НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

“До захисту допущено”
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА

“___”_____2025 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
“Цифрові технології в енергетиці”
зі спеціальності 122 “Комп’ютерні науки”

на тему: Система управління замовленнями споживачів в дистрибуції одягу та відповідних аксесуарів

Виконав:
студент IV курсу, групи ТР-13

МАМРЕНКО Дмитро Володимирович _____
(прізвище, ім’я, по батькові) *(підпис)*

Керівник: *доцент кафедри цифрових технологій в енергетиці*
к.ф-м.н., доцент, ДОНЕЦЬ Андрій Георгійович _____
(посада, науковий ступінь, вчене звання, прізвище, ім’я, по батькові) *(підпис)*

Рецензент: *доцент кафедри ТАЕ*
к.т.н., доцент, СІРИЙ Олександр Анатолійович _____
(посада, вчене звання, науковий ступінь, ім’я, ПРІЗВИЩЕ) *(підпис)*

Н.контроль: *асистент Волков Олександр Володимирович* _____
(посада, ім’я, ПРІЗВИЩЕ) *(підпис)*

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра _____ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ_____

Рівень вищої освіти – перший (бакалаврський)
Спеціальність 122 “Комп’ютерні науки”_____
Освітньо-професійна програма “Цифрові технології в енергетиці”_____

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА

“___”_____2025 р.

ЗАВДАННЯ
на дипломну роботу студенту

_____**МАМРЕНКО Дмитру Володимировичу**_____
(прізвище, ім’я, по батькові)

1. Тема роботи Система управління замовленнями споживачів в дистрибуції одягу та відповідних аксесуарів
керівник роботи _____**Донець Андрій Георгійович доцент**_____
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом по університету від «02» червня 2025р. №1875-с

2. Термін подання студентом роботи **04.06.2025 року**

3. Вихідні дані до роботи: мова програмування — C#, фреймворк ASP.NET Core Blazor Server для реалізації клієнтської та серверної логіки. Система управління базами даних — Microsoft SQL Server. Доступ до бази даних — Entity Framework Core як ORM-рішення. Застосовано HTML та CSS для верстки інтерфейсу. JavaScript-бібліотеку Sortable.js для додаткового функціоналу. Середовище розробки — Visual Studio 2022.

4. Перелік питань, які потрібно розробити

1) опис завдання автоматизованої обробки замовлень у сфері дистрибуції одягу та аксесуарів

2) аналіз існуючих програмних рішень для управління замовленнями в споживчій дистрибуції

3) опис функціональних та нефункціональних вимог до інформаційної системи

4) обґрунтування вибору технологій для реалізації вебзастосунку

5) розробка інформаційної системи управління замовленнями з урахуванням складського обліку, промокодів і варіантів товару

6) опис основних сценаріїв взаємодії користувача та адміністратора із системою.

5. Орієнтовний перелік ілюстративного матеріалу: зображення інтерфейсу веб-застосунку, скріншоти частин коду, діаграма прецедентів.

6. Дата видачі завдання «19» вересня 2024 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Затвердження теми роботи	19.09.24	Виконано
2	Аналіз задачі обробки замовлень у сфері дистрибуції одягу та аксесуарів	20.10.24-20.11.24	Виконано
3	Проєктування архітектури інформаційної системи та структури бази даних	01.12.24-07.01.25	Виконано
4	Реалізація серверної логіки та клієнтського інтерфейсу з урахуванням ролей користувачів	10.01.24-15.03.25	Виконано
5	Інтеграція функціональних модулів: обробка замовлень, промокоди, склад, адміністративна панель	21.04.25-11.05.25	Виконано
6	Оформлення пояснювальної записки	19.05.25-26.05.25	Виконано
7	Захист програмного продукту	14.05.25	Виконано
8	Передзахист	27.05.25	Виконано
9	Захист	16.06.25-20.06.25	Виконано

Студент _____ **Дмитро МАМРЕНКО**
(підпис) (ім'я, ПРІЗВИЩЕ)

Науковий керівник _____ **Андрій ДОНЕЦЬ**
(підпис) (ім'я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота виконана на 75 сторінках, містить 25 ілюстрації 2 додатоки, 24 джерела у переліку посилань.

Мета роботи — створення вебзастосунку для автоматизації процесів обробки замовлень споживачів у сфері дистрибуції одягу та аксесуарів.

Методи та засоби: мова програмування C#, фреймворк Blazor Server для побудови інтерфейсу та серверної логіки, обробка запитів реалізована за допомогою Razor-компонентів. Для зберігання даних використано реляційну СУБД Microsoft SQL Server, із застосуванням ORM Entity Framework Core. Валідацію реалізовано за допомогою DataAnnotations, взаємодію з базою — через контекст ApplicationDbContext. Стилізацію інтерфейсу виконано з використанням CSS і Tailwind-підходів. Контроль версій здійснювався через Git, середовище розробки — Visual Studio 2022.

Результат — інформаційна система, що забезпечує облік товарів, керування замовленнями, застосування промокодів, облік залишків на складі, адміністративне редагування вмісту та взаємодію користувача з каталогом.

Ключові слова: BLAZOR, ЗАМОВЛЕННЯ, ВЕБЗАСТОСУНОК, C#, SQL SERVER, ДИСТРИБУЦІЯ, ПРОМОКОД, АДМІНІСТРУВАННЯ, КОШИК.

ABSTRACT

The diploma thesis consists of 75 pages, includes 25 illustrations 2 appendix, and 24 references in the list of sources.

The aim of the work is to develop a web application for automating consumer order processing in the distribution of clothing and accessories.

Methods and tools: C# programming language, Blazor Server framework for building the interface and server logic, request handling implemented via Razor components. Microsoft SQL Server relational DBMS is used for data storage, with ORM Entity Framework Core. Validation is implemented using DataAnnotations, and database interaction is performed via the ApplicationDbContext. Interface styling is done using CSS and Tailwind-based approaches. Version control was performed through Git, and the development environment was Visual Studio 2022.

Result — an information system that supports product catalog management, order processing, promo code application, inventory tracking, administrative content editing, and user interaction with the catalog.

Keywords: BLAZOR, ORDERS, WEB APPLICATION, C#, SQL SERVER, DISTRIBUTION, PROMOCODE, ADMINISTRATION, CART.

