Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

Допущено до захисту
Завідувач кафедри комп'ютерних наук
доктор технічних наук, професор
Андрій БОНДАРЧУК
«01» червня 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «бакалавр»
спеціальність 122 Комп'ютерні науки
освітня програма 122.00.01 Інформатика

Тема: ЦИФРОВИЙ ЗАСТОСУНОК УПРАВЛІННЯ АСОРТИМЕНТОМ ЧОЛОВІЧОГО ВЕРХНЬОГО ОДЯГУ

Виконав
студент групи Інб-2-22-4.0д
Гедз Денис Олександрович
_____
(підпис)

Науковий керівник
кандидат технічних наук,
доцент
Яскевич В.О.
_____
(підпис)

Київ – 2026

Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп’ютерних наук

«Затверджую»
Завідувач кафедри комп’ютерних наук
доктор технічних наук, професор
Андрій БОНДАРЧУК
« 01 » жовтня 2025 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА
«ЦИФРОВИЙ ЗАСТОСУНОК УПРАВЛІННЯ АСОРТИМЕНТОМ ЧОЛОВІЧОГО ВЕРХНЬОГО ОДЯГУ»

Виконавець – студент групи ІНб-2-22-4.0д,
спеціальності 122 Комп’ютерні науки,
освітньої програми 122.00.01 Інформатика
Гедз Денис Олександрович

1. Вихідні дані: реальні дані каталогу інтернет-магазину (товари, категорії, атрибути, зображення) у реляційній моделі з EAV-фрагментом, офіційна документація Next.js / React, Node.js / Express.js, PostgreSQL, Stripe API, JWT, bcrypt, Sharp, а також наукові та галузеві публікації щодо EAV-моделі, фасетної фільтрації, ідемпотентної обробки платежів і безпеки вебзастосунків.

2. Здійснити аудит існуючих рішень для побудови інтернет-магазинів та обґрунтувати вибір власної реалізації. Обґрунтувати мету, об'єкт, предмет та обрати методи дослідження. Спроєктувати клієнт-серверну архітектуру вебзастосунку з розподілом відповідальності між шарами Presentation (Next.js / React), Application Logic (Express.js / Node.js) та Data (PostgreSQL). Розробити алгоритмічне забезпечення багатокритеріальної фасетної фільтрації товарів з підрахунком кількості позицій за один SQL-запит. Спроєктувати та імплементувати наскрізний процес оформлення замовлення з інтеграцією Stripe Payment Intents, ідемпотентною обробкою вебхуків та узгодженим списанням залишків у транзакціях з блокуванням рядків. Імплементувати модулі персистентного кошика (Zustand + localStorage), безпеки (bcrypt, JWT у HttpOnly-cookie, рольові middleware) та оптимізації зображень варіантів товару (Sharp). Провести експериментальне тестування швидкодії застосунку, профілювання часу відгуку API та коректності транзакційних сценаріїв. Оформити роботу відповідно до затверджених вимог. Підготувати тези доповіді за темою роботи.

3. Пояснювальна записка: Обсяг близько 60 стор. формату А4 комп’ютерного набору з дотриманням вимог стандарту і методичних рекомендацій кафедри.

4. Графічні матеріали: Комп'ютерна презентація доповіді.

5. Строк подання роботи на кафедру: «1» червня 2026 р.

Науковий керівник
к.т.н., доцент
_____ Яскевич В.О.
_____ дата

Виконавець:
_____ Гедз Д.О.
_____ дата

Календарний план

№	Етап виконання роботи	Термін виконання	Примітка
1	Формування теми дипломної роботи бакалавра та її затвердження	1.10.25 - 15.10.25	виконано
2	Аналіз предметної області, огляд існуючих рішень та написання першого розділу пояснювальної записки	20.10.25 - 28.11.25	виконано
3	Проєктування клієнт-серверної архітектури вебзастосунку (Next.js / Express.js / PostgreSQL) та розробка схеми реляційної бази даних з EAV-фрагментом для каталогу товарів	25.11.25 - 20.12.25	виконано
4	Розробка користувацького інтерфейсу клієнтської частини за допомогою Next.js / React (TypeScript) та налаштування маршрутизації між сторінками каталогу, картки товару й кошика	21.12.25 - 15.01.26	виконано
5	Реалізація серверного API для управління каталогом товарів, варіантами, категоріями та зображеннями на основі Express.js	28.01.26 - 15.02.26	виконано
6	Інтеграція платіжного шлюзу Stripe (Payment Intents) та реалізація ідемпотентної обробки вебхуків з транзакційним списанням залишків	16.02.26 - 22.02.26	виконано
7	Впровадження підсистеми безпеки: хешування паролів (bcrypt), автентифікація через JWT у захищених cookie, рольове розмежування доступу через middleware	03.03.26 - 09.03.26	виконано
8	Розробка механізму багатокритеріальної фасетної фільтрації товарів з підрахунком кількості позицій за один SQL-запит	11.03.26 - 15.03.26	виконано
9	Написання другого та третього розділів пояснювальної записки	16.03.26 - 15.04.26	виконано
10	Подання роботи на перевірку, проходження антиплагіату, внесення правок керівника	29.04.26 - 15.05.26	виконано
11	Підготовка презентаційних матеріалів, тексту доповіді та захист кваліфікаційної роботи	16.05.26 - 15.06.26	

«Затверджую»

Науковий керівник **Індивідуальний план склав**

к.т.н., доцент, Яскевич В.О. Гедз Д.О.

РЕФЕРАТ

Кваліфікаційна робота бакалавра: 75 с., 19 рис., 4 табл., 33 джерел.

Актуальність. У сучасних умовах розвитку електронної комерції якість цифрового сервісу та ефективне управління товарним асортиментом є важливими чинниками конкурентоспроможності інтернет-магазину. Особливо актуальною ця проблема є для товарів із високою варіативністю (розміри, кольори, матеріали), де помилки в каталозі, цінах або складських залишках можуть призводити до повернень і втрати довіри користувачів [30].

Покупці очікують актуального каталогу, зручної фільтрації товарів, прозорого процесу оформлення замовлення та безпечної онлайн-оплати. У зв’язку з цим розробка власного e-commerce вебзастосунку дозволяє забезпечити гнучке управління каталогом, інтеграцію платіжних сервісів і адаптацію системи до потреб бізнесу.

Об’єкт дослідження: інформаційні процеси управління каталогом товарів та оформлення замовлень в e-commerce системах.

Предмет дослідження: методи проєктування та реалізації клієнтської і серверної частин e-commerce вебзастосунку з підтримкою управління асортиментом, фасетної фільтрації та процесів оформлення замовлень.

Мета роботи: розробка e-commerce вебзастосунку для управління асортиментом чоловічого верхнього одягу з підтримкою каталогу товарів, фасетної фільтрації, оформлення замовлень та онлайн-оплати.

Завдання:

1. Проаналізувати сучасні підходи до побудови e-commerce систем та управління товарним асортиментом.
2. Спроєктувати структуру бази даних для каталогу товарів, варіантів та атрибутів.
3. Розробити клієнтську частину вебзастосунку з каталогом товарів, фільтрацією та кошиком.
4. Реалізувати серверну частину застосунку та API для роботи з даними.

5. Реалізувати механізм оформлення замовлень та інтеграцію онлайн-оплати.

6. Провести тестування основних функціональних можливостей вебзастосунку.

Основні результати. Розроблено e-commerce вебзастосунок на основі стеку Next.js, Express та PostgreSQL з підтримкою JWT-автентифікації та інтеграцією Stripe. У ході дослідження встановлено, що використання моделі EAV є ефективним підходом для реалізації каталогу товарів із варіативними атрибутами (розмір, колір, матеріал), оскільки вона забезпечує гнучке масштабування структури каталогу без зміни схеми бази даних.

Практичне значення дослідження. Дослідження продуктивності SQL-запитів показало, що застосування складених індексів суттєво зменшує час вибірки даних: при обсязі каталогу до 10 000 позицій середній час відповіді стабілізується на рівні 3–7 мс, тоді як без індексів спостерігається майже лінійне зростання часу виконання запитів. Це підтверджує доцільність використання індексації для реалізації фасетної фільтрації та каталогових сценаріїв в e-commerce системах [28].

Ключові слова: next.js, react, typescript, postgresql, eav-модель, фасетна фільтрація, stripe, jwt-автентифікація, rest api, express, управління асортиментом, e-commerce.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Скорочення	Розшифрування
API	прикладний програмний інтерфейс
bcrypt	Алгоритм криптографічного хешування паролів
CORS	Cross-Origin Resource Sharing – механізм контролю міжсайтових запитів
CSRF	Cross-Site Request Forgery – міжсайтова підробка запитів
DDL	Data Definition Language – мова визначення структури даних
EAV	Entity-Attribute-Value – модель зберігання атрибутів через три пов'язані таблиці
JWT	JSON Web Token – стандарт токена автентифікації
ORM	Object-Relational Mapping – технологія відображення об'єктів на таблиці БД
PCI-DSS	Payment Card Industry Data Security Standard – стандарт безпеки платіжних карток
REST	Representational State Transfer – архітектурний стиль побудови API
SEO	Search Engine Optimization – оптимізація для пошукових систем
SKU	Stock Keeping Unit – унікальний ідентифікатор товарного варіанту
SQL	Structured Query Language – мова структурованих запитів до реляційних БД
SSL	Secure Sockets Layer – протокол захищеного з'єднання
СУБД	Система управління базами даних
TypeScript	Надбудова над JavaScript зі статичною типізацією
UI	User Interface – інтерфейс користувача

ВСТУП

Електронна комерція сьогодні розвивається швидко, і конкуренція в ній уже не зводиться лише до ціни — багато важить те, наскільки злагоджено працює сам інтернет-магазин. Для категорії чоловічого верхнього одягу це особливо помітно: один товар часто існує у вигляді кількох розмірів, кольорів, складу матеріалу, тож вести такий каталог вручну стає важко. Накопичуються розбіжності в цінах, неактуальні залишки, плутанина між варіантами — і все це негативно впливає на продажі та довіру покупця. Окремий головний біль — узгодженість між адмінкою (де редагують каталог, варіанти й залишки) та вітриною, на якій клієнт обирає товар, кладе в кошик і платить онлайн.

У цій роботі я будую вебзастосунок інтернет-магазину, в якому основна вага припадає на кілька задач: упорядковане зберігання товарів і їхніх варіантів, відображення каталогу, кошик і покрокове оформлення замовлення, а також фасетна фільтрація з підрахунком лічильників (facets) для інтерфейсу. Практична користь — у тому, що готове рішення придатне як основа для малих і середніх магазинів, яким не по кишені дорогі ERP, але потрібен робочий інструмент керування асортиментом.

Мета бакалаврської роботи — спроєктувати і реалізувати вебзастосунок електронної комерції для управління асортиментом чоловічого верхнього одягу. Він має централізовано вести каталог товарів та їх варіантів, підтримувати фасетну фільтрацію за атрибутами, рахувати складські залишки і вести покупця через оформлення замовлення з онлайн-оплатою. Завдання роботи такі:

1. Розібрати предметну область e-commerce і вимоги до систем управління асортиментом та продажами;
2. Спроєктувати структуру даних каталогу — товари, варіанти, атрибути — і визначити зв'язки між основними сутностями;
3. Написати серверну частину застосунку (API) для роботи з каталогом, фільтрацією, замовленнями і допоміжними довідниками;

4. Реалізувати клієнтську частину (вітрину) з переглядом товарів, фільтрацією, кошиком та етапами checkout;
5. Інтегрувати онлайн-оплату й обробку результатів платежу;
6. Провести тестування ключових сценаріїв (каталог, фільтрація, кошик, замовлення, оплата) і оцінити коректність роботи системи.

Об'єкт дослідження — процеси управління товарним асортиментом і продажами в інтернет-магазині.

Предмет дослідження — методи й програмні засоби, якими я проєктую та реалізую вебзастосунок для управління каталогом товарів, його варіантами й атрибутами, а також для оформлення замовлення з онлайн-оплатою.

Методи й засоби дослідження**.** У роботі поєднано аналіз предметної області та функціональних вимог, моделювання реляційної бази даних, проєктування API і взаємодії компонентів, компонентний підхід у розробці інтерфейсу, а також практичну перевірку основних користувацьких сценаріїв та цілісності даних. Реалізація спирається на сучасний веб-стек: клієнт на Next.js/React, серверна частина на Express.js, база даних PostgreSQL; онлайн-оплату забезпечує платіжна платформа Stripe.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ СУЧАСНИХ ПІДХОДІВ ДО УПРАВЛІННЯ АСОРТИМЕНТОМ ТА КАТАЛОГОМ E-COMMERCE

1.1 Вимоги та тенденції розвитку e-commerce платформ: каталог, варіанти товарів і фасетна фільтрація.

Електронна комерція залишається однією з найбільш мінливих галузей сучасної економіки. Конкурентоспроможність магазину тут вимірюється не лише ціною й якістю товару, а й тим, наскільки злагоджено працюють цифрові процеси: актуальний каталог, точні дані про наявність, швидкий пошук, зрозуміле оформлення замовлення. Якщо магазин торгує товаром із високою варіативністю (а чоловічий верхній одяг — саме такий випадок), задача ускладнюється: один артикул часто розпадається на десятки варіантів за розміром, кольором і матеріалом, і кожен варіант живе власною ціною, фотонабором та залишком на складі. При ручному веденні таких даних легко припуститися помилки — продублювати позицію, забути оновити ціну, залишити нульовий «доступний» товар на вітрині, — а це одразу б'є по конверсії і повертає клієнтів із поверненнями.

Каталог потрібно мислити як систему взаємопов'язаних сутностей, інакше керувати ним стає неможливо. Інформаційна модель тут будується навколо ієрархії «категорії → товари → варіанти» з окремими довідниками атрибутів і значень («колір», «розмір») та зв'язками між ними. Такий підхід підтримує каталог керованим у довгостроковій перспективі: можна спокійно додавати нові категорії, не переписуючи логіку вітрини, і додавати нові значення атрибутів, не плодячи дублів у структурі таблиць чи в коді. Для товарів з великою кількістю варіантів модель має витримувати рутинні операції — додавання та редагування товару, створення варіантів (SKU), прив'язку атрибутів, завантаження й упорядкування зображень, контроль залишків і видимості позицій.

Поряд із цим важлива інша частина — інтерфейс. Покупець має знаходити потрібний товар швидко, і в сучасних e-commerce-системах це збирається з трьох речей: пошук, сортування та фасетна фільтрація. На відміну від звичайних фільтрів, фасети не просто відсіюють товари за значенням атрибута, а ще й підказують користувачеві лічильники: скільки позицій лишається для кожного значення, які значення комбінуються між собою. Відповідно, бекенд повинен уміти не тільки повертати список товарів за умовами, а й рахувати facets для поточного набору результатів. Це окрема задача побудови запитів до БД, особливо коли атрибути зберігаються через зв'язкові таблиці (варіант ↔ значення атрибута). Поряд із цим на якість пошуку впливають пагінація та сортування — наприклад, за ціною чи актуальністю — які мають коректно поєднуватися з фільтрами.

Інший великий блок вимог стосується транзакцій, тобто самої покупки. Типовий сценарій лінійний: зібрав кошик, ввів контакти й адресу, обрав доставку, створив замовлення, оплатив. Тут понад усе важить узгодженість даних — після успішного платежу замовлення повинно одразу опинитися в статусі «оплачено», а залишки на складі — зменшитися рівно на те, що купили. У реальних умовах події від платіжної платформи можуть приходити повторно або із затримкою, тож серверна логіка має бути ідемпотентною: критичні операції (наприклад, списання залишків) не повинні виконуватися двічі. Тому інтеграція з платіжкою — це не просто «кнопка оплати» на фронті, а повноцінний серверний процес із прийомом подій, перевіркою станів і захистом від повторів.

У роботі розроблено вебзастосунок інтернет-магазину з трьох частин: клієнтська вітрина, серверна частина і реляційна база даних. Система веде каталог з варіантами й атрибутами, дає користувачу пошук і фільтрацію, проводить його через checkout і приймає онлайн-оплату через платіжний сервіс. Для адміна передбачено окремий інтерфейс — створювати й редагувати категорії та товари, керувати варіантами і зображеннями, тримати в актуальному стані довідники, від яких залежить фільтрація.

Тож огляд сучасних підходів до управління каталогом і асортиментом в e-commerce дозволяє чітко окреслити вимоги до даних і функціональності, обґрунтувати модель представлення атрибутів і виділити ті технічні рішення, що визначають продуктивність та надійність застосунку. У підрозділі 1.1 далі я розгляну ключові вимоги і тенденції, які формують очікування до e-commerce-платформ у частині каталогу, варіантів та фасетної фільтрації.

1.2 Основні завдання інформаційної системи інтернет-магазину та особливості її функціонування

Українська електронна комерція — частина глобального ринку, тож тренди, що визначають розвиток e-commerce-платформ загалом (зростання ваги цифрового каталогу, очікування швидкого пошуку, персоналізація, зручне оформлення покупки), однаково актуальні і для вітчизняних магазинів. На практиці ефективність магазину залежить від того, наскільки злагоджено працює його операційне ядро: ведення асортименту, контроль залишків, обробка замовлень, інтеграція з доставкою та прийом платежів. Коли немає централізованої системи, типові проблеми не змушують себе чекати — дубльовані позиції, помилки в описах і цінах, неактуальні залишки, неправильно показані варіанти (розмір, колір), а в результаті — втрачені продажі і зростання повернень.

