { NextAuthOptions } from "next-auth";
import CredentialsProvider from "next-auth/providers/credentials";

export const authOptions: NextAuthOptions = {
  providers: [
    CredentialsProvider({
      name: "Credentials",
      credentials: {
        email: { label: "Email", type: "email" },
        password: { label: "Password", type: "password" },
      },
      async authorize(credentials) {
        if (!credentials) {
          return null;
        }

        const response = await login(credentials);

        if (response.success && response.data) {
          return {
            id: response.data.id,
            firstName: response.data.firstName,
            lastName: response.data.lastName,
            email: response.data.email,
            role: response.data.role,
          };
        }

        return null;
      },
    }),
  ],
  session: {
    strategy: "jwt",
  },
  secret: process.env.NEXTAUTH_SECRET,
  pages: {
    signIn: "/login",
  },
  callbacks: {
```

```
    async jwt({ token, user }) {
      if (user) {
        token.id = user.id;
        token.firstName = user.firstName;
        token.lastName = user.lastName;
        token.email = user.email;
        token.role = user.role;
      }
      return token;
    },
    async session({ session, token }) {
      if (token) {
        session.user.id = token.id as string;
        session.user.firstName = token.firstName as string;
        session.user.lastName = token.lastName as string;
        session.user.email = token.email as string;
        session.user.role = token.role as string;
      }
      return session;
    },
  },
};
```

Лістинг schema.prisma

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = "postgresql://postgres.toxacborybvpfiianipi:EQAAvn29Lqd*gHa@aws-0-eu-central-
1.pooler.supabase.com:5432/postgres"
}

model User {
  id         String      @id @default(uuid())
  email      String      @unique
  password   String
  firstName  String?
  lastName   String?
  role       Role        @default(USER)
  telegramId String? // only for admins
  addresses  Address[]
  cartItems  CartItem[]
```

```
  orders     Order[]
  createdAt  DateTime    @default(now())
  updatedAt  DateTime    @updatedAt
  Comment    Comment[]
  Watchlist  Watchlist[]
}

model Address {
  id           String   @id @default(uuid())
  user         User     @relation(fields: [userId], references: [id])
  userId       String
  recipientName String
  phoneNumber   String
  addressLine1  String
  addressLine2  String?
  city          String
  postalCode    String
  country       String   @default("Україна")
  isDefault     Boolean  @default(false)
  createdAt     DateTime @default(now())
  updatedAt     DateTime @updatedAt
  Order         Order[]
}

model Category {
  id          String    @id @default(uuid())
  name        String    @unique
  description String?
  imageUrl    String?
  products    Product[]
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt
}

model Product {
  id          String        @id @default(uuid())
  name        String
  description  String?
  price       Float
  stock       Int           @default(0)
  category    Category      @relation(fields: [categoryId], references: [id], onDelete: Cascade)
  categoryId   String
  images      ProductImage[]
  cartItems    CartItem[]
```

```
  orderDetails OrderDetail[]
  createdAt    DateTime       @default(now())
  updatedAt    DateTime       @updatedAt
  Comment      Comment[]
  Watchlist    Watchlist[]
}

model ProductImage {
  id        String   @id @default(uuid())
  product   Product  @relation(fields: [productId], references: [id])
  productId String
  imageUrl  String
  createdAt DateTime @default(now())
}

model CartItem {
  id        String   @id @default(uuid())
  user      User     @relation(fields: [userId], references: [id])
  userId    String
  product   Product  @relation(fields: [productId], references: [id])
  productId String
  quantity  Int      @default(1)
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}

model Order {
  id            String        @id @default(uuid())
  user          User          @relation(fields: [userId], references: [id])
  userId        String
  address       Address       @relation(fields: [addressId], references: [id])
  addressId     String
  totalPrice    Float
  status        String        @default("PENDING")
  paymentMethod String?       @default("cash")
  paymentStatus String?       @default("PENDING")
  paymentUrl    String?
  orderDetails  OrderDetail[]
  createdAt     DateTime      @default(now())
  updatedAt     DateTime      @updatedAt
}

model OrderDetail {
  id        String  @id @default(uuid())
```

```
  order     Order   @relation(fields: [orderId], references: [id])
  orderId   String
  product   Product @relation(fields: [productId], references: [id])
  productId String
  quantity  Int
  price     Float
}

model Comment {
  id        String   @id @default(uuid())
  content   String
  product   Product  @relation(fields: [productId], references: [id], onDelete: Cascade)
  productId String
  user      User     @relation(fields: [userId], references: [id], onDelete: Cascade)
  userId    String
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}

model Watchlist {
  id        String   @id @default(uuid())
  user      User     @relation(fields: [userId], references: [id], onDelete: Cascade)
  userId    String
  product   Product  @relation(fields: [productId], references: [id], onDelete: Cascade)
  productId String
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}

model Slider {
  id          String   @id @default(uuid())
  title       String
  description String
  imageUrl    String
  link        String
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
}

enum Role {
  ADMIN
  USER
  SUPER_ADMIN
}
```

```
enum OrderStatus {
  PENDING
  COMPLETED
  CANCELLED
}

enum RequestStatus {
  PENDING
  APPROVED
  REJECTED
}
```