ЗМІСТ

ВСТУП

В умовах стрімкої цифровізації дистрибуційних процесів підприємства дедалі частіше звертаються до автоматизованих систем, що допомагають ефективно організувати обробку споживчих замовлень. Ринок одягу та аксесуарів характеризується високою динамікою, широким асортиментом, частими оновленнями позицій та потребою в точному обліку залишків на складі. У таких умовах важливу роль відіграє система управління замовленнями, що забезпечує взаємодію між споживачем і розподільчою ланкою.

Використання уніфікованих або надмірно загальних рішень не завжди відповідає специфіці конкретного бізнесу або проєкту, що створює потребу у розробці адаптованої інформаційної системи, що враховує реальні вимоги до процесів дистрибуції, категоризації продукції, роботи зі змінними характеристиками (розмір, стать, кількість), а також контролю за виконанням замовлень.

Метою роботи є розробка інформаційної системи управління замовленнями споживачів в дистрибуції одягу та відповідних аксесуарів, яка реалізує повний цикл прийому та обробки замовлень, включаючи механізми вибору товару, обліку залишків, застосування промокодів і адміністрування вмісту системи.

Для досягнення поставленої мети передбачено виконання таких завдань:

– провести аналіз підходів, методів та алгоритмів вирішення задачі керування замовленнями та обліку товарів у сфері дистрибуції;

– обґрунтувати вибір програмних засобів для реалізації функціональності системи;

– реалізувати інформаційну систему відповідно до поставлених функціональних і технічних вимог;

– провести тестування компонентів системи для перевірки відповідності поставленим завданням.

Таким чином, розроблена система спрямована на покриття всіх ключових етапів взаємодії між клієнтом і адміністративною частиною дистрибуційного

процесу – від вибору товару до оформлення замовлення та обліку складу. Результати реалізації демонструють практичну користь інформаційної системи як інструмента підвищення керованості, зменшення помилок і автоматизації повсякденних операцій у галузі роздрібної дистрибуції одягу та аксесуарів.

Апробація результатів роботи відбулась у рамках переддипломної практики, де було розроблено діючий прототип системи, перевірено його на реалістичних сценаріях обробки замовлень, а також підтверджено відповідність логіки роботи умовам дистрибуції та складського обліку.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ПРОДУКТУ

Розробка будь-якого комерційного застосунку починається з аналізу потреб. Очікування користувача зосереджені на швидкості, простоті й надійності. Замовник прагне автоматизації процесів, контролю над товарами й замовленнями, стабільної роботи та зручного керування вмістом.

Ключовими завданнями є точний облік товарів із варіантами (розмірами, статтю), підтримка промокодів, фільтрації, історії замовлень та адміністрування контенту. Необхідно врахувати реальні умови використання, зокрема наявність обмеженого персоналу, потребу в інтуїтивному інтерфейсі, швидкому пошуку, керуванні складом і доступі до замовлень.

Система повинна бути стабільною, масштабованою, придатною до розширення функціональності без переробки архітектури. Продукт має відповідати базовим сценаріям роботи користувача та адміністратора. Важливо передбачити контроль за залишками, логіку блокування дій у разі помилок, перевірки при оформленні, а також захист даних і розмежування прав доступу.

Усе перетворено на конкретні функції, що реалізовані у фінальному застосунку: каталог із фільтрацією, кошик із перевірками, система промокодів, інструменти адміністрування та повноцінна логіка обліку залишків. Перелічені вимоги лягли в основу архітектурного та технологічного рішення.

1.1 Постановка задачі

Інформаційна система, що реалізується в межах проєкту, має забезпечувати підтримку основних бізнес-процесів у сфері дистрибуції одягу та відповідних аксесуарів. Поставлено завдання розробити повнофункціональний вебзастосунок, орієнтований на малий і середній бізнес, із поділом доступу на користувацьку й адміністративну частини.

Першим завданням є створення каталогу товарів із можливістю структурування продукції за категоріями, варіантами розмірів і статтю. Для кожного варіанта товару передбачено окремий облік кількості на складі, що дозволяє уникнути ситуацій із продажу відсутньої продукції.

Друге завдання — побудова повного циклу обробки замовлень: від додавання товарів до кошика до підтвердження й зміни статусу замовлення. Важливо забезпечити перевірку наявності кожної позиції перед оформленням, а також реалізувати логіку зміни залишків товару при підтвердженні або скасуванні замовлення.

Наступним елементом є система промокодів. Кожен промокод має унікальний код, строк дії та обмеження за кількістю використання. Знижка застосовується до загальної суми замовлення лише за умови відповідності даним критеріям. Активації промокодів зберігаються та контролюються відповідно до ідентифікатора користувача.

Окрему увагу приділено зручності інтерфейсу. Для кінцевого користувача передбачено інтуїтивно зрозумілий механізм взаємодії з каталогом, кошиком і формою замовлення. Для адміністратора реалізовано керування товарами, категоріями, промокодами та замовленнями в рамках окремої захищеної панелі.

Останнім компонентом є впровадження рольової авторизації. Кожен користувач має чітко визначений рівень доступу до функціональності. Для адміністративної частини реалізовано механізми автентифікації та перевірки прав доступу. Також передбачено базовий захист даних і контроль дій авторизованих осіб.

1.2 Наявні програмні рішення для управління замовленнями в дистрибуції

Для побудови ефективної системи управління замовленнями необхідно враховувати досвід існуючих платформ, що реалізують подібну функціональність.

У межах аналізу предметної області розглянуто низку рішень, що використовуються в онлайн-торгівлі для автоматизації обробки замовлень, керування товарним асортиментом, застосування промокодів, організації клієнтського та адміністративного інтерфейсів.

Серед найпоширеніших систем [1] — Shopify [2], OpenCart [3], WooCommerce[4], Prom.ua [5], Rozetka Marketplace [6]. Shopify орієнтовано на користувачів без досвіду програмування: магазин розгортається за готовими шаблонами, але вимагає постійної оплати за підпискою, а розширення функціоналу обмежене. WooCommerce — розширення для CMS WordPress — дозволяє інтегрувати торговельну систему до вже створеного сайту, проте потребує додаткових плагінів, налаштувань і постійної технічної підтримки.

OpenCart є повноцінною CMS із відкритим кодом, що забезпечує базову функціональність для інтернет-магазину. Платформа підтримує роботу з товарами, категоріями, кошиком і замовленнями. Проте реалізація специфічних задач, таких як складський облік варіантів товарів або логіка активації промокодів із лімітами використання, часто потребує суттєвого доопрацювання.