Розгляньмо роботу типового інтернет-магазину як інформаційної системи. Зазвичай вона ділиться на дві логічно різні частини — клієнтську вітрину й адміністративний контур. Клієнт працює з вітриною: переглядає товари, відбирає за фільтрами, кладе в кошик, проходить checkout, підтверджує оплату. Адмінська частина відповідає за наповнення й актуальність даних — створення категорій, додавання товарів, формування варіантів (SKU), керування атрибутами, завантаження зображень, оновлення цін і складських залишків. Обидві частини повинні працювати за одними правилами обробки і ділити спільну базу даних, інакше неминуче з'являється розсинхрон між бек-офісом і вітриною.

Функціонально магазин розпадається на кілька груп задач. Перша — управління асортиментом: ієрархія «категорії → товари → варіанти», описова інформація про товар, характеристики варіантів і атрибути (розмір, колір) у вигляді довідників, прив'язаних до конкретних варіантів. Друга — представлення каталогу на вітрині: сортування, пагінація, фасетна фільтрація, що дозволяє комбінувати умови та показує доступні значення з лічильниками. Третя — транзакційна частина: кошик, формування замовлення, дані доставки, підрахунок підсумку (з доставкою і податком), запуск онлайн-оплати та обробка її результату.---

Особливо чутлива тема — цілісність даних при оплаті й виконанні замовлення. Після підтвердження платежу система повинна змінити статус замовлення і коректно зменшити залишки по куплених варіантах. Платіжний сервіс при цьому здатен віддавати події повторно або із затримкою, тому серверна логіка зобов'язана бути стійкою до повторів і не виконувати критичні операції — як-от списання залишків — більше одного разу.

Тож основні задачі інформаційної системи інтернет-магазину — централізоване управління асортиментом, акуратне представлення каталогу на вітрині, повний цикл оформлення замовлення з оплатою і узгодженість та надійність даних на кожному з етапів. Функціональні модулі системи та їх призначення подано в таблиці 1.1.

Таблиця 1.1

Функціональні модулі інтернет-магазину та їх призначення

№	**Модуль /підсистема**	**Призначення**	**Основні операції / сценарії**
1	Каталог і категорії	Організація асортименту та навігації у вітрині	створення/редагування категорій; прив’язка товарів до категорій; перегляд каталогу
2	Товари (product)	Зберігання основних даних про товар	створення/редагування товару (назва, опис, базова ціна, slug); прив’язка матеріалу; публікація у вітрині
3	Варіанти товару (variant, SKU)	Відображення конкретних	створення/редагування SKU; ціна варіанту;

		модифікацій товару (розмір/колір) і облік залишків	залишок на складі; вибір варіанту на сторінці товару
4	Атрибути та значення атрибутів	Гнучке задання характеристик товарів і варіантів	ведення довідника атрибутів (size, color); ведення значень; прив’язка значень до варіантів
5	Матеріали (fabric/materials)	Нормалізоване зберігання інформації про матеріали виробів	додавання/перегляд матеріалів; використання матеріалу у товарі; фільтрація за матеріалом
6	Зображення варіантів	Візуальне представлення товарів у вітрині	завантаження/видалення зображень; упорядкування (image_order); формування thumbnail
7	Пошук, сортування та пагінація	Підвищення зручності навігації по каталогу	сортування (за ціною тощо); пагінація; стабільні URL-параметри
8	Фасетна фільтрація (facets)	Відбір товарів за атрибутами з підрахунком доступних значень	фільтри за color/size/fabric; лічильники значень; оновлення списку товарів і facets при зміні URL
9	Кошик	Формування переліку товарів до покупки	додавання/видалення позицій; зміна кількості; збереження стану (localStorage); підрахунок subtotal
10	Checkout (оформлення замовлення)	Збір даних покупця та підготовка замовлення до оплати	введення контактів/адреси; вибір доставки; розрахунок підсумку (tax + shipping)
11	Доставка (shipping methods)	Управління способами доставки і їх вартістю	отримання доступних методів; вибір методу; включення вартості у total
12	Замовлення (orders)	Фіксація покупки та її стану	створення замовлення; збереження snapshot-даних; перегляд замовлень користувача; зміна статусу (admin)
13	Онлайн-оплата (Stripe)	Приймання платежів та	створення PaymentIntent; підтвердження оплати;

		підтвердження оплати	webhook-обробка подій; ідемпотентність
14	Облік залишків	Забезпечення коректної наявності товарів	перевірка stock перед створенням order; списання stock після успішної оплати; недопущення oversell
15	Автентифікація та ролі	Контроль доступу до функцій системи	login; JWT у cookie; ролі (admin/customer); захист адмін-операцій
16	Wishlist (список бажаного)	Збереження обраних товарів користувачем	додати/видалити варіант у wishlist; перегляд списку

У сучасних джерелах розвиток та підвищення ефективності e-commerce-систем розглядають комплексно: цифрові канали продажів безпосередньо залежать від якості даних, швидкодії й того, наскільки зручно клієнту взаємодіяти з вітриною. Перерахую основні напрями досліджень і прикладних підходів, що мають значення для проєктування магазину з варіативним асортиментом:

- розвиток моделей даних товарного каталогу та варіантів (атрибутний підхід, нормалізація довідників характеристик) і узгодженість даних між адміністративною частиною та вітриною;
- побудова механізмів пошуку, сортування і фасетної фільтрації з лічильниками значень (facets), що скорочують час вибору товару і піднімають конверсію;
- оптимізація запитів до реляційних БД для каталогових сценаріїв — індексація, агрегації, боротьба з «row explosion» у складних join'ax — а також стабільна робота при зростанні обсягів даних [8];
- організація транзакційного контуру магазину: кошик, оформлення замовлення, snapshot-фіксація позицій у замовленні та коректний облік залишків;
- інтеграція платіжних сервісів і надійність процесу оплати (ідемпотентність, обробка webhook-подій, узгодженість статусів замовлень і складських залишків);

- поліпшення користувацького досвіду (UX) у критичних сценаріях: швидке завантаження каталогу, зрозумілий вибір варіанта (розмір/колір), мінімальна кількість кроків у checkout, інформативний зворотний зв'язок у разі помилок [27];
- питання безпеки та контролю доступу в e-commerce-застосунках — автентифікація, розмежування ролей адміністратора й покупця, захист адміністративних операцій.

Перелічені напрями формують вимоги до структури даних, серверної логіки та інтерфейсу, які я враховую при проєктуванні та реалізації свого вебзастосунку.

1.3 Основні механізми роботи інформаційної системи інтернет-магазину

На практиці видно: без централізованої інформаційної системи жоден інтернет-магазин не забезпечить безпомилковий облік і продажі, бо асортимент, ціни, залишки, замовлення й взаємодія з платіжними сервісами безперервно змінюються. Для категорій із високою варіативністю (чоловічий верхній одяг — типовий приклад) принципово важливо мати структуровані товарні варіанти (SKU) з їх атрибутами (розмір, колір, матеріал). Коли єдиного джерела даних немає, з'являються типові проблеми — неузгоджені описи, дубльовані позиції, помилки в залишках, некоректні варіанти у вітрині — і як наслідок втрата продажів та довіри покупця.

З погляду процесів магазин як інформаційна система розпадається на два контури: адміністративний (керування асортиментом і даними) та користувацький (вітрина, вибір товару, покупка). Адмінська частина створює й актуалізує каталог — категорії, товари, варіанти, атрибути, зображення — і підтримує в актуальному стані складські залишки. Користувацький контур веде сценарій покупця: перегляд і фільтрація, кошик, оформлення замовлення, вибір доставки й оплата. Щоб обидва контури працювали узгоджено, потрібна

єдина база даних, в якій всі сутності зв'язані, а зміни з адмінки одразу видно на вітрині — без затримок і спотворень.

Основні функціональні блоки інформаційної системи інтернет-магазину зручно показати на узагальненій схемі: на ній видно чотири ключові групи — вітрина (каталог, фільтрація, варіанти), кошик і оформлення замовлення, обліковий запис покупця, а також адміністрування каталогу (рис. 1.1).

Рисунок. 1.1. Основні функціональні блоки інформаційної системи інтернет-магазину

Основні механізми роботи інформаційної системи інтернет-магазину можна згрупувати за функціональними напрямами:

- ведення каталогу товарів: створення та редагування категорій і товарів, побудова зрозумілої ієрархії, унікальні ідентифікатори (slug) і описові характеристики товару;
- робота з варіантами (SKU) та атрибутами: опис варіантів за розміром і кольором, прив'язка значень атрибутів до варіантів — це й дає гнучкість моделі даних і можливість фасетної фільтрації;
- візуальне представлення товарів: завантаження зображень для варіантів, їх упорядкування і формування прев'ю (thumbnail) для каталогу та сторінки товару;

- представлення каталогу на вітрині: пошук, сортування і пагінація, плюс фасетна фільтрація з лічильниками значень (facets), щоб користувач міг швидко звузити вибір до релевантних позицій;
- кошик: додавання/видалення позицій і зміна кількості, збереження стану кошика, обчислення проміжної вартості замовлення;
- оформлення замовлення (checkout): збір контактів і адреси, вибір способу доставки, підрахунок підсумку з урахуванням доставки та податку;
- онлайн-оплата: інтеграція з платіжним сервісом для створення платіжного наміру (Payment Intent) і завершення оплати на стороні користувача;
- результат платежу та облік залишків: зміна статусу замовлення після підтвердження оплати і коректне списання залишків по варіантах; окремо враховуємо ідемпотентність серверної логіки, щоб дублювання платіжних подій не призводило до повторного списання;
- контроль доступу: автентифікація користувачів і розмежування прав (адміністративні операції виконуються лише користувачем із відповідною роллю).

Тож сучасний інтернет-магазин — це комплексна інформаційна система, в якій сходяться ведення каталогу та варіантів, відбір товарів на вітрині, транзакційні процеси оформлення замовлення й оплата, а ще облік залишків і контроль доступу. Щоб усе це працювало злагоджено, система має збирати і обробляти чималі обсяги даних — про категорії, товари, варіанти та атрибути, зображення, користувачів, доставку, замовлення, платежі. Саме тому автоматизація і стандартизація цих процесів — необхідна умова стабільної роботи e-commerce-застосунку і якісного клієнтського досвіду.

Для наочного відображення взаємодії основних компонентів системи під час оформлення замовлення на рис. 1.2 наведено життєвий цикл замовлення — від формування кошика до підтвердження оплати та оновлення складських залишків. Після формування кошика та введення даних доставки замовлення

створюється на сервері в межах транзакції зі статусом «pending», після чого ініціюється оплата через Stripe. Результат платежу надходить асинхронно — подією webhook, при обробці якої система перевіряє, чи замовлення вже не має статусу «paid»: це забезпечує ідемпотентність, тобто повторна доставка події не призводить до повторного списання залишків. Лише за першої успішної обробки виконується списання складських залишків і переведення замовлення у статус «оплачено».

1.4 Огляд наявних цифрових інструментів управління асортиментом в e-commerce

Розвиток e-commerce породив чимало цифрових рішень, з якими магазин можна запустити швидко і покрити базові процеси продажу. Умовно їх розкладають на три групи: SaaS-платформи, CMS-рішення з плагінами та headless/модульні платформи. У кожної групи свої сильні сторони і обмеження, які варто врахувати, перш ніж проєктувати власну інформаційну систему магазину.--

SaaS-платформи (приклад — Shopify) ґрунтуються на швидкому старті й «коробковому» наборі можливостей: готова вітрина, панель адміністратора, керування товарами й варіантами, інтеграції з оплатою і доставкою, базова аналітика. Головна перевага — мінімум часу на розгортання і підтримку інфраструктури. Проте й недоліків достатньо: глибокої кастомізації каталогу та бізнес-логіки практично немає, архітектуру даних диктує сама платформа, а розробник залишається в межах її тарифів і екосистеми інтеграцій. Якщо проєкту потрібна нестандартна модель атрибутів, специфічні правила фільтрації або власна логіка роботи з варіантами, такі обмеження стають критичними.--

CMS-рішення (WordPress + WooCommerce — типовий вибір) приваблюють легким стартом, низьким порогом входу й величезною кількістю готових плагінів. На них спокійно живуть магазини малого і середнього розміру: каталог, кошик, замовлення, інтеграції з оплатою й доставкою — все

це є. Сильна сторона — гнучкість UI на рівні тем і модулів. Ускладнення виникають, коли каталог зростає: складна фільтрація, велика кількість атрибутів і необхідність точного обліку залишків по SKU поступово навантажують систему і ускладнюють підтримку.-

Headless/модульні платформи (Medusa, Saleor, commercetools та подібні) йдуть від іншої архітектури — бекенд e-commerce відділений від фронтенду, спілкуються вони через API. Це дає простір: вітрину можна збирати під свої сценарії, оптимізувати UX, прописати власні правила каталогу й фільтрації, окремо масштабувати компоненти. Платять за це складністю впровадження: треба самостійно проєктувати архітектуру, налаштовувати інфраструктуру, інтегрувати оплату й доставку, писати адміністративні сценарії.

Розглянуті підходи показують одне: універсальні платформи зручні для швидкого старту, але як тільки до магазину додаються варіативний каталог (розмір/колір/матеріал), повноцінна фасетна фільтрація, точний облік залишків по варіантах і власна логіка checkout та оплати — раціональнішим стає окреме рішення під свою предметну область. Саме тому в моєму проєкті структура каталогу, модель даних атрибутів і механізми фільтрації узгоджені з реальними сценаріями інтернет-магазину чоловічого верхнього одягу, а платіжний контур побудовано з акцентом на надійну серверну обробку подій оплати.

У Таблиці 1.2 узагальнено порівняння типових підходів/платформ та розробленого рішення.

Характеристика SaaS (Shopify) CMS (WooCommerce) Headless платформи Розроблений вебзастосунок

Таблиця 1.2

Порівняння підходів до реалізації e-commerce платформи

Критерій	**Shopify (SaaS)**	**WooCommerce (WordPress)**	**Magento Open Source**	**Власний застосунок (Next.js + Express + PostgreSQL)**
Вбудована EAV-модель товарів	Ні — фіксована схема	Ні — атрибути реалізовані через	Так — EAV закладена в ядро	Так — власна реалізація на трьох таблицях: attribute,

	товару, кастомні атрибути лише через metafields і сторонні додатки	таксономії WordPress та плагіни (наприклад, ACF), без явної EAV	платформи з 2008 року	attribute_value, variant_attribute_value
Безкоштовне масштабування	Ні — обов'язкова підписка від $39/міс плюс комісія з продажів	Умовно — CMS безкоштовна, але більшість бізнес-плагінів платні ($50–300/рік за кожен)	Так — open source, оплата лише за хостинг	Так — open source стек, оплата лише за хостинг і керовану СУБД
Кастомні webhook для оплат	Ні — лише стандартні події Shopify, без можливості змінити серверну логіку обробки	Обмежено — обробка платежів інкапсульована у плагіни (Stripe for WooCommerce тощо), модифікація логіки потребує форку плагіна	Так, але через перевизначення модулів ядра (висока складність міграцій)	Так — повністю кастомний обробник payment_intent.succeeded від Stripe з ідемпотентним списанням залишків
Контроль над схемою БД	Відсутній	Обмежений — схема WordPress, прямі ALTER небезпечні через оновлення CMS	Частковий — EAV ядра ускладнює власні міграції	Повний — власні міграції, керовані індекси для фасетної фільтрації
Розширення фасетної фільтрації	Тільки через додатки магазину	Через плагіни на кшталт FacetWP; індекси не контролюються розробником	Через адмінку, але запити по EAV історично повільні без додаткового кешу	Реалізована на рівні SQL з контрольованими індексами та власною логікою підрахунку facets
Залежність від сторонніх компонентів	Висока (платформа + додатки)	Висока (CMS + 10–20 плагінів для повноцінного магазину)	Середня	Низька (явні npm-залежності, фіксовані версії)
Орієнтовні витрати на старті	від $39/міс + 2% комісія	$5–20/міс хостинг + $100–300/рік плагіни	$20–80/міс хостинг (Magento вимогливий)	$5–20/міс хостинг

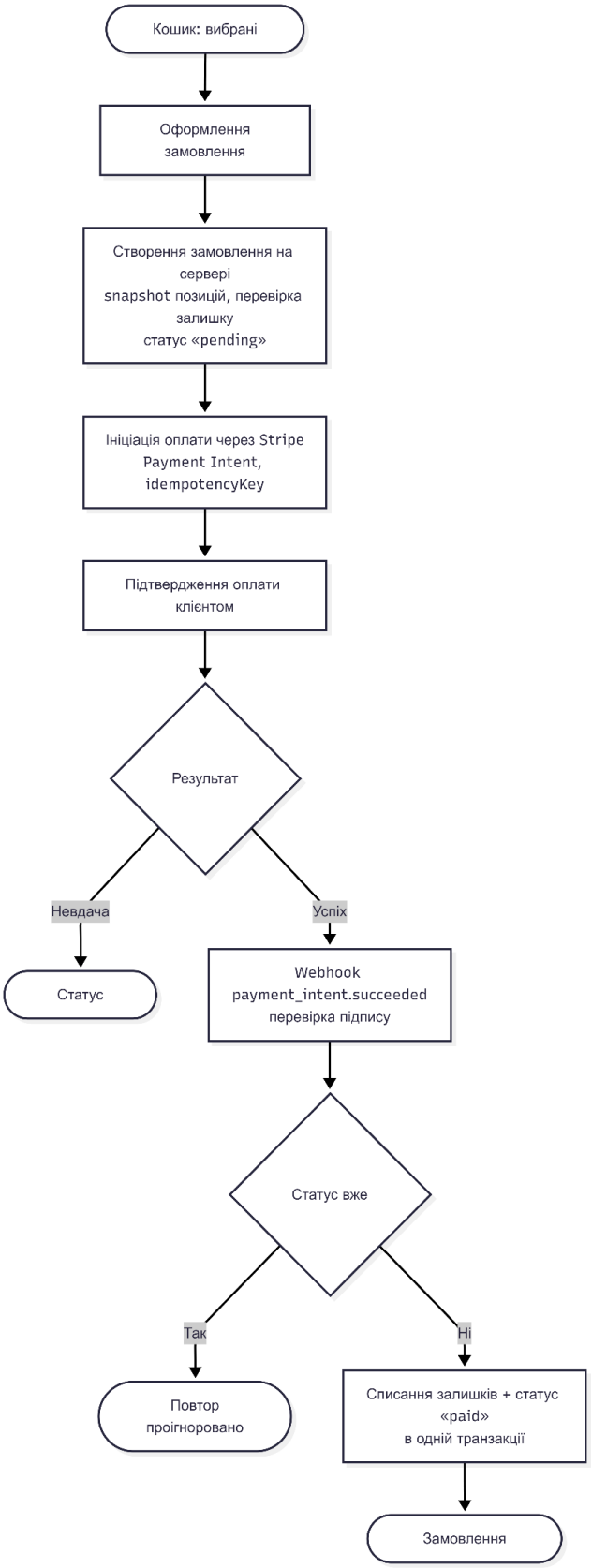

Рисунок. 1.2. Життєвий цикл оформлення замовлення: від кошика до оплати, обробки webhook від Stripe і списання залишків

1.5 Постановка задачі дослідження

Спираючись на викладене вище, сформульовано завдання на розробку власного програмного продукту. Продукт орієнтовано на автоматизацію основних процесів інтернет-магазину: ведення асортименту, відображення каталогу на вітрині, оформлення замовлення та приймання онлайн-оплати.