Лістинг StripePaymentForm.tsx

```
import { useState, useEffect } from "react";
import { Loader2, CreditCard } from "lucide-react";
import {
  PaymentElement,
  useStripe,
  useElements,
} from "@stripe/react-stripe-js";
import convertToSubcurrency from "@/libs/convertToSubcurrency";

type StripePaymentFormProps = {
  amount: number;
  onSuccess: (paymentUrl?: string) => void;
};

const StripePaymentForm = ({ amount, onSuccess }: StripePaymentFormProps) => {
  const stripe = useStripe();
  const elements = useElements();
  const [errorMessage, setErrorMessage] = useState<string | undefined>();
  const [clientSecret, setClientSecret] = useState("");
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    fetch("/api/create-payment-intent", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ amount: convertToSubcurrency(amount) }),
    })
```

```
    .then((res) => res.json())
    .then((data) => setClientSecret(data.clientSecret))
    .catch((err) =>
      setErrorMessage("Failed to initialize payment: " + err.message)
    );
}, [amount]);

const handleSubmit = async (event: React.FormEvent<HTMLFormElement>) => {
  event.preventDefault();
  setLoading(true);

  if (!stripe || !elements) {
    setLoading(false);
    return;
  }

  const { error: submitError } = await elements.submit();

  if (submitError) {
    setErrorMessage(submitError.message);
    setLoading(false);
    return;
  }

  const { error, paymentIntent } = await stripe.confirmPayment({
    elements,
    clientSecret,
    confirmParams: {
      return_url: window.location.origin + "/payment-success",
    },
    redirect: "if_required",
  });

  if (error) {
    setErrorMessage(error.message);
    setLoading(false);
  } else {
    const paymentUrl = paymentIntent?.id
      ? `https://dashboard.stripe.com/payments/${paymentIntent.id}`
      : undefined;
    onSuccess(paymentUrl);
  }
};
```

```
  if (!clientSecret || !stripe || !elements) {
    return (
      <div className="flex items-center justify-center p-6">
        <Loader2 className="h-8 w-8 animate-spin text-pink-600" />
      </div>
    );
  }

  return (
    <form onSubmit={handleSubmit} className="space-y-4 mt-4">
      <PaymentElement />

      {errorMessage && (
        <div className="bg-red-50 text-red-700 p-3 rounded-md text-sm">
          {errorMessage}
        </div>
      )}

      <button
        disabled={!stripe || loading}
        className="w-full py-4 px-6 bg-pink-600 text-white rounded-lg font-medium hover:bg-pink-
700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-pink-500 disabled:opacity-50
disabled:cursor-not-allowed transition-colors flex items-center justify-center gap-2"
      >
        {loading ? (
          <>
            <Loader2 className="h-5 w-5 animate-spin" />
            Processing Payment...
          </>
        ) : (
          <>
            <CreditCard className="h-5 w-5" />
            Pay ${amount.toFixed(2)}
          </>
        )}
      </button>
    </form>
  );
};

export default StripePaymentForm;
```