Marketplace-рішення на кшталт Prom.ua або Rozetka орієнтовані на масовий продаж і пропонують готову аудиторію, але не дають змоги повноцінно керувати бізнес-процесами. Контроль над структурою даних, дизайном, логікою взаємодії та обробкою замовлень обмежений інтерфейсом платформи.

Аналіз обраних систем дозволив визначити критичні для бізнесу функції, що мають бути реалізовані у власній інформаційній системі: детальна робота з варіантами товарів, перевірка доступності, автоматизоване збереження залишків, гнучке керування промокодами, стабільний інтерфейс для адміністратора без прив’язки до сторонніх сервісів. Окрему увагу приділено критеріям автономності, масштабованості та адаптації під конкретні бізнес-процеси без необхідності використання плагінів, надлишкового функціоналу або регулярних сторонніх витрат.

1.3 Веб-платформа OpenCart

Серед безкоштовних систем для створення онлайн-магазинів OpenCart займає провідну позицію. Це CMS з відкритим кодом, що дозволяє швидко запустити повноцінну торговельну платформу з базовим функціоналом: каталогізацією товарів, підтримкою категорій, оформленням замовлень, взаємодією з кошиком і застосуванням фільтрів.

Платформа побудована на класичній моделі взаємодії користувача з вітриною. У структурі магазину передбачено: головну сторінку з банерами й рекламними слайдами, перелік категорій, інтерактивний пошук, блок популярних товарів, а також навігацію до кошика (рисунок 1.1). Кожна позиція каталогу має фото, назву, короткий опис і ціну. Оформлення замовлення відбувається у кілька етапів: додавання товару до кошика, підтвердження, вибір способу доставки та завершення операції.

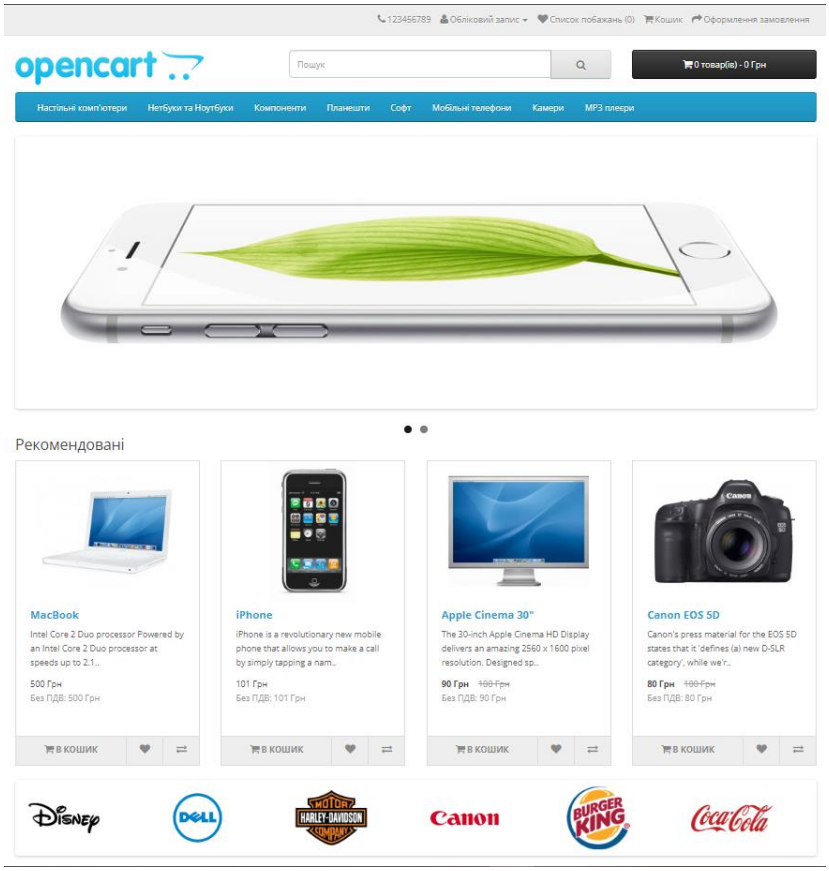

Рисунок 1.1 – Інтерфейс користувача демонстраційного магазину OpenCart [7]

У верхній частині розміщено пошуковий рядок і панель навігації з категоріями. Нижче – слайдер із банерами та блок рекомендованих товарів. Інтерфейс побудовано за принципами спрощеної візуалізації, з акцентом на швидкий доступ до основного асортименту.

Наведений приклад ілюструє типову структуру головної сторінки інтернет-магазину. Незважаючи на широкі можливості, OpenCart має обмеження щодо реалізації специфічних сценаріїв — зокрема, облік залишків на рівні варіантів товарів, гнучка логіка промокодів і фільтрація з урахуванням статі або розміру. Також виникає потреба в додаткових модулях або доопрацюваннях для реалізації функцій, що в спеціалізованій системі мають бути закладені з самого початку.

Аналіз даної платформи дозволив сформувати уявлення про мінімально необхідну функціональність, вимоги до структури інтерфейсу, спосіб подання товарів і навігації. Виявлені обмеження стали аргументом на користь розробки власної системи, що враховує особливості бізнес-процесів дистрибуції одягу та дозволяє реалізувати логіку без залучення сторонніх розширень.

2 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕСПЕЧЕНЯ

Реалізація інформаційної системи вимагає обґрунтованого вибору інструментів, мов програмування, фреймворків, бібліотек і супровідних технологій, здатних забезпечити стабільність, масштабованість і відповідність структурі проєкту. Обрані рішення повинні не лише відповідати функціональним вимогам, а й дозволяти гнучко реалізовувати прикладну логіку, підтримувати модульність і забезпечувати ефективну взаємодію між клієнтською та серверною частинами.

Особливу увагу приділено підтримці інтерактивного інтерфейсу, можливості обробки складних структур даних, підключенню бази даних з об’єктно-реляційним відображенням, забезпеченню авторизації, а також засобам контролю версій. Усі технології підібрано з урахуванням продуктивності, активної підтримки спільноти, доступної документації та відповідності сучасним стандартам розробки вебзастосунків.

2.1 Мова програмування C#

Для реалізації серверної та клієнтської логіки програмного забезпечення обрано мову програмування C# [8] — сучасну об'єктно-орієнтовану мову, розроблену компанією Microsoft, що поєднує в собі строгість типізації, зручність синтаксису, потужні засоби для роботи з даними та повну інтеграцію з платформою .NET.