Назва продукту — вебзастосунок інтернет-магазину для управління асортиментом чоловічого верхнього одягу.

Мета — спроєктувати й реалізувати інформаційну систему e-commerce, яка централізовано веде каталог товарів і варіантів (SKU), підтримує фасетну фільтрацію за атрибутами, контролює складські залишки, формує замовлення і приймає онлайн-оплату з надійною серверною обробкою платіжних подій.

Цільова аудиторія:

- покупці інтернет-магазину, які обирають товари й оформлюють замовлення через вебінтерфейс;
- адміністратори (контент-менеджери), які наповнюють каталог і керують товарами, варіантами, цінами, залишками й зображеннями.

Кінцевий продукт — вебзастосунок із клієнтською вітриною та адміністративним контуром. Доступ до функцій розмежований за ролями: покупець і адміністратор. Покупцеві доступні перегляд каталогу, кошик і оформлення замовлення; адміністратор керує каталогом і довідниками.

Програмний продукт повинен реалізовувати такі функціональні можливості:

1. Ведення бази даних каталогу:

- інформацію про категорії товарів (назва, slug);
- дані про товари (назва, опис, базова ціна, описові поля виробу, належність до категорії, матеріал);
- дані про варіанти товару (SKU, ціна варіанту за потреби, складський залишок);

- атрибути та їх значення (наприклад, розмір, колір) і прив'язку значень до варіантів;
- зображення для варіантів з можливістю впорядкування та формування прев'ю.

2. Представлення каталогу у вітрині:
- перегляд товарів з пагінацією та сортуванням;
- перегляд сторінки товару з вибором варіанту за атрибутами та переглядом зображень;
- фасетна фільтрація за атрибутами (розмір, колір) і характеристиками товару (наприклад, матеріал), із формуванням лічильників значень (facets).

3. Кошик:
- додавання/видалення позицій, зміна кількості;
- збереження стану кошика між сесіями користувача;
- розрахунок проміжної вартості (subtotal) та підсумкової вартості з урахуванням податку й доставки.

4. Оформлення замовлення (checkout):
- введення контактних даних та адреси доставки;
- вибір способу доставки із визначенням вартості й орієнтовних строків;
- створення замовлення із фіксацією (snapshot) основних даних товарних позицій на момент покупки.

5. Онлайн-оплата та обробка платежів:
- інтеграція з платіжним сервісом для створення платіжного наміру (Payment Intent) та завершення оплати на стороні користувача;
- серверна обробка подій оплати (webhook) з оновленням статусу замовлення;
- забезпечення ідемпотентності критичних операцій та коректного списання залишків після успішної оплати.

6. Додаткові можливості:

- функціональність списку бажаного (wishlist) для збереження обраних позицій;
- адміністративні операції керування каталогом (категорії, товари, варіанти, зображення, довідники атрибутів).

Технічні вимоги до вебзастосунку сформульовано як вимірювані характеристики, придатні до перевірки методами програмної інженерії якості (QA/SQE):

- технологічний стек: клієнтська частина — Next.js / React (TypeScript); серверна частина — Node.js / Express.js (TypeScript); зберігання даних — реляційна СУБД PostgreSQL; інтеграція платежів — Stripe;
- продуктивність: час завантаження сторінки каталогу — не більше 2 секунд; час відповіді API при застосуванні фасетного фільтра — не більше 200 мс [4];
- безпека: паролі користувачів зберігаються виключно у вигляді криптографічного хешу за алгоритмом bcrypt; автентифікація реалізована через JWT, що передається у захищених HTTP-only cookie з прапорцями Secure та SameSite;
- надійність та ідемпотентність: критичні операції транзакційного контуру (списання складських залишків, зміна статусу замовлення) виконуються ідемпотентно — повторна доставка події webhook від платіжного шлюзу не призводить до повторного списання залишків або дублювання замовлень.

Очікувані результати від використання вебзастосунку:

- підвищення якості та керованості каталогу товарів і варіантів;
- зменшення кількості помилок у даних (ціни, залишки, варіанти) за рахунок централізованого зберігання;
- покращення користувацького досвіду завдяки фільтрації, коректному відображенню доступних варіантів і прозорому checkout;

забезпечення надійного процесу оформлення замовлення та онлайн-оплати зі списанням залишків після успішного платежу.

Висновки до розділу 1

У першому розділі я розглянув сучасні підходи до побудови та розвитку e-commerce-систем, зосередившись на управлінні асортиментом, де товари мають високу варіативність — за розмірами, кольорами, матеріалами. Видно, що якість роботи магазину значною мірою ґрунтується на правильно організованому каталозі — як системі взаємопов'язаних сутностей «категорії → товари → варіанти» — і на гнучкій моделі атрибутів, яка дозволяє нарощувати асортимент без дублів у даних і без ускладнення логіки вітрини. Окремо я виділив роль фасетної фільтрації: вона покращує навігацію каталогом і дає користувачеві інформативний вибір завдяки лічильникам значень (facets).

Розібрано основні механізми роботи інформаційної системи магазину і визначено, що вона поєднує два контури — адміністративний (керування товарами, варіантами, атрибутами, зображеннями, залишками) і користувацький (перегляд каталогу, фільтрація, кошик, checkout). Транзакційні процеси (оформлення замовлення та онлайн-оплата) потребують узгодженості даних і надійної серверної обробки подій — для коректних змін статусу замовлення і списання складських залишків після підтвердження платежу. Інтеграцію з платіжним сервісом тут варто розглядати як частину цілісного процесу з ідемпотентністю критичних операцій.

Також я зробив огляд наявних підходів до реалізації e-commerce-платформ (SaaS, CMS, headless/модульні) і виділив їхні сильні сторони й обмеження з погляду варіативного каталогу та власної бізнес-логіки. На основі цього аналізу сформовано постановку задачі дослідження й вимоги до програмного продукту: підтримка каталогу з варіантами й атрибутами, фасетна фільтрація, кошик, checkout, інтеграція онлайн-оплати, акуратний облік залишків. Отриманий результат закладає основу для проєктування архітектури застосунку, моделі бази даних і реалізації основних модулів у наступних розділах.

РОЗДІЛ 2
ТЕХНОЛОГІЇ ТА ЗАСОБИ РОЗРОБКИ ІНФОРМАЦІЙНОЇ СИСТЕМИ ІНТЕРНЕТ-МАГАЗИНУ

2.1 Обґрунтування вибору мови програмування TypeScript для розробки веб-застосунку

TypeScript — мова програмування, що добудовує JavaScript статичною типізацією і додатковими засобами опису структури даних. Для вебзастосунку інтернет-магазину це виправданий вибір: у проєкті багато взаємопов'язаних сутностей (товари, варіанти, атрибути, замовлення, доставка, оплата), і дані постійно передаються між клієнтом, сервером і базою. У таких умовах типобезпека помітно знижує кількість помилок інтеграції й робить поведінку системи передбачуванішою [21, 22].

TypeScript застосовано як на клієнті (інтерфейс вітрини, кошик, checkout), так і в серверній логіці та API — там, де потрібно гарантувати коректність даних запитів і відповідей. Далі розглянемо не загальні властивості мови, а конкретний внесок статичної типізації у надійність розробленого застосунку та впорядкування роботи з даними.

1. Моделювання предметної області через типи

- структуру ключових сутностей (product, variant, attribute, order, order_item, shipping method) зафіксовано в інтерфейсах і типах, тож модель «товар → варіант → атрибут» має єдине формальне визначення, спільне для всіх модулів;
- скінченні множини станів — зокрема статуси замовлення (pending, paid, failed) — описано переліченнями (enum) і об'єднаннями літеральних типів, що унеможливлює використання неіснуючого статусу й робить обробку станів вичерпною;
- знімки даних позицій замовлення (snapshot ціни, SKU, кольору, розміру) типізовано окремо від «живих» сутностей каталогу, що на рівні типів розмежовує історичні та актуальні дані.

2. Надійність контрактів і роботи з даними

- типізовані параметри функцій і відповіді API знижують ризик помилок при зміні бекенд-ендпоінтів або структури БД: невідповідність контракту виявляється на етапі компіляції, а не під час виконання;
- схеми валідації вхідних даних (zod) і виведені з них типи (z.infer) дають єдине джерело правди — структура, що перевіряється у рантаймі, і тип, відомий компілятору, не розходяться;
- результати запитів до бази даних мапляться у типізовані об'єкти, тож обробка рядків таблиць (orders, product_variants, order_items) контролюється типами на всьому шляху від SQL до відповіді API;
- узгоджені контракти між шарами полегшують розділення відповідальності між UI, API та бізнес-логікою, а generics дають змогу повторно використовувати утиліти — для пагінації, роботи зі списками та типобезпечних викликів API.

3. Інструменти розробки та безпечний розвиток системи

- інтеграція з IDE (передусім Visual Studio Code) забезпечує автодоповнення за типами, підсвічування помилок і швидку навігацію до визначень типів і функцій;
- рефакторинг — перейменування, витягування функцій, зміна сигнатур — виконується безпечніше саме завдяки типам, що критично для проєкту, який поступово нарощує функціональність;
- налаштування суворості перевірок через tsconfig.json дозволяє підлаштувати рівень контролю під потреби проєкту.

Тож TypeScript у цьому проєкті відіграв роль не синтаксичної надбудови, а засобу забезпечення узгодженості даних між компонентами, API та базою даних: формалізація предметної області в типах, типобезпечні контракти й виведені зі схем валідації типи зменшили кількість помилок інтеграції та спростили підтримку складних користувацьких сценаріїв, таких як фасетна фільтрація й оформлення замовлення з онлайн-оплатою.

2.2 Архітектура та компонентний підхід React для створення інтерактивних інтерфейсів

React — декларативна компонентна бібліотека для побудови інтерфейсів на JavaScript, яку розвиває Facebook (Meta). Її основна ідея — багаторазові UI-компоненти, з яких збираються складні інтерфейси, з акцентом на передбачуваність і повторне використання. Замість докладного переказу внутрішніх механізмів бібліотеки розглянемо, як саме її архітектурні принципи застосовано в розробленому застосунку — у вітрині, фасетному фільтрі, адміністративному каталозі та сценарії оплати [19].

Компонентна композиція та повторне використання. Інтерфейс зібрано з атомарних компонентів, які повторно використовуються в різних частинах застосунку: спільні елементи (Checkbox, Button) задіяні і у фасетному фільтрі, і в адмінських формах, а узагальнений компонент розкривної секції (Section) однаково обслуговує блоки «Sort By», «Size», «Color», «Fabric». Інкапсуляція стану й розмітки в окремих компонентах дозволила нарощувати функціональність вітрини без дублювання коду.

Керування станом через React Hooks. Локальний стан компонентів реалізовано на хуках: керування локальним станом за допомогою useState дозволило ефективно реалізувати динамічне оновлення лічильників значень (facets) у фасетному фільтрі без перезавантаження сторінки каталогу — при зміні набору фільтрів перелік доступних значень і їх кількість оновлюються миттєво, а опції з нульовим лічильником автоматично блокуються. Для уникнення зайвих обчислень похідні дані (наприклад, перелік застосованих фільтрів-чипів та відсортовані розміри) мемоізуються через useMemo і перераховуються лише при зміні параметрів запиту. Складнішу логіку адміністративного каталогу винесено у власний хук (useProductList), який інкапсулює стан форм, варіантів, чернеток і діалогів та надає компонентам готовий, упорядкований інтерфейс — це відокремило бізнес-логіку керування каталогом від подання.

Декларативне оновлення інтерфейсу. Компоненти описують бажаний стан інтерфейсу, а React самостійно оновлює лише змінені його частини, тому часте перерахування лічильників фасетів не спричиняє повного перемальовування сторінки. Інтерфейс завжди відображає поточний стан даних: застосовані фільтри показуються у вигляді чипів, недоступні значення блокуються, а кнопка оплати під час обробки платежу переходить у стан «Processing…» і блокується від повторного натискання.

Інтеграція зовнішніх сервісів через хуки. Хуковий підхід спростив підключення сторонніх бібліотек: у формі оплати через хуки useStripe і useElements отримано доступ до платіжного контексту Stripe і керовано станом надсилання (isSubmitting, error), а синхронізацію стану фасетного фільтра з URL реалізовано хуками навігації Next.js (useSearchParams, useRouter, usePathname), завдяки чому застосовані фільтри зберігаються в адресі сторінки й піддаються поширенню через посилання.

Сучасний React будується на функціональних компонентах і хуках: код виходить лаконічнішим за класовий, простіше пишеться й тестується, а власні хуки дозволяють повторно використовувати логіку. При цьому React не нав'язує єдиного способу керування станом — у проєкті виявилося достатнім поєднання локального стану компонентів, стану в URL та власних хуків, без потреби в зовнішніх менеджерах станів (Redux, Zustand, Jotai) [20].

2.3 Використання Next.js як фреймворку для побудови клієнтської частини інтернет-магазину

Next.js я обрав основою фронтенду саме тому, що для інтернет-магазину критичні два моменти — швидке завантаження сторінок каталогу і зручна навігація між розділами. У системі є головна сторінка, каталог, картка товару, кошик і поетапне оформлення замовлення. Для такого набору сценаріїв потрібен фреймворк, який добре дружить і з рендерингом на сервері, і з інтерактивною роботою в браузері, та ще й дозволяє чітко структурувати код за сторінками і модулями.

Найвагоміший плюс Next.js у цьому проєкті — поєднання сторінкової архітектури з можливістю окремо оптимізувати критичні сценарії. Каталог і картка товару повинні швидко відображатися при першому відкритті й бути доступними для індексації, тоді як кошик і checkout — динамічні процеси, де важлива висока інтерактивність, робота з формами, збереження стану і миттєва реакція на дії користувача.

файлова структура задає маршрутизацію, і це спрощує підтримку маршруту «каталог — сторінка товару — кошик — checkout»;

1. Зручна організація маршрутизації
- підтримка динамічних маршрутів — без неї не зробити сторінку товару за slug;
- для сторінок, де важливі швидкий перший рендер і SEO, використовується серверне формування HTML;
2. Підтримка різних режимів рендерингу
- для інтерактивних сценаріїв (кошик, вибір способу доставки, оплата) працює клієнтська логіка з реактивною зміною інтерфейсу без перезавантажень;
- Для інтерактивних сценаріїв (кошик, вибір способу доставки, оплата) працює клієнтська логіка — інтерфейс перебудовується реактивно, без перезавантаження сторінок.
3. Оптимізація продуктивності у критичних сценаріях
- автоматичне розділення коду зменшує обсяг завантаження при першому відкритті сайту;
- перехід між сторінками прискорюється через попередню підготовку ресурсів і маршруту;
- для e-commerce це принципово: затримки на каталозі або checkout одразу б'ють по конверсії.
4. Сумісність з екосистемою проєкту
- Next.js природно поєднується з React і компонентним підходом — інтерфейс легко збирається з повторно використовуваних блоків;

- підтримка TypeScript полегшує роботу з типами даних каталогу, замовлень і платежів і знижує ризик розсинхрону між UI та API.

Тож Next.js у моєму проєкті — основа клієнтської вітрини магазину: він підтримує і статичні сторінки, і інтерактивне оформлення замовлення, і дозволяє побудувати продуктивний інтерфейс, орієнтований на реальні сценарії користувача [3].