Лістинг product.ts

```
"use server";
import { prisma } from "@/libs/prisma";
import { errorResponse, successResponse } from "@/utils/response";
import {
  createProductSchema,
  updateProductSchema,
} from "@/validation/productSchema";
import { revalidatePath } from "next/cache";
import { savePhotos } from "./photoSave";
import { getServerSession } from "next-auth";
import { authOptions } from "@/libs/authOptions";

export const createProduct = async ({
  name,
  description,
  price,
  stock,
  categoryId,
  photos,
}: {
  name: string;
  description?: string;
  price: number;
  stock: number;
  categoryId: string;
  photos: File[];
}) => {
  try {
    const session = await getServerSession(authOptions);

    if (
      !session ||
      (session.user.role !== "ADMIN" && session.user.role !== "SUPER_ADMIN")
    ) {
      return errorResponse(
        "Access denied: You are not authorized to perform this action"
      );
    }

    const photoUrls = await savePhotos(photos);

    const product = await prisma.product.create({
      data: {
        name,
```

```
        description,
        price,
        stock,
        categoryId,
        images: {
          create: photoUrls.map((url) => ({
            imageUrl: url,
          })),
        },
      },
      include: {
        images: true,
      },
    });

    return successResponse(product, "Product created successfully");
  } catch (error: any) {
    return errorResponse(`Error creating product: ${error.message}`);
  }
};

export const getAllProducts = async ({
  page = 1,
  limit = 10,
  filters = {},
}: {
  page?: number;
  limit?: number;
  filters?: {
    category?: string;
    minPrice?: string;
    maxPrice?: string;
    search?: string;
  };
}) => {
  try {
    const offset = (page - 1) * limit;

    const whereClause: any = {};

    if (filters.category) {
      whereClause.category = {
        name: {
          equals: filters.category,
```

```
      mode: "insensitive",
    },
  };
}

if (filters.minPrice) {
  whereClause.price = {
    ...(whereClause.price || {}),
    gte: Number(filters.minPrice),
  };
}

if (filters.maxPrice) {
  whereClause.price = {
    ...(whereClause.price || {}),
    lte: Number(filters.maxPrice),
  };
}

if (filters.search) {
  whereClause.OR = [
    { name: { contains: filters.search, mode: "insensitive" } },
    { description: { contains: filters.search, mode: "insensitive" } },
  ];
}

const [products, total] = await Promise.all([
  prisma.product.findMany({
    skip: offset,
    take: limit,
    where: whereClause,
    include: {
      images: true,
      category: true,
    },
    orderBy: {
      createdAt: "desc",
    },
  }),
  prisma.product.count({ where: whereClause }),
]);

if (products.length === 0) {
  return errorResponse("No products found");
```

```
    }

    const totalPages = Math.ceil(total / limit);

    return successResponse(
      {
        products,
        total,
        totalPages,
        currentPage: page,
        hasNextPage: page < totalPages,
        hasPreviousPage: page > 1,
      },
      "Products retrieved successfully"
    );
  } catch (error: any) {
    return errorResponse(`Error retrieving products: ${error.message}`);
  }
};

export const getProductById = async (id: string) => {
  try {
    const product = await prisma.product.findUnique({
      where: { id },
      include: {
        images: true,
        category: true,
      },
    });

    if (!product) {
      return errorResponse("Product not found");
    }

    return successResponse(product, "Product retrieved successfully");
  } catch (error: any) {
    return errorResponse(`Error retrieving product: ${error.message}`);
  }
};

export const updateProduct = async (
  id: string,
  updateProductDto: {
    name?: string;
```

```
    description?: string;
    price?: number;
    stock?: number;
    categoryId?: string;
    newImages?: File[];
  }
) => {
  try {
    const session = await getServerSession(authOptions);

    if (
      !session ||
      (session.user.role !== "ADMIN" && session.user.role !== "SUPER_ADMIN")
    ) {
      return errorResponse(
        "Access denied: You are not authorized to perform this action"
      );
    }

    const parsedData = updateProductSchema.parse(updateProductDto);

    const product = await prisma.product.findUnique({
      where: { id },
    });

    if (!product) {
      return errorResponse("Product not found");
    }

    let newImageUrls: string[] = [];
    if (updateProductDto.newImages && updateProductDto.newImages.length > 0) {
      newImageUrls = await savePhotos(updateProductDto.newImages);
    }

    const updatedProduct = await prisma.product.update({
      where: { id },
      data: {
        name: parsedData.name,
        description: parsedData.description,
        price: parsedData.price,
        stock: parsedData.stock,
        categoryId: parsedData.categoryId,
        images:
          newImageUrls.length > 0
```

```
            ? {
                deleteMany: {},
                create: newImageUrls.map((url) => ({
                  imageUrl: url,
                })),
              }
            : undefined,
        },
        include: {
          images: true,
        },
      });

      return successResponse(updatedProduct, "Product updated successfully");
    } catch (error: any) {
      return errorResponse(`Error updating product: ${error.message}`);
    }
};

export const deleteProduct = async (id: string) => {
  try {
    const session = await getServerSession(authOptions);

    if (
      !session ||
      (session.user.role !== "ADMIN" && session.user.role !== "SUPER_ADMIN")
    ) {
      return errorResponse(
        "Access denied: You are not authorized to perform this action"
      );
    }

    const product = await prisma.product.findUnique({
      where: { id },
    });

    if (!product) {
      return errorResponse("Product not found");
    }

    await prisma.product.delete({
      where: { id },
    });
```

```
    return successResponse(null, "Product deleted successfully");
  } catch (error: any) {
    return errorResponse(`Error deleting product: ${error.message}`);
  }
};

export const getProductsByCategoryName = async (
  categoryName: string,
  page: number,
  limit: number
) => {
  try {
    const offset = (page - 1) * limit;

    const category = await prisma.category.findUnique({
      where: { name: categoryName },
    });

    if (!category) {
      return {
        success: false,
        message: "Category not found",
      };
    }

    const [products, totalProducts] = await Promise.all([
      prisma.product.findMany({
        where: { categoryId: category.id },
        skip: offset,
        take: limit,
        include: {
          images: true,
        },
        orderBy: {
          createdAt: "desc",
        },
      }),
      prisma.product.count({
        where: { categoryId: category.id },
      }),
    ]);

    const totalPages = Math.ceil(totalProducts / limit);
```

```
    return {
      success: true,
      data: {
        products,
        currentPage: page,
        totalPages,
        hasNextPage: page < totalPages,
        hasPreviousPage: page > 1,
      },
    };
  } catch (error: any) {
    return { success: false, message: error.message };
  }
};
```