Використання C# дозволяє розробляти масштабовані вебзастосунки, що мають чітку структуру, розмежування відповідальності та підтримку повторного використання коду. Мова підтримує парадигми інкапсуляції, наслідування, поліморфізму та асинхронного програмування, що особливо важливо для розробки продуктивних вебінтерфейсів з інтерактивною поведінкою.

У рамках проєкту C# є основним інструментом реалізації бізнес-логіки: перевірки стану замовлень, застосування промокодів, обробки дій користувача,

керування правами доступу, взаємодії з базою даних через ORM-технологію Entity Framework Core. Завдяки статичній типізації досягається підвищений рівень надійності коду, а підтримка інструментів рефакторингу у Visual Studio забезпечує зручність супроводу й розширення системи.

Мова надає розвинену систему обробки виключень, що дозволяє будувати стабільні механізми контролю помилок і передбачати обробку граничних ситуацій, наприклад, нестачі товару на складі або помилкового промокоду під час оформлення замовлення.

2.2 Фреймворк ASP.NET Core Blazor Server

Фреймворк Blazor Server [9], що входить до складу платформи ASP.NET Core [10], обрано основою для побудови вебінтерфейсу. Blazor дозволяє створювати інтерактивні вебзастосунки з використанням C# замість JavaScript, зберігаючи можливість компонентної структури, повторного використання коду та прямої інтеграції з серверною логікою.

У моделі Blazor Server компоненти рендеряться на сервері, а взаємодія з користувачем здійснюється через двосторонній зв’язок за допомогою SignalR [11], що забезпечує миттєву реакцію на дії користувача та мінімальні затрати на ресурсну потужність клієнта, через що підхід є придатним для створення адаптивного та динамічного інтерфейсу, навіть у середовищах з обмеженою обчислювальною потужністю на боці клієнта.

Blazor підтримує розділення інтерфейсу на компоненти — логічно ізольовані частини, що мають власну розмітку, події та параметри. Така структура застосована для побудови елементів каталогу, кошика, форми замовлення, адміністративної панелі керування товарами, категоріями та промокодами. Компонентний підхід дозволяє не лише спростити розробку, а й підвищити масштабованість системи.

Фреймворк також інтегрується з механізмами авторизації ASP.NET Core, що дає змогу реалізувати захищений доступ до адміністративних функцій без

додаткових зовнішніх засобів. Розмежування ролей, перенаправлення неавторизованих користувачів, захист адміністративного розділу – усе реалізується в межах єдиної технологічної екосистеми.

2.3 Система управління базами даних Microsoft SQL Server

У реалізації інформаційної системи використано систему управління базами даних Microsoft SQL Server [12], що забезпечує надійне зберігання, обробку та доступ до структурованих даних. SQL Server є реляційною СУБД корпоративного рівня, що підтримує масштабованість, транзакційність, інтеграцію з .NET-платформою та високий рівень безпеки.

Обрана СУБД дозволяє працювати з великим обсягом пов’язаних даних, що особливо актуально для проєкту, у якому реалізовано складну структуру таблиць: товари, категорії, варіанти розмірів, замовлення, промокоди, користувачі. Вбудовані механізми забезпечення цілісності даних, зовнішні ключі, індекси та перевірки унеможливлюють появу неконсистентної інформації та помилок при зміні даних.

Серед ключових можливостей SQL Server, що використовуються у проєкті:

– підтримка транзакцій для забезпечення цілісності під час оформлення та скасування замовлень;

– індексування таблиць з метою прискорення фільтрації, сортування та пошуку;

– реалізація зовнішніх ключів для побудови зв’язків між сутностями;

– збереження історії замовлень, залишків товарів, а також інформації про активацію промокодів.

SQL Server також забезпечує безперебійну інтеграцію з платформою .NET, що дає змогу ефективно взаємодіяти з базою через Entity Framework Core, виконувати запити у вигляді об’єктів і уникати надлишкової SQL-логіки без втрати керованості.

Для локальної розробки використано SQL Server Express, що має повну сумісність з основними компонентами повної редакції та не потребує окремої ліцензії. База даних створена з урахуванням майбутньої можливості розширення структури та додавання нових сутностей без необхідності радикального перегляду схеми.

2.4 ORM-технологія Entity Framework Core

Для взаємодії із реляційною базою даних Microsoft SQL Server застосовано технологію Entity Framework Core [13] – об’єктно-реляційне відображення, що забезпечує прямий зв’язок між об’єктами мови програмування C# та таблицями бази. Даний підхід дозволяє уникнути написання вручну SQL-запитів для типових операцій, а також значно прискорює процес розробки.

EF Core функціонує на основі моделей — C# класів, що відображають структуру відповідних таблиць бази даних. Для кожної сутності (наприклад, Product, Category, ProductVariant, Order, PromoCode) створено окрему модель, що містить набір властивостей, відповідних полям у базі. Зв’язки між сутностями реалізовано через навігаційні властивості та зовнішні ключі.

Ключовою перевагою використання EF Core є підтримка міграцій – механізму автоматичного оновлення структури бази даних на основі змін у моделях, що дозволяє поступово розвивати схему БД без втрати даних і синхронізувати кодову базу з фізичним вмістом бази на будь-якому етапі.

Доступ до даних здійснюється через об’єкт DbContext [14], що інкапсулює всі набори таблиць та правила їх конфігурації. У класі OnModelCreating визначено специфіку обмежень, унікальність, типи полів, каскадне видалення та додаткові правила. Запити до бази формуються за допомогою LINQ [15] – декларативного синтаксису для вибірки, фільтрації, сортування й об’єднання даних. Такий підхід зберігає типобезпеку, покращує читабельність коду та дає змогу інтегрувати логіку

доступу до даних у бізнес-рівень застосунку без необхідності ручного створення SQL-запитів.

Підтримка асинхронних методів (async/await) [16] дозволяє забезпечити обробку запитів без блокування потоку виконання, що є критично важливим у контексті багатокористувацької взаємодії з вебзастосунком. EF Core також забезпечує повноцінну підтримку транзакцій, що використовується для оформлення й скасування замовлень із відповідним оновленням залишків товару.

2.5 Середовище розробки Visual Studio 2022

Для реалізації інформаційної системи використано інтегроване середовище розробки Visual Studio 2022 [17], що забезпечує повний цикл створення, тестування та налагодження застосунків на платформі .NET. Visual Studio підтримує проєкти на мові C#, інтегрується з ASP.NET Core та Entity Framework Core, надаючи розширені засоби для роботи з Blazor Server.