2.4 Реляційна база даних PostgreSQL: модель даних, доступ та транзакційна надійність

Для серверної частини магазину обрано реляційну СУБД PostgreSQL із прямим доступом через параметризовані SQL-запити. Такий вибір для e-commerce виправданий: PostgreSQL одночасно закриває три ключові потреби проєкту — узгодженість даних, опис складних зв'язків між сутностями та транзакційну надійність критичних операцій, — а прямий доступ дає тонкий контроль над структурою запитів і продуктивністю вибірок. У системі база даних виконує роль єдиного надійного джерела даних про каталог, варіанти (SKU), атрибути, замовлення, доставку та оплату [6, 10].

1. Реляційна модель даних і складні зв'язки. Каталог природно описується реляційними сутностями та зв'язками між ними: ієрархія «категорії → товари → варіанти» доповнюється довідниками атрибутів. Значення атрибутів (наприклад, колір і розмір) зберігаються в окремих таблицях-довідниках і пов'язуються з варіантами через сполучні таблиці, тож кількість значень можна нарощувати без зміни структури таблиць і без дублювання даних. Така модель спрощує і розширення асортименту, і подальшу роботу фасетної фільтрації [1, 5, 9].
2. Доступ через пул підключень і параметризовані запити. Взаємодія з базою побудована на пулі підключень, що дозволяє ефективно обслуговувати паралельні запити без постійного відкриття нових з'єднань. Усі запити виконуються параметризованими: значення з HTTP-запиту не потрапляють безпосередньо в текст SQL, що унеможливлює

SQL-ін'єкції та підвищує безпеку взаємодії з базою. Додатково безпеку забезпечують розмежування прав доступу до бази та захищене підключення.

3. Агрегатні вибірки та фасетна фільтрація. Для вітрини на боці бази формуються агреговані вибірки: список товарів із підрахунком доступних варіантів і залишків, переліком наявних кольорів та прев'ю-зображеннями. Відбір товарів за атрибутами реалізовано запитами, які коректно обробляють кілька умов одночасно (колір, розмір, матеріал, наявність), а для інтерфейсу окремо формується вибірка значень фасетів із лічильниками — щоб користувач бачив доступні опції та кількість товарів для кожної з них.
4. Транзакційна узгодженість і конкурентний доступ. Критичні операції виконуються як єдина узгоджена дія в межах транзакції. Створення замовлення є атомарним: дані замовлення та його позицій записуються узгоджено, причому в позиціях фіксуються snapshot-значення (назва товару, SKU, колір, розмір, зображення, ціна), що зберігає історичну коректність даних незалежно від подальших змін каталогу. Перед прийняттям замовлення перевіряється достатність складського залишку. Для безпечної одночасної покупки тих самих позицій застосовується блокування рядка замовлення (SELECT … FOR UPDATE), а вбудовані механізми журналювання змін (WAL) забезпечують коректне відновлення стану бази після збоїв [11].
5. Ідемпотентне узгодження оплати та списання залишків. Після підтвердження платежу сервер у межах однієї транзакції оновлює статус замовлення та списує залишки за відповідними варіантами. Логіку побудовано ідемпотентно: перед списанням перевіряється, чи замовлення вже не має статусу «paid», тож повторна або відкладена доставка платіжної події не призводить до повторного списання залишків. Списання виконується лише за умови достатнього залишку, інакше транзакція відкочується [16].

6. Продуктивність і оптимізація запитів. PostgreSQL надає індекси та оптимізатор запитів, що дає простір для прискорення типових для e-commerce вибірок із агрегаціями — підрахунку доступних значень фільтрів і обчислення залишків. У міру зростання обсягів даних продуктивність можна додатково підвищувати тонким налаштуванням індексації та оптимізацією складних вибірок [7].

Отже, PostgreSQL у проєкті виконує роль центрального сховища даних: забезпечує транзакційну надійність замовлень і оплат, дозволяє гнучко описати модель каталогу з варіантами й атрибутами, підтримує фасетну фільтрацію через агрегатні вибірки та залишає простір для оптимізації запитів вітрини під зростаюче навантаження.

2.5 Реалізація автентифікації та авторизації у веб-застосунку (JWT, cookies, ролі доступу)

Для інтернет-магазину контроль доступу обов'язковий. Покупець має могти робити свої дії — оформити замовлення, переглянути свої покупки, зберегти адресу доставки. Натомість адміністративні операції (керування каталогом, редагування товарів і варіантів, завантаження зображень, зміна статусів замовлень) повинні бути доступні лише авторизованим користувачам із відповідною роллю. Тому в проєкті я реалізував автентифікацію користувача та авторизацію за ролями.

У цьому веб-застосунку я використав підхід на JWT (JSON Web Token), що зберігається в cookie. При вході користувач передає логін і пароль, сервер перевіряє облікові дані і — якщо все коректно — формує підписаний токен з ідентифікатором користувача та роллю. Далі браузер автоматично передає cookie з токеном у запитах до захищених ендпоінтів. Так сервер ідентифікує користувача, не зберігаючи сесій у себе, а сам механізм перевірки доступу залишається простим.

Основні механізми реалізації:

1. Вхід користувача та формування токена

- після отримання даних входу сервер шукає користувача в базі за email;
- пароль перевіряється проти хешу (bcrypt) — у відкритому вигляді він ніде не лежить;
- у разі успішної перевірки створюється JWT з ключовими полями (userId, role) і обмеженим часом дії [14];
- токен записується в httpOnly cookie — це знижує ризик доступу до нього з клієнтського JavaScript.

2. Перевірка автентифікації на бекенді

- для захищених маршрутів спрацьовує middleware, який читає токен із cookie;
- токен перевіряється секретним ключем — у разі валідності сервер дістає userId і роль;
- дані користувача (id, роль) додаються до контексту запиту, щоб контролери та репозиторії могли працювати від його імені.

3. Авторизація та розмежування прав

- для адміністративних операцій передбачено окрему перевірку ролі (admin);
- якщо роль не відповідає вимогам, сервер повертає Forbidden;
- Завдяки цьому користувацький функціонал (замовлення, адреси) відокремлений від адміністративного (керування каталогом, зміна статусів замовлень).

4. Захист транзакційних сценаріїв

- У запитах, що стосуються замовлень, ідентифікатор користувача береться з токена, а не з тіла запиту: так знижується ризик підміни userId та доступу до чужих даних.
- захищені ендпоінти (orders, addresses, wish-lists) доступні лише після успішної автентифікації.

Тож реалізована система автентифікації й авторизації забезпечує контроль доступу до функцій магазину, розділення ролей і захист критичних операцій. Для проєкту з поділом на користувацький і адміністративний

контури такого підходу достатньо, і він узгоджується з вимогами безпечної обробки даних у веб-застосунках.

2.6 Оптимізація та підхід до розгортання веб-застосунку

Розробка інтернет-магазину не обмежується каталогом, кошиком і оплатою: щоб застосунок можна було ввести в експлуатацію, важливі також організація збірки й запуску, керованість конфігурації та оптимізація швидкодії клієнтської частини. Тому в проєкті, поряд із функціональністю, враховано принципи розділення компонентів і підготовки системи до розгортання.

Двосервісна архітектура. Систему розділено на два незалежні сервіси — клієнтську частину (Next.js) і серверний API (Express), які запускаються як окремі процеси. Таке розділення дозволяє незалежно конфігурувати, оновлювати й масштабувати кожен сервіс: інтерфейс можна оновлювати без змін у серверній логіці й навпаки. У локальному середовищі обидва процеси запускаються паралельно (через утиліту concurrently), що відтворює ту саму схему взаємодії «клієнт ↔ API ↔ база даних», що й у цільовому середовищі.

Зовнішні керовані сервіси. База даних і файлове сховище винесені в окремі керовані компоненти. PostgreSQL підключається за рядком з'єднання із захищеним каналом (SSL), а зображення товарів зберігаються в об'єктному сховищі (Supabase Storage); посилання на сховище налаштовано і на боці Next.js (оптимізація зображень), і на боці API. Завдяки цьому стан системи (дані та медіафайли) відокремлений від коду застосунку.

Конфігурація через змінні середовища. Параметри, що залежать від середовища й містять секрети (рядок підключення до бази, ключі Stripe, секрет вебхука, налаштування автентифікації), винесено у змінні середовища і не зберігаються у вихідному коді. Це дозволяє використовувати різні набори налаштувань для локальної розробки й цільового середовища без зміни коду та відповідає базовим вимогам безпеки.

Збірка та запуск. Для кожного сервісу визначено стандартні сценарії збірки та запуску: клієнтська частина проходить етап збірки (next build) із подальшим запуском (next start), серверна частина запускається як Node.js-процес. Централізовані скрипти на рівні монорепозиторію спрощують встановлення залежностей і одночасний запуск обох сервісів, зменшуючи кількість ручних кроків.

Оптимізація клієнтської частини. Швидкодію вітрини забезпечують вбудовані механізми Next.js: автоматичне розділення коду (code splitting) зменшує обсяг початкового завантаження, навігація між каталогом і checkout відбувається без повного перезавантаження сторінок, а для зображень застосовано вбудовану оптимізацію. Це покращує час першого відображення та загальне відчуття швидкодії.

Підготовка до розгортання та напрями подальшого розвитку. Описана архітектура є придатною до розгортання (deployment-ready): незалежні сервіси, винесена конфігурація та зовнішні керовані БД і сховище дозволяють розмістити клієнтську частину на платформі для Next.js-застосунків (наприклад, Vercel), а серверний API — на хостингу для Node.js-сервісів (наприклад, Render або Railway), зберігши базу даних і сховище як керовані хмарні сервіси. Подальшим напрямом є контейнеризація сервісів (Docker) для уніфікації середовищ і впровадження автоматизованого CI/CD-пайплайна (наприклад, на основі GitHub Actions) для автоматичної збірки, перевірки та розгортання при оновленні коду.

Висновки до розділу 2

У другому розділі я обґрунтував вибір технологій і засобів розробки веб-застосунку інтернет-магазину і показав, що обраний стек відповідає вимогам до продуктивності, підтримуваності коду й надійності транзакційних сценаріїв. Перебрано як клієнтську частину (інтерфейс каталогу, кошика, checkout), так і серверну (API, робота з базою даних, обробка платежів) — це дає змогу реалізувати повний цикл роботи e-commerce-системи.

TypeScript у проєкті — це основа типобезпечної розробки: він зменшує кількість помилок у роботі з даними каталогу, замовлень і доставки та полегшує підтримку коду під час розширення функціональності. Компонентний підхід на React дозволяє будувати інтерфейс із повторно використовуваних елементів і зручно реалізовувати інтерактивні сценарії — керування кошиком, вибір варіантів товару, роботу з формами checkout.

Next.js застосовано як фреймворк для клієнтської вітрини з огляду на швидке перше відображення сторінок і зручну маршрутизацію. Для каталогу та сторінок товарів це принципово, бо саме швидкодія інтерфейсу й зрозуміла навігація формують користувацький досвід. Поряд із цим я описав підхід до оптимізації та розгортання застосунку — він ґрунтується на розділенні фронтенду та бекенду як окремих компонентів і дозволяє розвивати кожну частину незалежно.

Для зберігання даних обрано PostgreSQL — надійну реляційну СУБД з підтримкою транзакцій, що дає змогу будувати узгоджені операції для критичних бізнес-процесів. Доступ до БД у проєкті організовано через параметризовані SQL-запити: так я утримую під контролем продуктивність вибірок каталогу, фасетну фільтрацію і коректність операцій створення замовлення та списання залишків. Окремо описано автентифікацію й авторизацію на JWT і cookie — на цьому ґрунтується розмежування доступу між користувацьким та адміністративним контурами і захист критичних ендпоінтів.

Тож за результатами розділу 2 можна сказати, що вибраний стек технологій створює всі необхідні передумови для e-commerce-застосунку: підтримуваність коду, продуктивну клієнтську частину, надійний доступ до даних, безпечний доступ і стабільність транзакцій, пов'язаних із замовленнями та оплатою. Це й закладає основу для опису проєктування архітектури та реалізації основних модулів системи в наступному розділі.

РОЗДІЛ 3
ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ІНТЕРНЕТ-МАГАЗИНУ

3.1 Структура веб-застосунку інтернет-магазину

Архітектура мого веб-застосунку складається з чотирьох основних компонентів (рис. 3.1): клієнт (Next.js), сервер (Express), база даних (PostgreSQL) і платіжний

сервіс (Stripe).

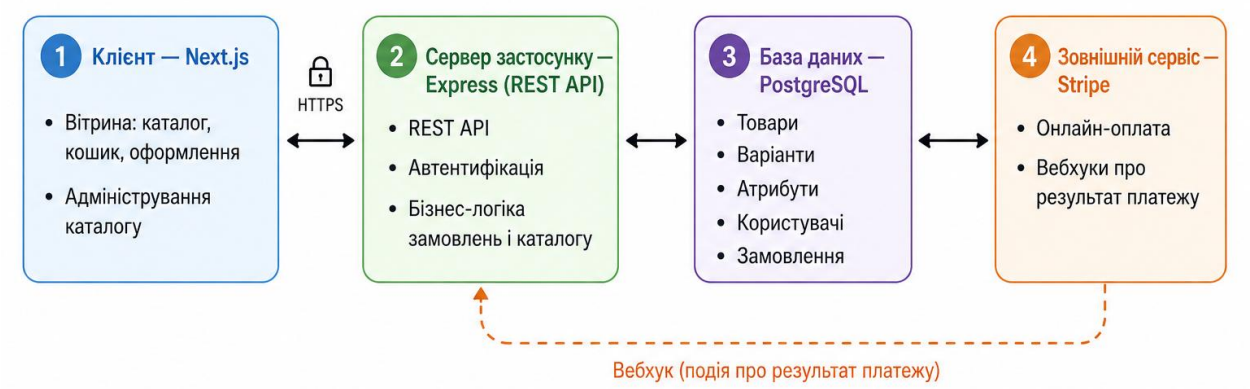

Рисунок 3.1. Архітектура системи інтернет-магазину (Next.js, Express, PostgreSQL, Stripe).

3.1.1 Загальна інформація про структуру веб-застосунку

Веб-застосунок інтернет-магазину розпадається на дві частини — клієнтську (вітрину) і серверну (API), які спілкуються між собою через HTTP-запити. Це дозволяє чітко розділити відповідальність між інтерфейсом і бізнес-логікою, а заодно тримати запас на масштабування системи в міру зростання каталогу і кількості користувачів [29].

Клієнтська частина обслуговує користувацькі сценарії — перегляд каталогу, вибір варіантів товарів, кошик, оформлення замовлення. Серверна частина веде роботу з базою даних, обслуговує ендпоінти каталогу та замовлень і відповідає за інтеграцію онлайн-оплати й обробку платіжних

подій. База даних — центральний вузол системи, в ній лежать товари, варіанти, атрибути, зображення, користувачі та замовлення.

Основні елементи структури застосунку:

- Головна сторінка — стартова точка з первинною навігацією і виходом на каталог.
- Сторінка каталогу товарів — список товарів із сортуванням, пагінацією та фасетною фільтрацією за атрибутами.
- Сторінка товару — детальна інформація, вибір варіанта (наприклад, розмір і колір), перегляд зображень.
- Кошик — обрані позиції, зміна кількості, видалення, перегляд проміжної вартості.
- Checkout — поетапне оформлення замовлення: контактні дані й адреса, вибір способу доставки, створення замовлення, перехід до оплати.
- Сторінки результату оплати — повідомлення про успішне завершення або помилку.
- Адміністративний контур (сторінка керування каталогом) — інструменти для створення й редагування категорій, товарів, варіантів, атрибутів і зображень.

Тут варто додати, що структура застосунку враховує адаптивне відображення на різних пристроях. У користувацькому контурі акцент я зробив на зручну навігацію каталогом і мінімум кроків при оформленні замовлення; в адмінському — на швидкість операцій із каталогом, цілісність даних і можливість оперативно оновлювати асортимент.

Тож структура веб-застосунку охоплює повний цикл роботи системи — від ведення каталогу й показу товарів на вітрині до оформлення замовлення та обробки онлайн-оплати — і слугує основою для подальшого детального опису модулів.

3.1.2 Навігаційна структура та рівні доступу користувачів

Для зручної навігації у веб-застосунку магазину я будую структуру сторінок навколо основних сценаріїв користувача: перегляд каталогу, вибір товару, формування кошика, оформлення замовлення. Окремо існує адміністративний контур для керування асортиментом. Так навігація логічно розпадається на дві частини — користувацьку (вітрина) та адміністративну.

Система розмежовує доступ за ролями. У межах проєкту я реалізував дві основні ролі: покупець (користувач вітрини) і адміністратор (керує каталогом і має доступ до адміністративних операцій).

CUSTOMER (Покупець)

перегляд сторінок каталогу — список товарів із фільтрацією, сортуванням і пагінацією;

сторінка товару — опис, вибір варіанта (розмір, колір), перегляд зображень;

кошик — додавання і видалення товарів, зміна кількості, перегляд підсумкових сум;

оформлення замовлення (checkout) — контактні дані й адреса, вибір способу доставки, перехід до оплати;

оплата — завершення платежу й отримання повідомлення про результат (success/error);

додатково — список бажаного (wishlist) для збереження обраних позицій.

ADMIN (Адміністратор системи)

керування категоріями: створення, редагування й видалення;

керування товарами: створення й редагування товарів, керування описом, цінами, матеріалами;

керування варіантами (SKU): створення, редагування, облік залишків, прив'язка атрибутів;

робота із зображеннями: завантаження, видалення й упорядкування для кожного варіанта;

робота із замовленнями: перегляд замовлень і зміна їх статусу (як адміністративна операція).