Середовище включає інструменти автоматичного завершення коду (IntelliSense), вбудовану систему керування пакетами NuGet [18], інтерактивну панель керування міграціями бази даних, механізми відлагодження на рівні клієнтської та серверної частин, а також підтримку роздільного запуску компонентів проєкту.

Visual Studio також інтегрується з Git, що дозволяє безпосередньо з середовища розробки працювати з системою контролю версій: створювати гілки, відстежувати зміни, виконувати коміти та злиття. Це забезпечує зручну організацію командної або індивідуальної роботи з кодовою базою.

Фреймворк Blazor Server підтримується безпосередньо у Visual Studio, що дозволяє запускати і налагоджувати компоненти інтерфейсу без залучення зовнішніх засобів. Також підтримується робота з Razor-компонентами, HTML-розміткою, CSS-стилями та JavaScript-фрагментами, що використовуються для взаємодії із зовнішніми бібліотеками (наприклад, Sortable.js).

У межах проєкту Visual Studio використовувалася для створення структури застосунку, реалізації бізнес-логіки, запуску тестування, створення міграцій, а також налагодження взаємодії з базою даних через вбудовані інструменти.

2.6 Система контролю версій Git та хостинг GitHub

Для зберігання, контролю змін та координації етапів розробки програмного забезпечення використано систему контролю версій Git [19] у поєднанні з віддаленим репозиторієм GitHub [20]. Таке рішення дозволяє організувати структурований і безпечний процес розробки із збереженням усієї історії змін, можливістю відновлення попередніх версій та незалежної розробки окремих функціональних модулів.

Git реалізує розподілений принцип зберігання, де кожна копія репозиторію містить повну історію проєкту. Це забезпечує автономність роботи з кодовою базою та можливість створення окремих гілок для реалізації функціональності, тестування чи виправлення помилок без впливу на основну версію застосунку. Після завершення розробки окремих фрагментів коду зміни об’єднуються з основною гілкою через операції злиття, що дозволяє уникнути конфліктів та втрати даних.

У рамках проєкту Git використовувався для поетапного збереження логіки створення моделей, компонентів, сервісів, адміністративної панелі та механізмів перевірки при оформленні замовлення. Репозиторій проєкту розміщено на платформі GitHub, що забезпечує централізований доступ, можливість відстеження змін, ведення документації, а також спрощує демонстрацію результатів розробки.

Середовище Visual Studio 2022 інтегровано з Git, що дозволяє виконувати всі основні дії безпосередньо у межах редактора: створення комітів, перегляд відмінностей між версіями, перемикання між гілками, злиття та відправлення змін у віддалений репозиторій.

2.7 Додаткові інструменти

У процесі реалізації інтерфейсної частини застосунку використано низку допоміжних засобів і бібліотек, що забезпечують покращену взаємодію з користувачем та розширюють функціональні можливості стандартного середовища Blazor Server.

Для організації елементів сторінки, побудови макету, форм і таблиць застосовуються засоби HTML [21] та CSS [22]. HTML використовується для опису структури інтерфейсу – списків товарів, форм введення, розділів каталогу. CSS забезпечує стилізацію елементів: розміри, кольори, відступи, адаптивність до різних пристроїв. Адаптивна верстка реалізується за допомогою медіазапитів і гнучких сіток, що забезпечують коректне відображення на мобільних і десктопних екранах.

Для реалізації сортування елементів інтерфейсу шляхом перетягування використано JavaScript-бібліотеку Sortable.js [23], що інтегрується з Blazor через виклики JS-функцій. Застосування даного інструменту забезпечує покращення зручності керування вмістом у панелі адміністратора.

Усі згадані бібліотеки та інструменти інтегруються безпосередньо у структуру Razor-компонентів, що дозволяє об’єднати розмітку, стилізацію та логіку взаємодії в межах єдиного середовища розробки.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Програмне забезпечення реалізовано з урахуванням архітектурної цілісності, розмежування логіки на клієнтську та серверну частини, а також модульного поділу функціоналу. Система містить набір компонентів, що відповідають за каталогізацію товарів, обробку замовлень, керування складом, застосування промокодів, адміністрування вмісту та авторизацію користувачів.

Кожен компонент побудовано з урахуванням повторного використання коду, ізоляції відповідальності та розширюваності. Дані зберігаються у структурованій базі з об’єктно-реляційним відображенням, а взаємодія між частинами реалізується за допомогою служб, що абстрагують бізнес-логіку.

Представлено структуру модулів, моделі даних, логіку основних функціональних блоків та загальні принципи побудови взаємозв’язків між об'єктами системи.

3.1 Структура системи

Архітектура системи побудована за принципом клієнт–серверної взаємодії з використанням фреймворку Blazor Server. Усі компоненти функціонують у межах єдиного вебзастосунку, що дозволяє забезпечити тісну інтеграцію між користувацьким інтерфейсом, серверною логікою й доступом до бази даних.

Система умовно поділяється на три логічні рівні:

– презентаційний рівень, що реалізує інтерфейс користувача через Razor-компоненти;

– рівень бізнес-логіки, який відповідає за обробку дій користувача, перевірки, застосування промокодів, керування кошиком і замовленнями;

– рівень доступу до даних, що включає моделі сутностей, контекст бази даних та засоби роботи з Entity Framework Core.

Інтерфейс взаємодії реалізовано у вигляді розділів, кожен з яких відповідає за окремий аспект функціональності: перегляд товарів, роботу з кошиком, оформлення замовлення, перегляд історії, адміністративне керування товарами, категоріями, промокодами та залишками товару.

Серверна частина реалізує логіку збереження, обробки та перевірки даних, взаємодіє з SQL Server через ORM-технологію. Усі взаємозв'язки між сутностями забезпечуються через об'єкти моделей і конфігурацію в DbContext.

Для забезпечення коректної авторизації реалізовано розмежування ролей: авторизований користувач може переглядати історію замовлень, використовувати промокоди, в свою чергу адміністратор має доступ до керування вмістом і перегляду всієї системної інформації.

Застосування архітектурного поділу на рівні забезпечує легке масштабування функціоналу, гнучкість супроводу коду й можливість поступової інтеграції нових модулів. Компоненти системи взаємодіють через сервісні інтерфейси, що дозволяє уніфікувати логіку обробки замовлень, застосування промокодів та оновлення складу без дублювання коду. Такий підхід також спрощує написання автоматизованих тестів і перевірку окремих блоків системи.