Авторизація і контроль доступу реалізовані на сервері. Після входу формується токен, який передається в cookie і відкриває доступ до захищених ендпоінтів. Для адміністративних маршрутів дописана окрема перевірка ролі — щоб критичні операції могли виконати тільки користувачі з правами адміністратора.

3.1.3 Наповнення (контент)

Каталог наповнює адміністратор системи. До контенту тут належать категорії, товари (з описами й додатковими полями), матеріали, варіанти (SKU) з атрибутами (наприклад, розмір і колір), ціни, складські залишки та зображення. Адмін створює і редагує всі ці сутності через адміністративний інтерфейс, а зміни одразу віддзеркалюються у вітрині (рис. 3.1). Тут є нюанс: коректність наповнення безпосередньо впливає на користувацькі сценарії — фасетна фільтрація, доступні варіанти, прев'ю-зображення в каталозі й облік наявності залежать від того, як заповнено дані. Тому при наповненні забезпечується узгодженість між сутностями (категорія — товар — варіант — атрибути — зображення) і контролюю унікальність ключових ідентифікаторів (slug, SKU).

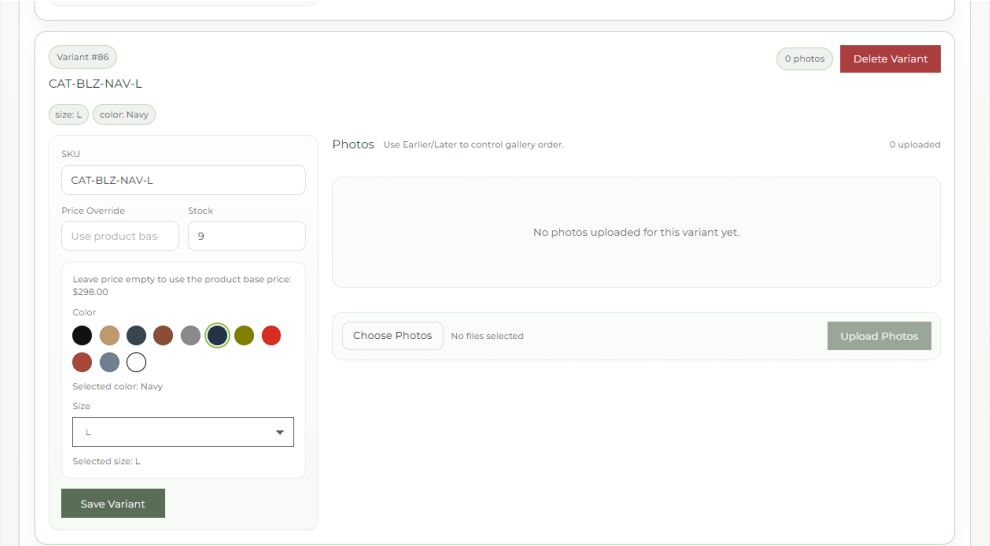

Рисунок 3.1. Інтерфейс редагування варіанту товару (SKU, атрибути, залишки, завантаження фото)

3.1.4 Дизайн

Дизайн веб-застосунку магазину витриманий у стриманому мінімалістичному стилі з акцентом на читабельність і зручну роботу з каталогом. Основну увагу я приділив ключовим сценаріям e-commerce: швидко переглянути список товарів, перейти на сторінку товару, обрати варіант (розмір, колір), попрацювати з кошиком, пройти етапи checkout. Інтерфейс будується на компонентному підході — це утримує елементи керування в одному стилі, дає повторне використання UI-блоків і передбачувану поведінку сторінок.

Адаптивність інтерфейсу — обов'язкова вимога: значна частина користувачів купує з мобільних. Тому дизайн враховує коректне відображення на різних розмірах екранів — від смартфонів до настільних комп'ютерів. Для мобільних сценаріїв я зробив зручні елементи вибору варіанта товару, компактний кошик і чітку послідовність кроків checkout.

Типографіку і візуальну ієрархію підібрано так, щоб користувач швидко зчитував інформацію про товар (назва, ціна, доступні варіанти, наявність) і без помилок заповнював форми checkout. Елементи сторінок розташовані за принципом інтуїтивної навігації: головна інформація і дії — «додати в кошик»,

«перейти до оплати» — потрапляють у зону першочергової уваги, а допоміжне винесене в другорядні блоки.

Дизайн у цьому проєкті тісно пов'язаний із технічною реалізацією. Клієнтська частина побудована на Next.js і React, тож сторінкова структура та інтерактивні елементи інтерфейсу збираються природно. Компонентний підхід полегшує підтримку консистентності стилів і дозволяє розширювати інтерфейс без дублювання коду.

3.2 Проектування та створення бази даних інформаційної системи інтернет-магазину

На основі аналізу функціональних вимог і сценаріїв використання системи (діаграма прецедентів — рис. 3.2) виокремлено дві основні ролі: Клієнт (перегляд каталогу, кошик, оформлення замовлення, оплата, обліковий запис, список бажаного) та Адміністратор (керування категоріями, товарами, варіантами, зображеннями). Для підтримки цих сценаріїв я спроєктував реляційну базу даних у схемі ecommerce СУБД PostgreSQL.

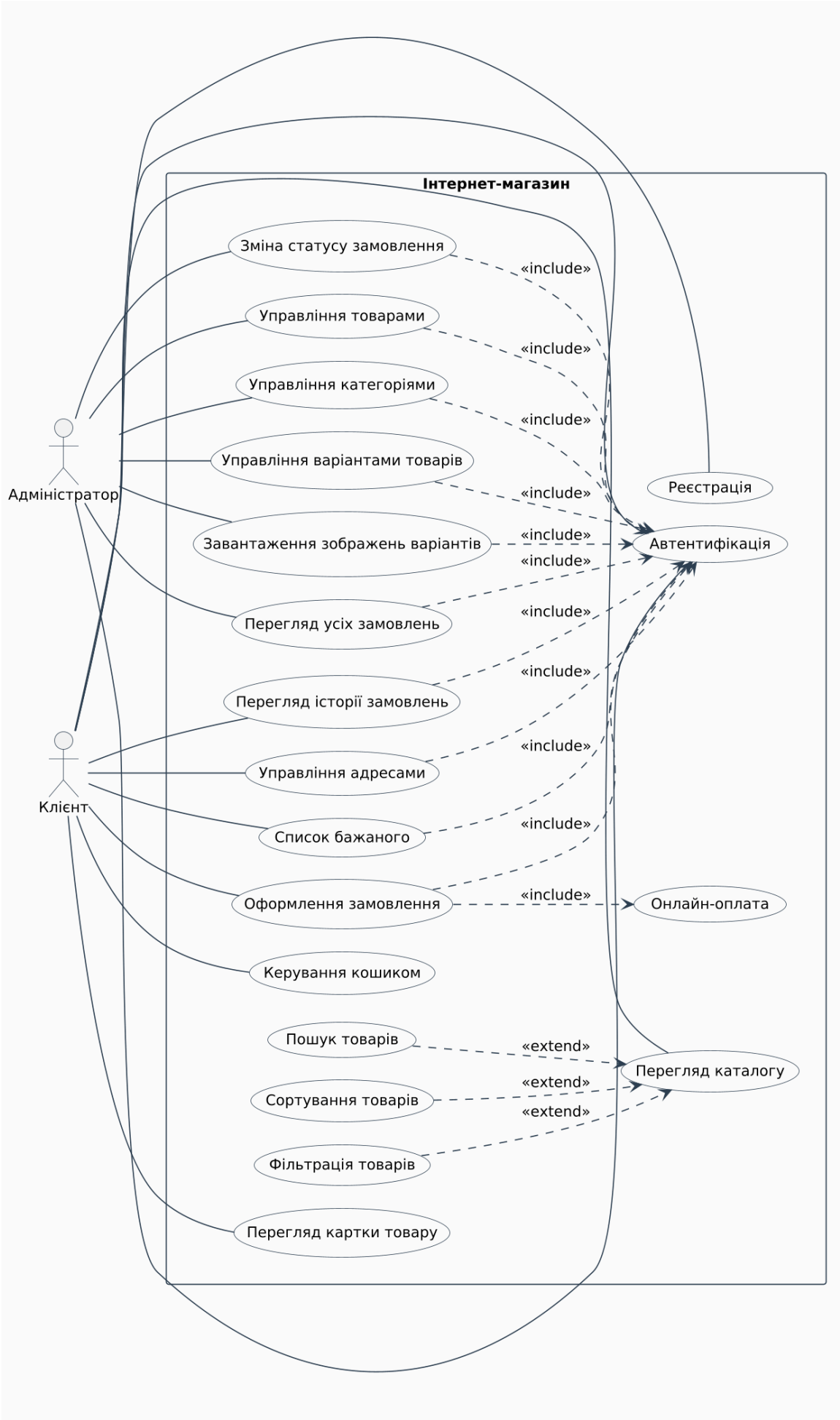
Інтернет-магазин
Адміністратор
Клієнт
Зміна статусу замовлення
Управління товарами
Управління категоріями
Управління варіантами товарів
Завантаження зображень варіантів
Перегляд усіх замовлень
Перегляд історії замовлень
Управління адресами
Список бажаного
Оформлення замовлення
Керування кошиком
Пошук товарів
Сортування товарів
Фільтрація товарів
Перегляд картки товару
Реєстрація
Автентифікація
Онлайн-оплата
Перегляд каталогу
«include»
«extend»

Рисунок 3.2. Діаграма прецедентів інформаційної системи інтернет-магазину (ролі «Клієнт» та «Адміністратор»).

Замість того щоб під кожен атрибут товару заводити окрему колонку, для варіантів (розмір, колір тощо) я застосував модель Entity-Attribute-Value (EAV): довідник атрибутів (attributes), словник значень (attribute_values) і зв'язкова таблиця variant_attribute_values, яка зчеплює варіант SKU з конкретними значеннями. Це дозволяє розширювати набір характеристик без зміни структури таблиці варіантів. Загальний фрагмент зв'язків «товар — варіант — атрибут — значення» наведено на рис. 3.3.

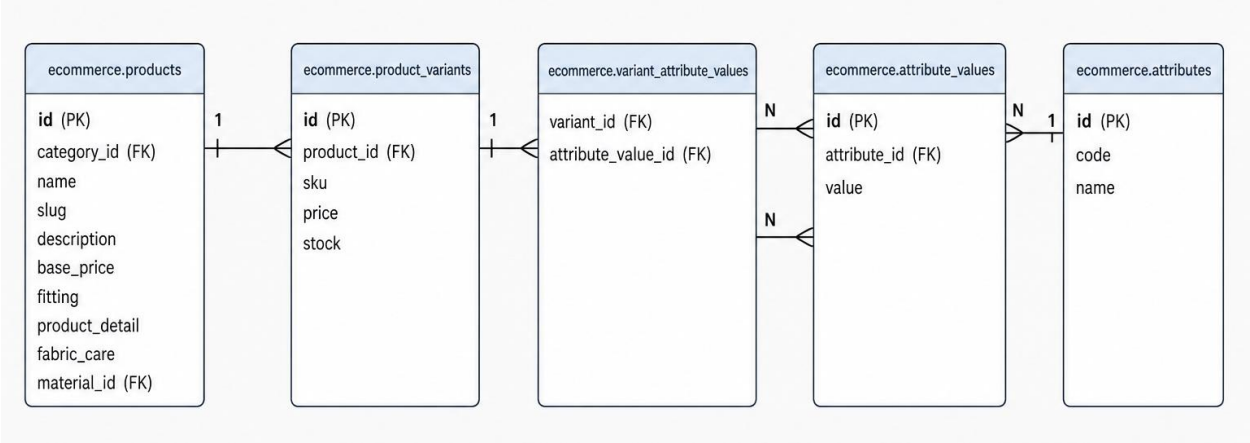

Рисунок 3.3. ER-діаграма фрагмента бази даних: модель EAV каталогу

3.3 Проектування серверної частини системи інтернет-магазину

Серверна частина реалізована окремим застосунком на Node.js та Express. Клієнт (Next.js) звертається до неї по REST API: всі операції з каталогом, замовленнями й обліковими записами проходять через HTTP-запити з JSON. Доступ до PostgreSQL виконується з репозиторіїв — там зібрані SQL-запити, транзакції для замовлень і перевірка прав через middleware [2, 18].

Перед обробниками маршрутів підключено CORS (дозволені адреси фронтенду, credentials: true для cookie), JSON-парсер і cookie-parser для JWT. Окремо налаштований маршрут вебхука Stripe з сирим тілом запиту: підпис події перевіряється до парсингу JSON.

3.3.1 Основні функції REST API

Серверну частину інформаційної системи інтернет-магазину я зібрав як окремий застосунок на Node.js з використанням фреймворка Express. Логіка реалізована у вигляді REST API, який клієнт (Next.js) викликає через HTTP-запити з cookie-автентифікацією. Така архітектура з розділенням клієнта і сервера дає змогу незалежно розвивати інтерфейс і бізнес-логіку та розгортати їх як окремі компоненти [12, 13, 17].

API згруповано за ресурсами предметної області: автентифікація, каталог товарів, фільтри, варіанти та зображення, замовлення й оплата, довідники (категорії, матеріали, способи доставки), адреси користувачів, список бажаного. Публічні методи (перегляд каталогу, фільтри, картка товару) доступні без автентифікації; операції з замовленнями і списком бажаного потребують входу, а адміністративні маршрути захищені перевіркою ролі admin через окремий middleware. Основні маршрути REST API наведено в табл. 3.1.

Таблиця 3.1

Основні функції REST API серверної частини

Маршрут	**Опис**	**Метод**	**Відповідь**
/auth/register	Реєстрація	POST	Користувач, JWT у cookie
/auth/login	Вхід	POST	JWT у cookie
/auth/google	Вхід через Google	POST	JWT у cookie
/products	Список товарів (фільтри, сторінки)	GET	Масив товарів
/products/:slug	Картка товару з варіантами	GET	Об’єкт товару
/filters	Фасети для фільтрів	GET	Кольори, розміри, матеріали, counts
/categories	Категорії	GET	Масив категорій
/materials	Матеріали (тканини)	GET	Масив
/wish-lists	Список бажаного	GET, POST	Масив / запис
/wish-lists/:variantId	Видалити з wish list	DELETE	—
/addresses	Адреса користувача	GET, PUT	Об’єкт адреси
/shipping-methods	Способи доставки	GET	Масив методів

/orders	Створити замовлення	POST	Замовлення
/orders	Мої замовлення	GET	Масив
/orders/:orderId/items	Замовлення з позиціями	GET	Замовлення + items
/orders/:orderId/create-payment-intent	Оплата Stripe	POST	client_secret
/webhook	Подія Stripe (підтвердження оплати)	POST	{ received: true }
/categories	Створити категорію	POST	Категорія
/categories/:slug	Змінити / видалити категорію	PATCH, DELETE	Категорія
/products	Створити товар	POST	Товар
/products/:slug	Змінити / видалити товар	PATCH, DELETE	Товар
/variants	Створити варіант (SKU, атрибути)	POST	Варіант
/variants/:variantId	Змінити / видалити варіант	PATCH, DELETE	Варіант
/variant-images/:variantId/upload	Завантажити фото варіанту	POST	Зображення
/attribute-values	Додати значення атрибута	POST	Значення
/orders/admin	Список усіх замовлень	GET	Масив
/orders/admin/:orderId/items	Замовлення з позиціями	GET	Об’єкт
/orders/:orderId/status	Змінити статус	PATCH	Замовлення

3.3.2 Структура серверної частини

Підсумок такий: REST API закриває потреби клієнта в усіх основних сценаріях — від перегляду каталогу та оформлення замовлення до адміністративних операцій із каталогом і керування статусом замовлень.

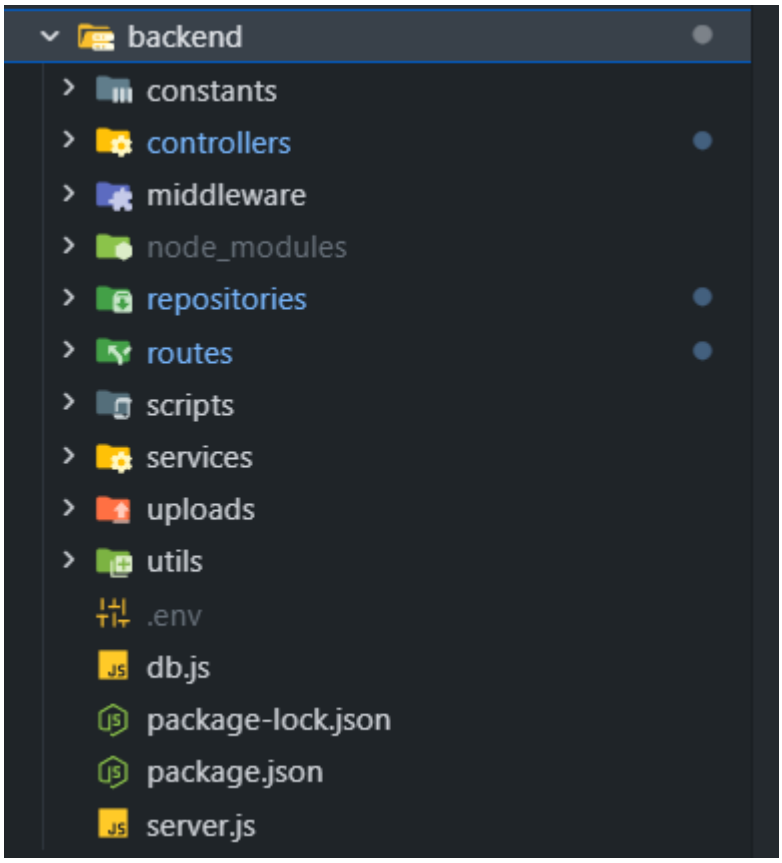

Рисунок. 3.4 Структура каталогів backend

3.3.3 Взаємодія серверної частини з базою даних

Після проєктування схеми ecommerce (розд. 3.2) залишалося визначити, як саме серверна частина спілкуватиметься з PostgreSQL. У реалізації я не став ставити окремий ORM-шар: натомість використано репозиторії з прямими SQL-запитами через драйвер pg. Так зручно гнучко будувати складні вибірки для фасетної фільтрації та EAV-моделі і явно контролювати транзакції при роботі із замовленнями.