Взаємодію користувача, адміністратора та системи, а також загальні сценарії використання реалізовано у вигляді діаграми прецедентів, поданої в [Додатку А].

3.2 Структура бази даних

База даних побудована за реляційною моделлю з використанням системи управління базами даних Microsoft SQL Server. Структура забезпечує зберігання, зв'язність та цілісність даних, що стосуються товарів, замовлень, категорій, варіантів товарів, користувачів, промокодів і відповідних зв'язків між ними.

Взаємозв'язки між основними сутностями реалізовано через зовнішні ключі, що забезпечують логічну зв'язність і цілісність даних у межах всієї структури. Загальну схему таблиць і їхніх зв'язків наведено нижче (рисунок 3.1).

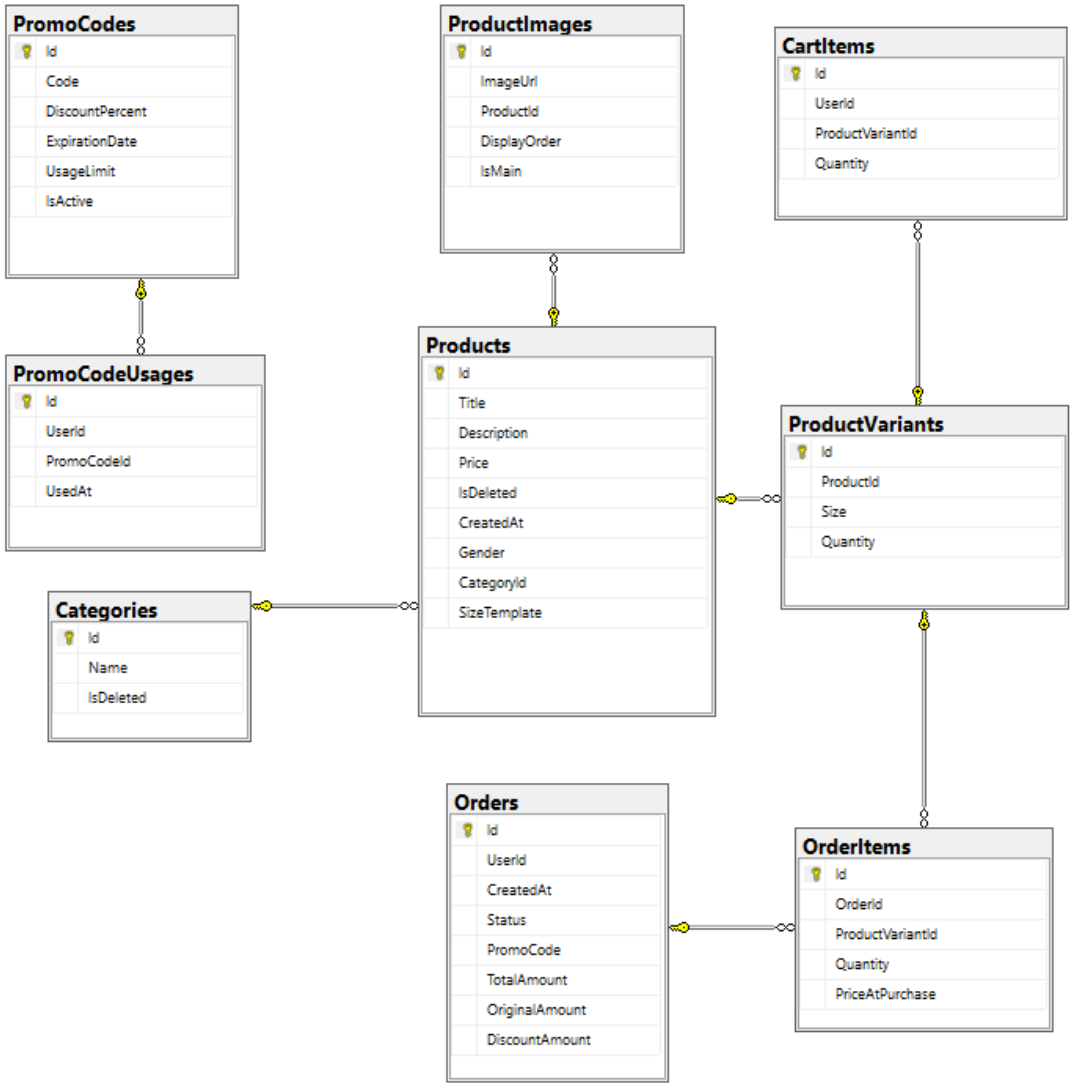

Рисунок 3.1 – Структура бази даних системи

Модель бази даних включає наступні сутності, які розглянемо далі.

3.2.1 Сутність Product

Сутність Product відповідає за збереження основної інформації про товар. Кожен запис містить назву, опис, ціну, стать, шаблон розміру, ознаку видалення, дату створення, а також зовнішній ключ на категорію. Додатково передбачено списки пов’язаних зображень та варіантів товару.

Основні поля:

– Id – первинний ключ, унікальний ідентифікатор товару;

– Title – назва товару (обов’язкове поле);

– Description – текстовий опис товару (обов’язкове поле);

– Price – вартість товару, значення повинно бути більшим за 0;

– IsDeleted – логічний прапорець м’якого видалення [24];

– CreatedAt – дата й час створення запису;

– Gender – цільова стать: чоловіча, жіноча або унісекс;

– SizeTemplate – вказує, чи товар має розмірну сітку (наприклад, одяг, взуття);

– CategoryId – зовнішній ключ, що пов’язує товар з категорією;

– Category – навігаційна властивість для доступу до пов’язаної категорії;

– Images – колекція зображень товару (ProductImage);

– Variants – колекція варіантів товару (ProductVariant), наприклад, за розмірами.

Кожен товар може мати кілька зображень, одне з яких може бути позначене основним (IsMain), а також набір варіантів, що визначають наявні розміри та кількість на складі.

3.2.2 Сутність Category

Сутність Category використовується для групування товарів за певними ознаками – наприклад, одяг, взуття, аксесуари. Вона є базовою складовою структури каталогу й дозволяє організувати навігацію та фільтрацію товарів у клієнтській частині інтерфейсу.