Розподіл відповідальності між компонентами шару даних наведено в табл. 3.2.

Таблиця 3.2

Компоненти доступу до PostgreSQL та їх роль

Компонент	Роль
db.js (Pool)	Підключення до PostgreSQL через DATABASE_URL та повторне використання з’єднань

repositories/*.repository.js	SQL-запити до таблиць ecommerce.*, повернення рядків або JSON
controllers/*.controller.js	Валідація даних (Zod), виклик репозиторіїв, формування HTTP-відповіді
middleware/authMiddleware.js	JWT-автентифікація та ідентифікація користувача

Усі звернення до бази даних виконуються параметризованими запитами ($1, $2 і так далі), тобто значення з HTTP-запитів не потрапляють напряму в SQL-рядок. Це знижує ризик SQL-ін'єкцій і піднімає загальний рівень безпеки системи.

Сценарій 1 — читання каталогу з фільтрами. Клієнт передає в query-параметрах значення кольору, розміру, матеріалу й категорії. Утиліта utils/filters.js нормалізує їх у структуру умов для SQL. Репозиторій filters.repository.js рахує фасети (кількість варіантів для кожного значення атрибута), а product.repository.js формує саму вибірку товарів через з'єднання product_variants → variant_attribute_values → attribute_values → attributes. Контролер далі складає JSON-відповідь для клієнта. У цьому сценарії прямі SQL-запити виявилися зручнішими за ORM саме через складну фільтрацію і EAV-зв'язки.

Сценарій 2 — створення замовлення. На цьому етапі треба перевірити наявність товарних варіантів, зафіксувати знімки ціни й атрибутів у order_items, порахувати підсумкову суму з податком і доставкою і зберегти адресу користувача в таблиці orders. У orders.repository.js це реалізовано як єдиний транзакційний блок: BEGIN → перевірка залишків → вставка позицій → створення замовлення → COMMIT; якщо щось пішло не так — ROLLBACK. Після підтвердження оплати Stripe надсилає webhook на сервер, де в окремій транзакції оновлюється статус замовлення і зменшуються складські залишки — це й знімає неузгодженості в даних.

Загальна схема цих двох сценаріїв — від HTTP-запиту до таблиць PostgreSQL без проміжного ORM — наведена на рис. 3.5.

Рисунок. 3.5 Потік обробки запиту з зверненням до PostgreSQL (контролер → репозиторій → СУБД)

3.4 Інтеграція платіжної системи Stripe

Прийом онлайн-оплати реалізовано через Stripe із механізмом Payment Intent. Замість того щоб приймати реквізити карток у себе на сервері, бекенд лише створює намір на оплату у Stripe, повертає клієнту client_secret і чекає на подію від платіжного сервісу. Така схема знімає з проєкту обов'язки PCI-DSS і спрощує підтримку різних способів оплати [24, 26].

Загальна послідовність взаємодії клієнта, сервера, бази і Stripe показана на рис. 3.6.

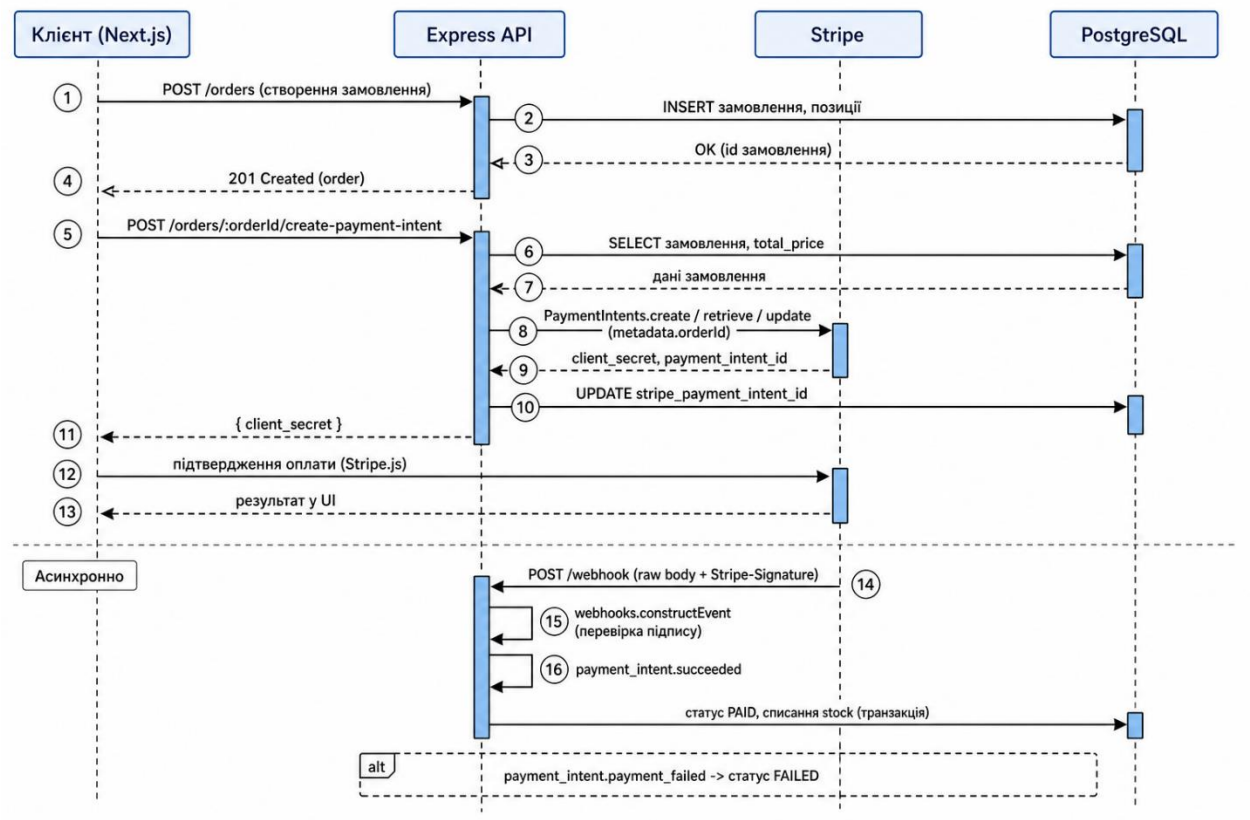

Рисунок. 3.6 UML sequence: оплата через Stripe (Payment Intent + вебхук)

Послідовність дій така. Клієнт надсилає POST /orders — сервер у транзакції створює замовлення з позиціями та знімками атрибутів у order_items. Далі на кроці оплати фронтенд викликає POST /orders/:orderId/create-payment-intent: контролер перевіряє замовлення в БД, створює або оновлює Payment Intent у Stripe (з metadata.orderId) і повертає client_secret. Оплату клієнт завершує через Stripe.js уже на своєму боці.

Підтвердження результату приходить не у відповіді на HTTP-запит, а окремою подією — вебхуком. Маршрут POST /webhook приймає сирий запит, перевіряє підпис через stripe.webhooks.constructEvent і обробляє дві головні події:

- payment_intent.succeeded — замовлення переходить у статус paid, у тій самій транзакції зменшуються залишки stock у product_variants;
- payment_intent.payment_failed — статус замовлення змінюється на failed, залишки лишаються незмінними.

Розділення синхронної частини (створення Payment Intent) і асинхронної (вебхук) дозволяє узгодити стан замовлення з реальним результатом платежу навіть тоді, коли Stripe надсилає події повторно: у репозиторії передбачено перевірку, що статус не міняється другий раз і залишки не списуються двічі.

3.4.1 Реалізація функціональності для різних ролей користувачів

Клієнтська частина веб-застосунку показує різний інтерфейс і набір функцій залежно від ролі автентифікованого користувача. У системі дві ролі — користувач (покупець) та адміністратор. Роль зчитується з JWT-токена в HTTP-only cookie і впливає на доступ до сторінок та операцій API.

Користувач (роль user)

1. перегляд каталогу товарів із фільтрами за категоріями та атрибутами (колір, розмір, матеріал);
2. перегляд картки товару з вибором варіанта і доступних зображень;
3. робота з кошиком: додавання й видалення позицій, зміна кількості, збереження стану між сеансами;
4. поетапне оформлення замовлення — контактні дані, адреса доставки, вибір способу доставки, оплата;
5. онлайн-оплата через Stripe і перегляд результату транзакції;
6. Перегляд історії власних замовлень.
7. перегляд історії власних замовлень;

Адміністратор (роль admin)

1. робота зі списком бажаного — додавання варіантів і їх видалення.
2. доступ до адміністративного контуру через окрему сторінку;
3. управління категоріями каталогу: створення, редагування, видалення;
4. управління товарами: додавання карток товарів, редагування полів, видалення;
5. управління варіантами (SKU) товарів і прив'язка атрибутів за моделлю EAV;
6. завантаження та впорядкування зображень для кожного варіанта;

7. додавання нових значень атрибутів (наприклад, нового кольору або розміру), а також доступ до перегляду замовлень покупців і зміни їх статусу.

Розмежування прав я реалізував на двох рівнях. На клієнті адміністративні сторінки показуються лише користувачу з відповідною роллю, а на сервері всі адмін-маршрути захищені middleware (authMiddleware + adminMiddleware), яка перевіряє роль із JWT до того, як викликати контролер. Завдяки цьому навіть пряме звернення до API без UI не дасть звичайному користувачеві виконати адмінські операції.

3.4.2 Взаємодія з API та клієнтська частина авторизації

Клієнтська частина веб-застосунку звертається до серверного REST API через стандартний fetch із прапором credentials: "include". Так браузер автоматично віддає в запит HTTP-only cookie з JWT-токеном (описаним у підрозділі 2.6). Завдяки цьому клієнтський код не зберігає і не передає токен самостійно — відповідальність за автентифікацію цілком лежить на сервері [23].

Запити до публічних ендпоінтів (каталог, фільтри, картка товару) йдуть без додаткових налаштувань. Для захищених маршрутів (/orders, /wish-lists, /addresses, адмінські операції каталогу) на сервері спрацьовує middleware з п. 2.6; якщо токена немає або він невалідний — клієнт отримує 401 Unauthorized, а при недостатніх правах — 403 Forbidden. Клієнтський код обробляє ці випадки уніфіковано: переводить користувача на сторінку входу або показує повідомлення про відмову в доступі.

На рівні інтерфейсу адміністративний контур видно лише користувачеві з відповідною роллю: при завантаженні адмін-сторінок клієнт перевіряє роль (через окремий ендпоінт або значення, отримане при вході) і ховає службові розділи від звичайних користувачів. Однак ця перевірка — суто UX-захист: справжній контроль доступу робиться на сервері (adminMiddleware), тож і пряме звернення до API без UI не дасть виконати адмін-операції.

Стан між сеансами клієнта я зберігаю мінімально і цілеспрямовано. Кошик реалізовано окремим store на бібліотеці Zustand зі збереженням у localStorage: коли користувач повертається на сайт, вміст кошика підтягується без додаткового запиту до API. А ось дані користувача (ім'я, email, історія замовлень, адреси) на клієнті не кешуються — їх запитує сервер за потреби, і вони завжди мають актуальний стан.

Приклад HTTP-запиту до серверного API з підтримкою cookie-автентифікації — на рис. 3.7.

```
export async function createOrder(
  payload: CreateOrderPayload,
): Promise<CreatedOrder> {
  const response = await fetch(`${API_BASE}/orders`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    credentials: "include",
    body: JSON.stringify({
      shipping_first_name: payload.shipping.first_name,
      shipping_last_name: payload.shipping.last_name,
      shipping_cost: payload.shippingCost,
      shipping_email: payload.shipping.email,
      shipping_phone: payload.shipping.phone,
      shipping_city: payload.shipping.city,
      shipping_address: payload.shipping.address,
      shipping_postal_code: payload.shipping.postal_code,
      shipping_country: payload.shipping.country,
      shipping_company: payload.shipping.company || undefined,
      shipping_apartment: payload.shipping.apartment || undefined,
      items: payload.items.map((item) => ({
        variant_id: item.id,
        quantity: item.quantity,
      })),
    }),
  });

  return parseResponse<CreatedOrder>(response);
}
```

Рисунок 3.7. Виклик API з клієнтської частини (fetch із підтримкою cookie)

3.5 Програмна реалізація ключових модулів системи

У цьому підрозділі наведено програмну реалізацію чотирьох ключових аспектів системи: моделі даних, серверної логіки, інтерфейсу користувача та механізмів безпеки. Для кожного аспекту подано фрагменти коду (лістинги) із стислим поясненням.

3.5.1 Реалізація моделі даних

Базу даних розгорнуто у схемі ecommerce СУБД PostgreSQL. Каталог описано за моделлю «категорія → товар → варіант», а характеристики варіантів (колір, розмір) винесено у довідники за моделлю EAV. DDL основних таблиць наведено в лістингу 3.1.

Лістинг 3.1 — DDL таблиць каталогу та моделі EAV

```
CREATE TABLE ecommerce.products (
  id          SERIAL PRIMARY KEY,
  category_id INTEGER NOT NULL REFERENCES ecommerce.categories(id),
  material_id INTEGER REFERENCES ecommerce.materials(id),
  name        TEXT NOT NULL,
  slug        TEXT NOT NULL UNIQUE,
  base_price  NUMERIC(10,2) NOT NULL
);
CREATE TABLE ecommerce.product_variants (
  id         SERIAL PRIMARY KEY,
  product_id INTEGER NOT NULL REFERENCES ecommerce.products(id),
  sku        TEXT NOT NULL UNIQUE,
  price      NUMERIC(10,2),
  stock      INTEGER NOT NULL DEFAULT 0 CHECK (stock >= 0)
);
CREATE TABLE ecommerce.attributes (
  id SERIAL PRIMARY KEY, code TEXT NOT NULL UNIQUE   -- 'color', 'size'
);
CREATE TABLE ecommerce.attribute_values (
  id           SERIAL PRIMARY KEY,
  attribute_id INTEGER NOT NULL REFERENCES ecommerce.attributes(id),
  value        TEXT NOT NULL
);
CREATE TABLE ecommerce.variant_attribute_values (
```

```
    variant_id                          INTEGER   NOT   NULL   REFERENCES
ecommerce.product_variants(id),
    attribute_value_id      INTEGER      NOT      NULL      REFERENCES
ecommerce.attribute_values(id),
    PRIMARY KEY (variant_id, attribute_value_id)
);
```

Оскільки фасетна фільтрація виконує з'єднання зв'язкової таблиці variant_attribute_values, її ключові колонки проіндексовано (лістинг 3.2). Вплив індексів на час відповіді досліджено в п. 3.6.2.

Лістинг 3.2 — Створення індексів для пришвидшення вибірок EAV

```
CREATE INDEX idx_vav_variant    ON ecommerce.variant_attribute_values
(variant_id);
CREATE INDEX idx_vav_value      ON ecommerce.variant_attribute_values
(attribute_value_id);
CREATE   INDEX   idx_variants_prod   ON   ecommerce.product_variants
(product_id);
CREATE INDEX idx_products_cat  ON ecommerce.products (category_id);
```

3.5.2 Реалізація серверної логіки

Серверну частину реалізовано на Express за схемою «маршрут → контролер → репозиторій». Параметри фільтра нормалізуються, а контролер формує JSON-відповідь із набором фасетів та їх лічильниками (лістинг 3.3).

Лістинг 3.3 — Формування відповіді з фасетами (filters.controller.js)

```
export const getFacets = async (req, res, next) => {
  try {
    const filters = parseProductFilters(req.query);
    const rows = await getFacetsRepository(filters);
    const facets = { color: [], size: [], fabric: [] };
    for (const r of rows) {
      const item = { value: r.value, count: r.count };
      if (r.attribute_code === "color")  facets.color.push(item);
      if (r.attribute_code === "size")   facets.size.push(item);
      if (r.attribute_code === "fabric") facets.fabric.push(item);
    }
    res.json({ facets });
  } catch (error) { next(error); }
};
```

Окремої уваги потребує обробка повторних подій оплати (колізії вебхуків): Stripe може доставити одну й ту саму подію повторно. Щоб уникнути подвійного списання залишків, замовлення блокується (SELECT … FOR UPDATE), і якщо воно вже має статус paid, транзакція завершується без змін (лістинг 3.4).