Основні поля:

– Id – первинний ключ, унікальний ідентифікатор категорії;

– Name – назва категорії, обов’язкове поле, обмежене 100 символами;

– IsDeleted – логічна ознака м’якого видалення, що дозволяє приховати категорію з інтерфейсу без фізичного видалення з бази;

– Products – навігаційна властивість, що вказує на список товарів, пов’язаних із даною категорією.

Усі товари в системі обов’язково прив’язуються до певної категорії. Така модель дозволяє підтримувати узгодженість структури каталогу, фільтрувати асортимент за категоріями та забезпечувати коректну організацію інтерфейсу як на публічній сторінці, так і в адміністративній панелі.

3.2.3 Сутність ProductVariant

Сутність ProductVariant використовується для зберігання варіантів товарів за розмірами та їх кількості. Вона дозволяє представити один товар у кількох розмірних модифікаціях, кожна з яких має окрему кількість одиниць на складі. Такий підхід особливо актуальний для дистрибуції одягу та взуття, де для одного товару можливі численні розмірні варіанти.

Основні поля:

– Id – первинний ключ, унікальний ідентифікатор варіанту;

– ProductId – зовнішній ключ, що вказує на товар, до якого належить варіант;

– Product – навігаційна властивість для доступу до відповідного товару;

– Size – текстове значення розміру (наприклад: "S", "M", "L", "42", "44" тощо);

– Quantity – кількість одиниць даного розміру на складі, не може бути від’ємною.

Завдяки наявності даної сутності система може відстежувати залишки кожного розміру окремо, обмежувати оформлення замовлень залежно від наявності, а також забезпечувати гнучкість при відображенні доступних опцій у клієнтському інтерфейсі. У разі скасування замовлення або його редагування передбачено оновлення кількості відповідних варіантів товару.

3.2.4 Сутність Order

Сутність Order відповідає за зберігання даних про замовлення, створені користувачами. Кожен запис містить інформацію про покупця, список товарів, що були замовлені, застосований промокод, а також суму замовлення з урахуванням знижки та поточний статус обробки.

Основні поля:

- Id – первинний ключ, унікальний ідентифікатор замовлення;
- UserId – ідентифікатор користувача, що здійснив замовлення;
- CreatedAt – дата й час створення замовлення;
- Status – статус замовлення (очікує обробки, в обробці, відправлено, завершено, скасовано); значення представлені як перелік enum;
- Items – список позицій у межах замовлення (OrderItem);
- PromoCode – текстовий код застосованого промокоду (якщо є);
- OriginalAmount – загальна сума до застосування знижки;
- DiscountAmount – надана сума знижки;
- TotalAmount – остаточна сума до сплати після врахування знижки.

Сутність передбачає гнучку структуру для роботи з акційними механізмами (через поле PromoCode), дозволяє контролювати динаміку статусів і підтримує відстеження кожної зміни — від моменту створення до завершення або скасування замовлення. Завдяки наявності зв’язку з OrderItem, кожне замовлення може містити довільну кількість товарів з вказанням кількості та обраного варіанту.

3.2.5 Сутність OrderItem

Сутність OrderItem описує окрему позицію в межах замовлення. Містить посилання на обраний варіант товару (ProductVariant), кількість одиниць, що були замовлені, та зафіксовану ціну на момент покупки. Така структура дозволяє зберігати історію замовлень незалежно від змін цін або залишків у майбутньому.

Основні поля:

– Id – первинний ключ, унікальний ідентифікатор позиції замовлення;

– OrderId – зовнішній ключ, що вказує на замовлення, до якого належить позиція;

– Order – навігаційна властивість до відповідного замовлення;

– ProductVariantId – зовнішній ключ, що вказує на обраний варіант товару;

– ProductVariant – навігаційна властивість до відповідного варіанту;

– Quantity – кількість одиниць товару у позиції;

– PriceAtPurchase – ціна одиниці товару на момент замовлення.

Наявність окремої сутності OrderItem забезпечує гнучкість у формуванні складних замовлень, що складаються з кількох різних товарів і варіантів. Також дозволяє обчислювати загальну суму, зберігати історію транзакцій і здійснювати точний контроль залишків.

3.2.6 Сутність PromoCode

Сутність PromoCode використовується для збереження інформації про знижкові коди, що можуть бути застосовані користувачами під час оформлення замовлення. Кожен промокод має унікальний код, відсоток знижки, а також може мати обмеження за терміном дії або кількістю використання.

Основні поля:

– Id – первинний ключ, унікальний ідентифікатор промокоду;

– Code – унікальне текстове значення промокоду, що вводить користувач;

– DiscountPercent – відсоток знижки, що застосовується до замовлення; допустимі значення: від 1% до 100%;

– ExpirationDate – дата завершення дії промокоду; якщо не вказана — промокод не має строку дії;

– UsageLimit – максимальна кількість використань; якщо значення не задано, промокод вважається безлімітним;

– IsActive – логічне поле, що визначає, чи доступний промокод для застосування на поточний момент.

Система перевіряє чинність промокоду на момент його застосування: активність, строк дії, ліміт використань. Після успішного використання кількість активацій оновлюється. Таким чином, PromoCode дозволяє реалізувати гнучку політику знижок та керування маркетинговими кампаніями.

3.2.7 Сутність PromoCodeUsage

Сутність PromoCodeUsage зберігає інформацію про факти використання промокодів конкретними користувачами. Вона дозволяє обмежити повторне використання одного і того ж коду та відстежити історію його застосування.

Основні поля:

– Id – первинний ключ, унікальний ідентифікатор запису;

– UserId – ідентифікатор користувача, використав промокод;

– PromoCodeId – зовнішній ключ до таблиці PromoCode;

– PromoCode – навігаційна властивість до об’єкта промокоду;

– UsedAt – дата та час використання промокоду.

Дана таблиця використовується для перевірки, чи має користувач право ще раз скористатися конкретним промокодом, якщо для нього встановлено обмеження в один або кілька разів. Завдяки цій сутності система забезпечує перевірку індивідуальної історії використання промокодів. Кожне застосування фіксується незалежно від того, чи промокод є одноразовим чи багаторазовим, що дозволяє гнучко обмежувати кількість використань не лише загалом, а й на рівні окремого користувача.

3.2.8 Сутність CartItem

Сутність CartItem призначена для зберігання даних про поточний вміст кошика користувача до моменту оформлення замовлення. Вона відображає проміжний стан вибраних товарів і використовується для формування замовлення в процесі підтвердження.