Лістинг 3.4 — Ідемпотентне списання залишків (orders.repository.js)

```
await client.query("BEGIN");
const { rows } = await client.query(
  `SELECT * FROM ecommerce.orders WHERE id = $1 FOR UPDATE`, [orderId]);
const order = rows[0];

if (order.status === "paid") {          // повтор вебхука: не списуємо вдруге
  await client.query("COMMIT");
  return order;
}
await client.query(
  `UPDATE ecommerce.product_variants pv
     SET stock = pv.stock - oi.quantity
    FROM ecommerce.order_items oi
    WHERE oi.order_id = $1 AND oi.variant_id = pv.id
      AND pv.stock >= oi.quantity`, [orderId]);
await client.query(
  `UPDATE ecommerce.orders SET status = 'paid' WHERE id = $1`, [orderId]);
await client.query("COMMIT");
```

3.5.3 Реалізація інтерфейсу користувача

Клієнтську частину побудовано на компонентах React (Next.js). Глобальний стан зведено до мінімуму: між сеансами зберігається лише кошик. Його реалізовано окремим сховищем (store) на бібліотеці Zustand із middleware persist, що серіалізує стан у localStorage (лістинг 3.5). Завдяки цьому при поверненні користувача вміст кошика відновлюється без додаткового запиту до API.

Лістинг 3.5 — Сховище стану кошика на Zustand (cartStore.ts)

```
export const useCartStore = create<CartStore>()(
  persist(
```

```
          (set) => ({
            items: [],
            addItem: (item) => set((state) => {
              const existing = state.items.find((i) => i.id === item.id);
              return { items: existing
                ? state.items.map((i) =>
                    i.id === item.id ? { ...i, quantity: i.quantity +
item.quantity } : i)
                : [...state.items, item] };
            }),
            removeItem: (id) => set((state) => ({
              items: state.items.filter((i) => i.id !== id) })),
            updateQuantity: (id, quantity) => set((state) => ({
              items: state.items.map((i) => (i.id === id ? { ...i, quantity }
: i)) })),
            clearCart: () => set({ items: [] }),
          }),
          { name: "cart-storage" },   // ключ у localStorage
        ),
      );
```

3.5.4 Забезпечення безпеки застосунку

Безпеку застосунку реалізовано на кількох рівнях. Паролі не зберігаються у відкритому вигляді: при реєстрації обчислюється хеш за алгоритмом bcrypt, а автентифікація виконується через JWT, що видається у cookie з прапорцями httpOnly, secure та sameSite (лістинг 3.6) [15].

Лістинг 3.6 — Хешування пароля та видача JWT у захищеному cookie (auth.controller.js)

```
      const passwordHash = await bcrypt.hash(password, 10);
      const user = await createUserRepository({ email, passwordHash, role:
"user" });

      const token = jwt.sign(
        { userId: user.id, role: user.role },
        process.env.JWT_SECRET, { expiresIn: "4h" });

      res.cookie("token", token, {
        httpOnly: true,                              // недоступний для JS
        secure: process.env.NODE_ENV === "production",
```

```
    sameSite: "strict",                          // захист від CSRF
    maxAge: 4 * 60 * 60 * 1000,
});
```

Доступ до захищених маршрутів контролює middleware: authMiddleware перевіряє підпис токена і витягує ідентифікатор та роль користувача, а adminMiddleware обмежує адміністративні операції роллю admin. За відсутності чи недійсності токена повертається 401 Unauthorized, за браку прав — 403 Forbidden (лістинг 3.7).

Лістинг 3.7 — Перевірка токена та ролі (middleware/authMiddleware.js)

```
const authMiddleware = (req, res, next) => {
  const token = req.cookies.token;
  if (!token) return res.status(401).json({ error: "Unauthorized" });
  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.userId = decoded.userId;
    req.role = decoded.role;
    next();
  } catch (err) {
    return res.status(401).json({ error: "Unauthorized" });
  }
};

const adminMiddleware = (req, res, next) => {
  if (!req.role || req.role !== "admin")
    return res.status(403).json({ error: "Forbidden" });
  next();
};
```

Додатково всі звернення до бази даних виконуються параметризованими запитами ($1, $2, …), тож значення з HTTP-запитів не потрапляють безпосередньо в текст SQL, що унеможливлює SQL-ін'єкції [25].

3.6 Перевірка роботи системи

3.6.1 Перевірка функціональних можливостей

У процесі розробки я перевірив повний цикл роботи системи відповідно до функціональних вимог із підрозділу 1.5. Зокрема, перевірено такі сценарії:

- реєстрація користувача та вхід (включно з Google-автентифікацією);
- перегляд каталогу, фасетна фільтрація, картка товару з варіантами;
- додавання товарів у кошик і збереження стану між сеансами;
- оформлення замовлення з вибором адреси та способу доставки;
- оплата у тестовому режимі Stripe з обробкою вебхука й оновленням статусу замовлення;
- адміністрування каталогу: створення категорій, товарів, варіантів і завантаження зображень.

Усі сценарії пройдено успішно. Окремо я перевірив нефункціональні вимоги — захист API через JWT-автентифікацію, час відповіді API на типових запитах каталогу та узгодженість даних під час транзакційного оновлення статусу замовлення і залишків після оплати.

3.6.2 Результати експерименту з продуктивності бази даних

Щоб оцінити вплив індексів на швидкість вибірок у моделі EAV, я провів експеримент: на тестовій копії БД виконувався однотипний запит фільтрації товарів за значеннями атрибутів. Запит запускався в двох конфігураціях схеми ecommerce — без індексів і зі складеними індексами на таблицях variant_attribute_values та product_variants. Обсяг даних змінювали від 100 до 10 000 товарів і для кожного запуску фіксували середній час відповіді бази даних у мілісекундах.

Результати експерименту наведено на рис. 3.8.

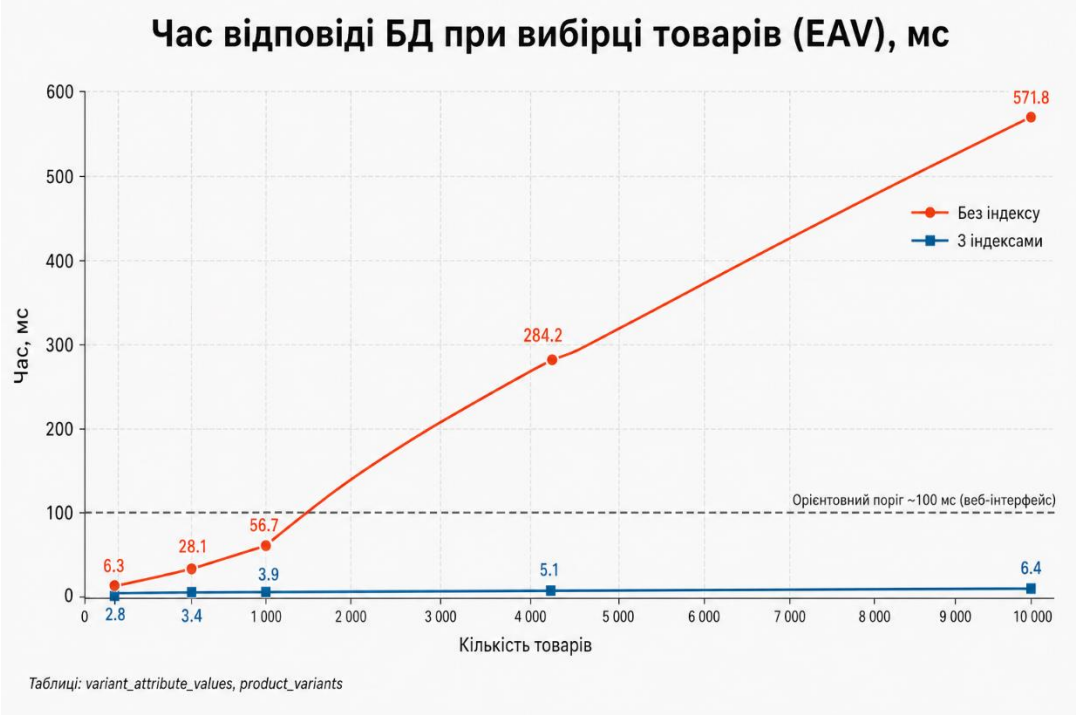

Рисунок 3.8. Час відповіді БД при вибірці товарів в EAV-схемі (мс)

Як видно з графіка, без індексів час виконання запиту зростає майже лінійно з кількістю позицій: близько 6 мс при 100 товарах і вже 572 мс при 10 000. Причина — послідовне сканування таблиці variant_attribute_values по кожному атрибуту фільтра, що на великих обсягах суттєво навантажує систему.

Із складеними індексами час відповіді стабілізується в межах 3–7 мс і фактично не залежить від обсягу даних навіть при 10 000 товарів. Орієнтовний поріг прийнятної реакції для веб-інтерфейсу — близько 100 мс: без індексів його перевищено вже на кількох тисячах позицій, а з індексами система впевнено лишається в комфортних межах для масштабу реального інтернет-магазину.

Тож експеримент показав: індексування ключових таблиць EAV-моделі — необхідна умова стабільної роботи системи при зростанні асортименту. Обраний підхід до проєктування індексів утримує час відповіді БД на рівні одиниць мілісекунд незалежно від обсягу каталогу і підтверджує доцільність

застосування EAV-моделі для проєкту магазину з великою кількістю варіантів і атрибутів товарів.

Для підтвердження працездатності системи нижче наведено скріншоти основних користувацьких сценаріїв із п. 3.6.1 та інтерфейс адміністрування каталогу.

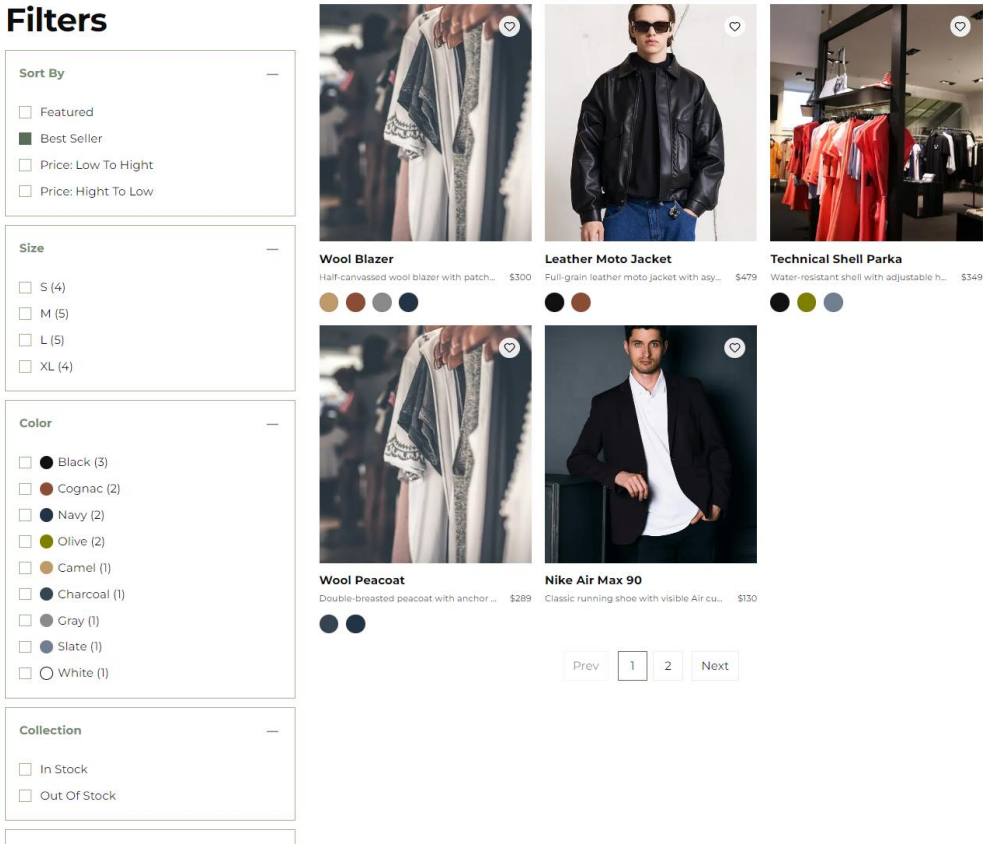

Рисунок 3.9. Головна сторінка каталогу інтернет-магазину.

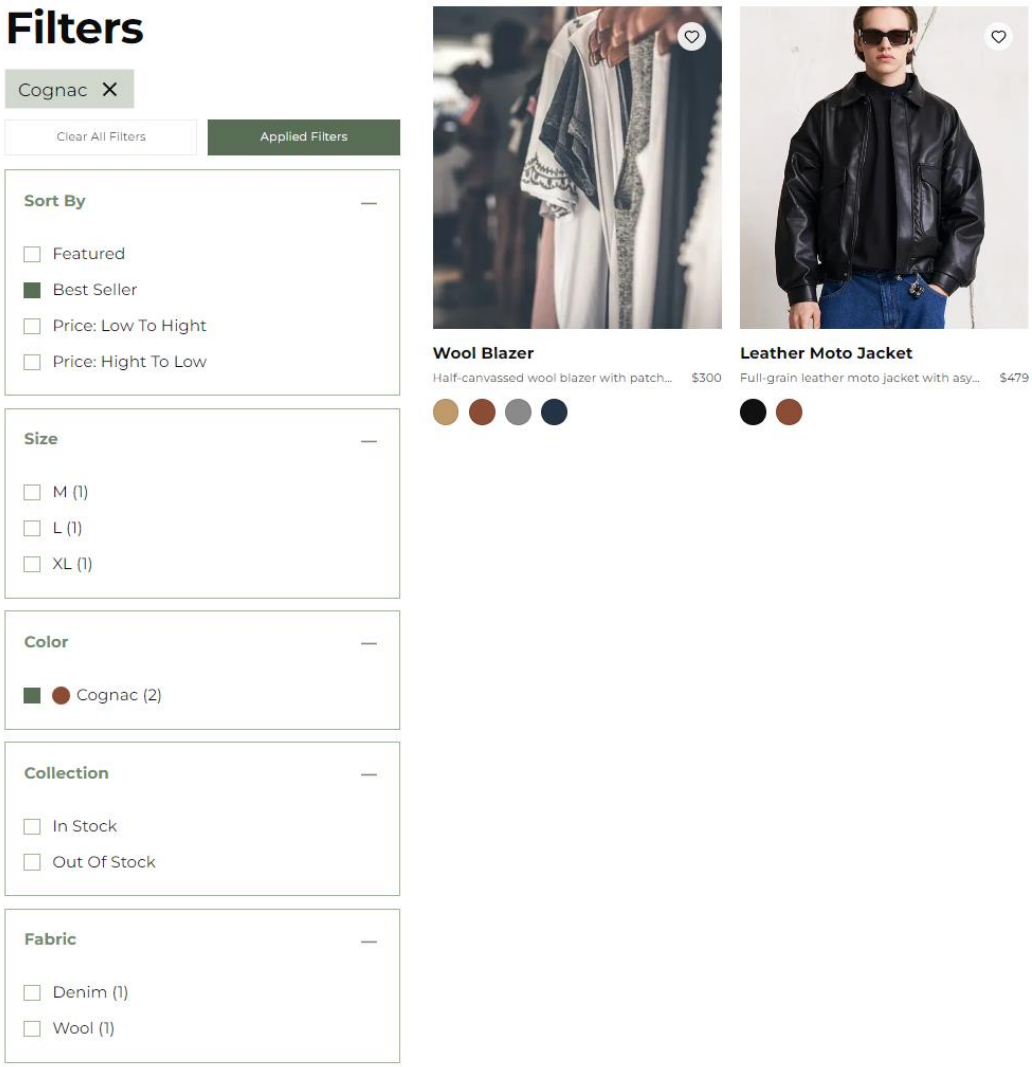

Рисунок 3.10. Сторінка каталогу з активною фасетною фільтрацією.

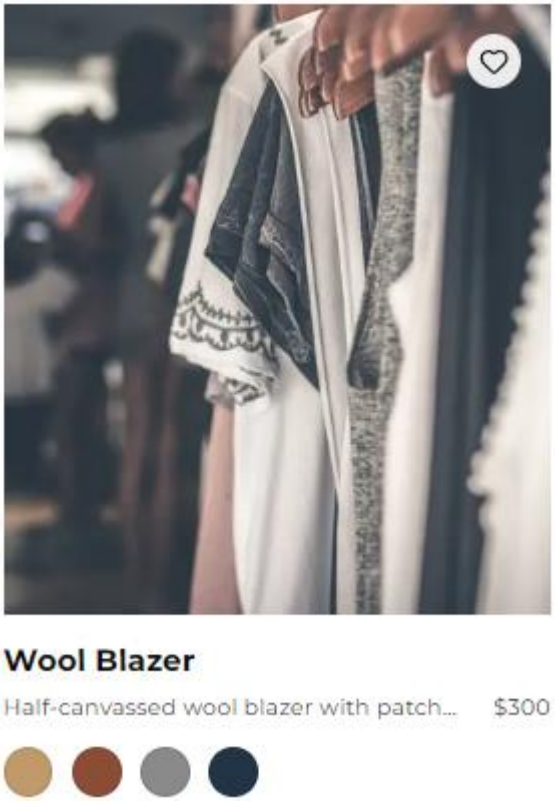

Рисунок 3.11. Картка товару з вибором варіанта.

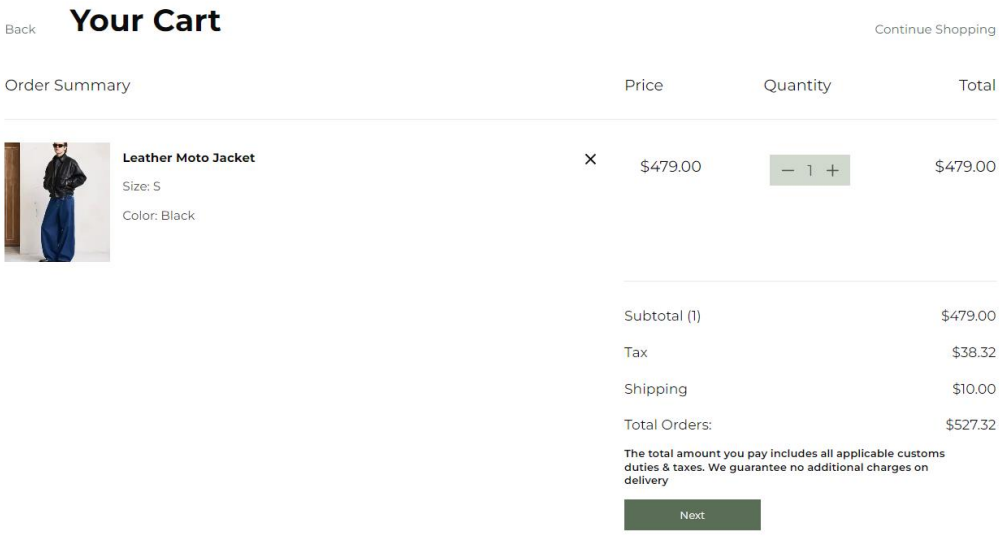

Рисунок 3.12. Кошик користувача.

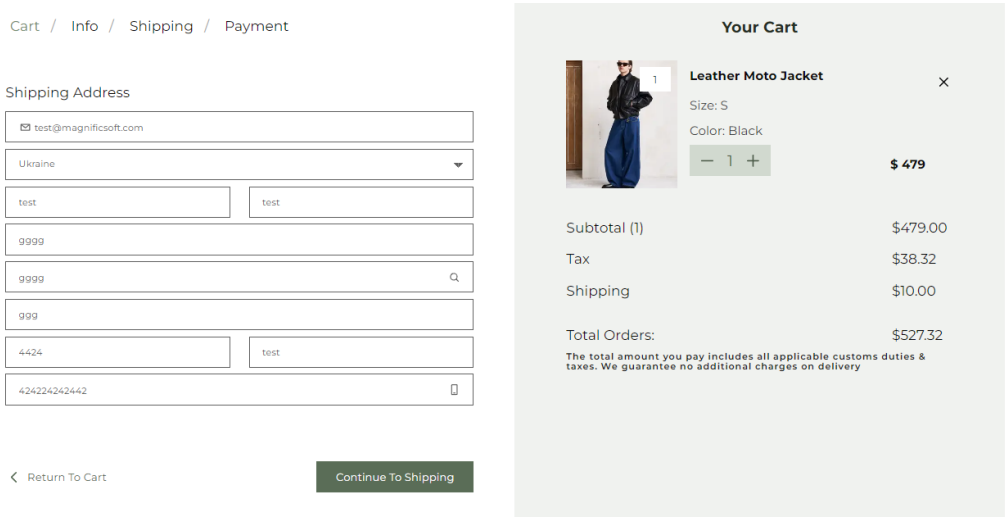

Рисунок 3.13. Етап оформлення замовлення (введення даних, спосіб доставки, оплата).

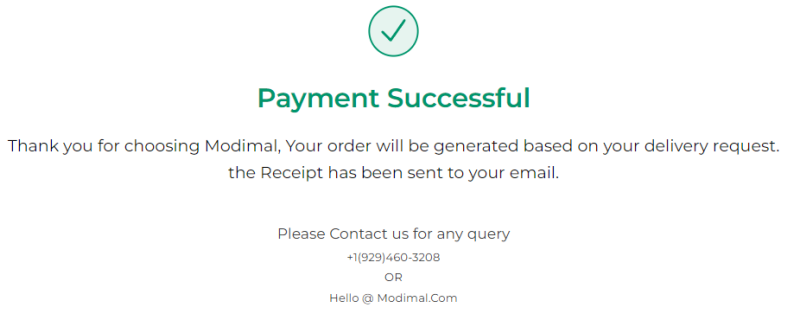

Рисунок 3.14. Результат успішної оплати через Stripe.

Адміністрування каталогу:

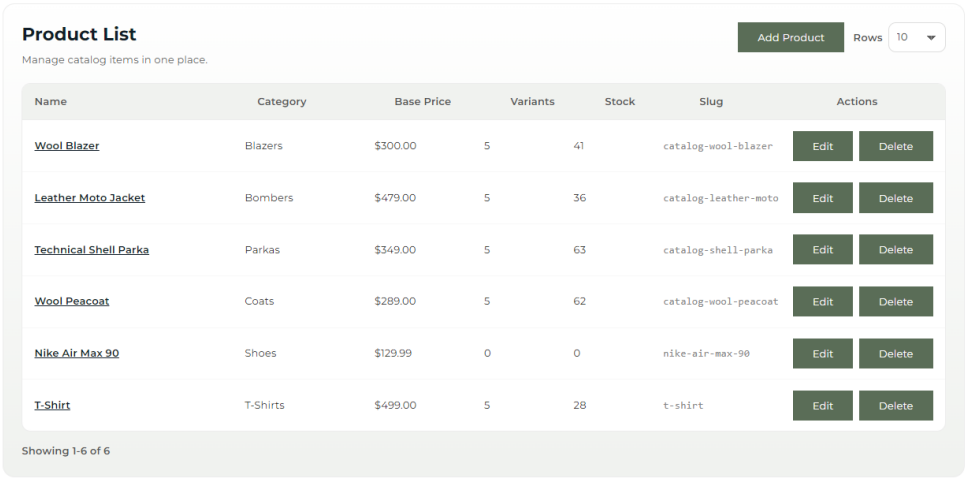

Рисунок 3.15. Сторінка адміністрування: перелік товарів.

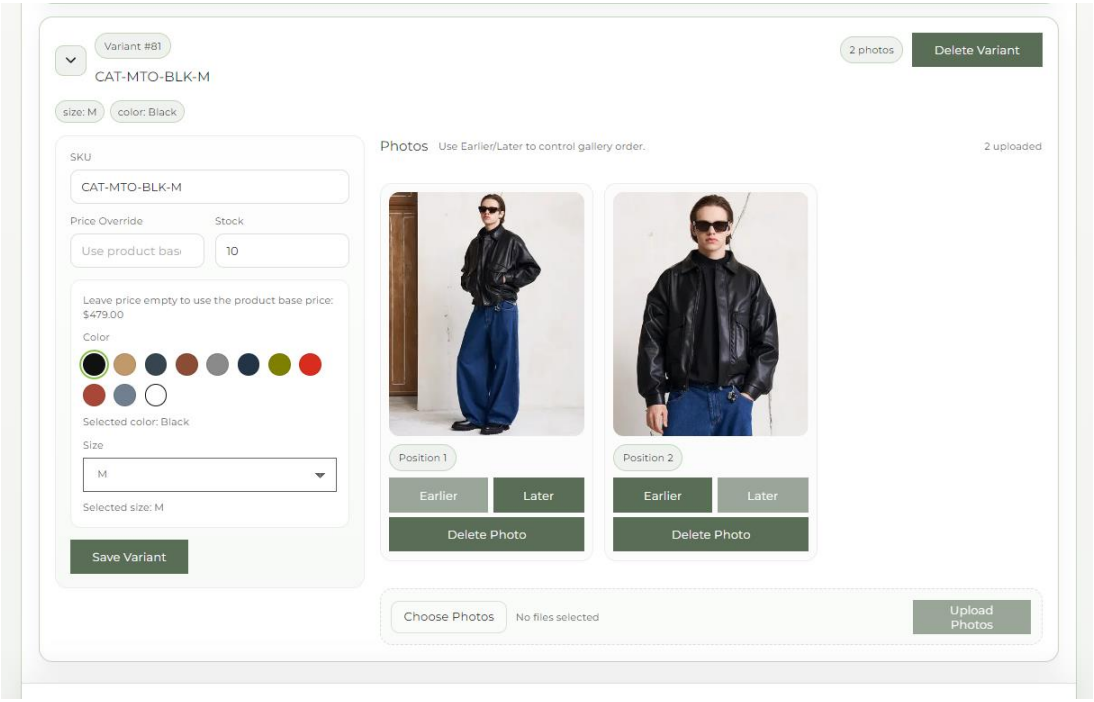

Рисунок 3.16. Редагування варіантів і завантаження зображень.

Висновки до розділу 3

У третьому розділі я описав проєктування й програмну реалізацію інформаційної системи інтернет-магазину. Архітектуру побудовано з розділенням клієнтської (Next.js) та серверної (Express) частин, спроєктовано схему БД ecommerce з моделлю EAV для атрибутів варіантів товарів і реалізовано REST API для каталогу, замовлень, оплати та адміністрування. Доступ до PostgreSQL організовано через драйвер pg із параметризованими запитами і транзакційним створенням замовлень.

Окремо реалізовано онлайн-оплату через Stripe із механізмом Payment Intent та обробкою вебхуків — це й забезпечує узгодженість статусу замовлень і складських залишків незалежно від того, як саме діяв користувач у платіжному вікні. Розмежування ролей user і admin виконано на JWT-автентифікації у HTTP-only cookie: вона контролює доступ і до користувацьких функцій (замовлення, адреси), і до операцій адміністрування каталогу.

У ході перевірки системи я послідовно перевірив основні сценарії — від реєстрації до оплати в тестовому режимі Stripe — і провів експеримент із продуктивності бази даних. Результати експерименту підтвердили: складені індекси на таблицях EAV-моделі утримують час відповіді БД на рівні одиниць мілісекунд при зростанні обсягу каталогу до 10 000 товарів. Це й підтверджує доцільність обраної архітектури і створює основу для подальшого масштабування системи в разі впровадження.

ВИСНОВКИ

У ході розробки було реалізовано такі функціональні можливості:

- реєстрацію й автентифікацію користувачів (зокрема вхід через Google);
- ведення каталогу інтернет-магазину з підтримкою категорій, товарів і варіантів за моделлю EAV;
- фасетну фільтрацію та сортування товарів на вітрині;
- кошик користувача зі збереженням стану між сеансами;
- поетапне оформлення замовлення з вибором адреси й способу доставки;
- онлайн-оплату через сервіс Stripe із обробкою вебхуків і автоматичним списанням залишків;
- адміністрування каталогу: керування категоріями, товарами, варіантами, атрибутами та зображеннями.

Технічні характеристики розробленого веб-застосунку такі:

- клієнтська частина — Next.js (App Router) і React на TypeScript;
- серверна частина — окремий REST API на Node.js та Express;
- СУБД — PostgreSQL зі схемою ecommerce;
- доступ до бази — драйвер pg із параметризованими SQL-запитами та транзакціями;
- автентифікація й авторизація — власна реалізація на JWT у HTTP-only cookie з розмежуванням ролей user і admin;
- онлайн-оплата — інтеграція зі Stripe (Payment Intent, вебхуки);
- зберігання зображень — об'єктне сховище Supabase Storage.

Програма працює за такою інструкцією:

- користувач відкриває каталог, обирає категорію і задає фільтри за атрибутами товарів;
- із картки товару обирає потрібний варіант і додає його до кошика;
- у кошику переглядає позиції, змінює кількість або прибирає зайве, далі переходить до оформлення замовлення;

- послідовно вводить контактні дані, адресу доставки, обирає спосіб доставки і переходить до оплати;
- завершує оплату у платіжному вікні Stripe; після підтвердження статус замовлення оновлюється автоматично;
- авторизований користувач може переглянути історію своїх замовлень і зберегти адресу для повторного використання;
- адміністратор працює в окремому розділі інтерфейсу — керує каталогом і переглядає замовлення.

У разі впровадження запропонований веб-застосунок автоматизує основні процеси інтернет-торгівлі: ведення каталогу з варіативними атрибутами, оформлення замовлення з онлайн-оплатою і керування асортиментом. Простий користувацький інтерфейс та чітке розмежування ролей закладають основу для практичного використання системи в малому й середньому бізнесі та її подальшого масштабування при зростанні асортименту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Пасічник В. В., Резніченко В. А. Організація баз даних та знань. – Київ : BHV, 2006. – 480 с.

2. Tilkov S., Vinoski S. Node.js: Using JavaScript to Build High-Performance Network Programs // IEEE Internet Computing. – 2010. № 6. – P. 80–83.

3. Next.js Documentation [Електронний ресурс]. URL: https://nextjs.org/docs (дата звернення: 11.05.2026).

4. Nielsen J. Response Times: The 3 Important Limits [Електронний ресурс]. URL: https://www.nngroup.com/articles/response-times-3-important-limits/ (дата звернення: 11.05.2026).

5. Codd E. F. A Relational Model of Data for Large Shared Data Banks // Communications of the ACM. – 1970. – Vol. 13, № 6. – P. 377–387. – DOI: 10.1145/362384.362685.

6. Date C. J. An Introduction to Database Systems. – 8th ed. – Boston : Pearson / Addison-Wesley, 2003. – 1040 p.

7. Garcia-Molina H., Ullman J. D., Widom J. Database Systems: The Complete Book. – 2nd ed. – Upper Saddle River : Pearson Prentice Hall, 2008. – 1248 p.

8. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. – Sebastopol : O'Reilly Media, 2017. – 614 p.

9. Nadkarni P. M., Marenco L., Chen R., Skoufos E., Shepherd G., Miller P. Organization of Heterogeneous Scientific Data Using the EAV/CR Representation // Journal of the American Medical Informatics Association. – 1999. – Vol. 6, № 6. – P. 478–493. – DOI: 10.1136/jamia.1999.0060478.

10. Momjian B. PostgreSQL: Introduction and Concepts. – Boston : Addison-Wesley, 2001. – 496 p.

11. PostgreSQL 16 Documentation [Електронний ресурс]. URL: https://www.postgresql.org/docs/16/ (дата звернення: 11.05.2026).

12. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : Ph.D. dissertation [Електронний ресурс]. University of California, Irvine, 2000. URL: https://ics.uci.edu/~fielding/pubs/dissertation/ (дата звернення: 11.05.2026).

13. Fielding R., Nottingham M., Reschke J. HTTP Semantics. RFC 9110 [Електронний ресурс]. Internet Engineering Task Force, 2022. URL: https://www.rfc-editor.org/rfc/rfc9110 (дата звернення: 11.05.2026).

14. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519 [Електронний ресурс]. Internet Engineering Task Force, 2015. URL: https://www.rfc-editor.org/rfc/rfc7519 (дата звернення: 11.05.2026).

15. Provos N., Mazières D. A Future-Adaptable Password Scheme // Proceedings of the 1999 USENIX Annual Technical Conference, FREENIX Track. – Monterey, CA : USENIX Association, 1999. – P. 81–92.

16. Helland P. Idempotence Is Not a Medical Condition // Communications of the ACM. – 2012. – Vol. 55, № 5. – P. 56–65. – DOI: 10.1145/2160718.2160734.

17. Young A., Meck B., Cantelon M., Oxley T., Harter M., Holowaychuk T. J., Rajlich N. Node.js in Action. – 2nd ed. – Shelter Island : Manning Publications, 2017. – 392 p.

18. Express.js – Fast, unopinionated, minimalist web framework for Node.js [Електронний ресурс]. URL: https://expressjs.com/ (дата звернення: 11.05.2026).

19. React Documentation [Електронний ресурс]. URL: https://react.dev/learn (дата звернення: 11.05.2026).

20. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. – 2nd ed. – Sebastopol : O'Reilly Media, 2020. – 310 p.

21. TypeScript Handbook [Електронний ресурс]. URL: https://www.typescriptlang.org/docs/handbook/ (дата звернення: 11.05.2026).

22. Flanagan D. JavaScript: The Definitive Guide. – 7th ed. – Sebastopol : O'Reilly Media, 2020. – 706 p.

23. MDN Web Docs [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web (дата звернення: 11.05.2026).

24. Stripe API Reference [Електронний ресурс]. URL: https://docs.stripe.com/api (дата звернення: 11.05.2026).

25. OWASP Top 10:2021 [Електронний ресурс]. URL: https://owasp.org/Top10/ (дата звернення: 11.05.2026).

26. Payment Card Industry Data Security Standard: Requirements and Testing Procedures. Version 4.0 [Електронний ресурс]. PCI Security Standards Council, 2022. URL: https://www.pcisecuritystandards.org/document_library/ (дата звернення: 11.05.2026).

27. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. – 3rd ed. – Berkeley : New Riders, 2014. – 216 p.

28. Wagner J. Web Performance in Action: Building Faster Web Pages. – Shelter Island : Manning Publications, 2017. – 376 p.

29. Соломін А. В. Веб-орієнтована розробка програмного забезпечення : практикум : навч. посіб. для студ. спец. 122 «Комп'ютерні науки та інформаційні технології» [Електронний ресурс]. – Київ : КПІ ім. Ігоря Сікорського, 2018. – 131 с. URL: https://ela.kpi.ua/handle/123456789/23678 (дата звернення: 11.05.2026).

30 . Бондарчук, А. П., & Глушак, О. М. (2025). Предиктивне управління оновленнями програмного забезпечення в інтернеті речей.Зв'язок,(5), 13-17.

31. Плескач В. Л., Затонацька Т. Г. Електронна комерція : підручник. – Київ : Знання, 2007. – 535 с.

32. Abramov, V., Astafieva, M., Boiko, M., Bodnenko, D., Bushma, A., Vember, V., & Yaskevych, V. (2021). Theoretical and practical aspects of the use of mathematical methods and information technology in education and science.

33. Машкіна, І. В., Абрамов, В. О., Носенко, Т. І., Іваніченко, Є. В., & Яскевич, В. О. (2024). Навчально-методичний посібник до виконання і захисту кваліфікаційної роботи бакалавра спеціальностей 122 Комп’ютерні науки для здобувачів вищої освіти денної форми навчання.