Основні поля:

- Id – первинний ключ, унікальний ідентифікатор запису;
- UserId – ідентифікатор користувача, якому належить кошик;
- ProductVariantId – зовнішній ключ до конкретного варіанту товару;
- ProductVariant – навігаційна властивість для доступу до товару;
- Quantity – кількість одиниць вибраного варіанту.

Кожен користувач має власний набір позицій у кошику, що не зберігаються у вигляді замовлення до моменту підтвердження. Зміна кількості, видалення або перевірка доступності товару відбувається безпосередньо в межах списку. Таким чином, CartItem є ключовою частиною логіки попереднього перегляду й редагування замовлення до його остаточного оформлення.

3.2.9 Сутність ProductImage

Сутність ProductImage зберігає інформацію про зображення, пов’язані з конкретним товаром. Вона забезпечує можливість прикріплення кількох зображень до одного товару, визначення основного з них та встановлення порядку їх відображення.

Основні поля:

- Id – первинний ключ, унікальний ідентифікатор зображення;
- ImageUrl – шлях до файлу зображення;
- ProductId – зовнішній ключ, що вказує на відповідний товар;
- Product – навігаційна властивість до сутності Product;
- DisplayOrder – порядок відображення зображень на сторінці товару;

– IsMain – логічне поле, яке позначає основне зображення товару.

Сутність дозволяє організувати гнучке керування медіаконтентом у межах адміністративного інтерфейсу. Завдяки використанню поля DisplayOrder реалізовано сортування зображень за допомогою drag-and-drop механізму. Основне зображення (IsMain = true) використовується для представлення товару в каталозі та при попередньому перегляді.

3.3 Взаємодія з базою даних через ORM

Доступ до бази даних реалізовано за допомогою технології Entity Framework Core, що забезпечує об’єктно-реляційне відображення сутностей. Центральним елементом виступає клас ApplicationDbContext, який успадковує IdentityDbContext та містить визначення таблиць бази через властивості DbSet<> (рисунок 3.2).

```
11 references
public DbSet<Product> Products { get; set; }
15 references
public DbSet<Category> Categories { get; set; }
5 references
public DbSet<ProductImage> ProductImages { get; set; }
5 references
public DbSet<Order> Orders { get; set; }
0 references
public DbSet<OrderItem> OrderItems { get; set; }
14 references
public DbSet<CartItem> CartItems { get; set; }
8 references
public DbSet<PromoCode> PromoCodes { get; set; }
6 references
public DbSet<PromoCodeUsage> PromoCodeUsages { get; set; }
6 references
public DbSet<ProductVariant> ProductVariants { get; set; }
```

Рисунок 3.2 – Визначення таблиць бази даних

У методі OnModelCreating реалізовано конфігурацію моделей через Fluent API. Встановлено точність для полів ціни, унікальність значень (PromoCode.Code, Category.Name, Product.Title), фільтрацію м’яко видалених записів, а також обмеження на довжину деяких полів (рисунок 3.3).

```
modelBuilder.Entity<Category>()
    .HasIndex(c => c.Name)
    .IsUnique();

modelBuilder.Entity<Product>()
    .HasQueryFilter(p => !p.IsDeleted);

modelBuilder.Entity<OrderItem>()
    .Property(p => p.PriceAtPurchase)
    .HasPrecision(18, 2);
```

Рисунок 3.3 – Конфігурація моделей БД

Запити до бази формуються за допомогою LINQ, що забезпечує типобезпеку та зручну інтеграцію з бізнес-логікою. У процесі оформлення замовлення система перевіряє залишки товарів, застосовує знижку, фіксує актуальні ціни та формує зв’язки між замовленням і його позиціями. При скасуванні замовлення передбачено повернення кількості на склад.

Для контролю змін у структурі бази застосовуються міграції. Кожна зміна у моделях супроводжується створенням нової версії міграції, що дозволяє поступово оновлювати схему без втрати даних і дає змогу підтримувати актуальність бази як на етапі розробки, так і при розгортанні на сервері.

3.4 Реалізація ролей і доступу

У системі реалізовано базову модель авторизації з розмежуванням доступу для двох основних типів користувачів: звичайного користувача та адміністратора. Основою для реалізації прав доступу виступає механізм аутентифікації та авторизації, вбудований у фреймворк ASP.NET Core із використанням технології Identity [26].

Користувачі проходять реєстрацію та вхід до системи за допомогою стандартних засобів автентифікації. Рольова модель будується на призначенні

ролей User та Admin. Відповідно до належності до певної ролі, надається або обмежується доступ до функціональності та сторінок вебзастосунку.

Адміністративна частина (наприклад, панель керування товарами, категоріями, замовленнями) захищена за допомогою атрибута [Authorize(Roles = "Admin")] (рисунок 3.4). Даний механізм обмежує доступ до Razor-сторінок або компонентів лише для авторизованих користувачів з відповідною роллю.

```
@attribute [Authorize(Roles = "Admin")]
```

Рисунок 3.4 – Атрибут авторизації

Для звичайних користувачів доступні сторінки з каталогом, кошиком, оформленням замовлення, а також історією замовлень. Авторизація на рівні користувача (User) необхідна для перегляду особистої історії замовлень та застосування промокодів.

Ролі зберігаються у базі даних у рамках механізму ASP.NET Core Identity. Їх призначення виконується вручну на етапі ініціалізації системи. Доступ до чутливих даних, таких як списки користувачів, статуси замовлень, товарні залишки, категорично заборонено для звичайних користувачів.

Для захисту від несанкціонованого доступу система автоматично перенаправляє неавторизованих користувачів на сторінку входу. Внутрішня перевірка ролей відбувається як на рівні маршрутизації, так і в логіці сервісів при спробі виконання критичних дій.

Така модель авторизації дозволяє ефективно контролювати доступ до функцій системи, забезпечує безпеку адміністрування та захищає дані від несанкціонованих змін. Розмежування ролей реалізовано без надмірної складності, що сприяє стабільній роботі застосунку та зручності його підтримки.

3.5 Клієнтські компоненти Blazor

Інтерфейс системи побудовано за принципом логічного розділення функціональності відповідно до ролей користувачів. Реалізація здійснена на основі компонентної моделі Blazor Server, що забезпечує інтерактивність і розширюваність вебінтерфейсу. Кожна функціональна область винесена в окремі Razor-компоненти, згруповані за напрямами: для клієнта та адміністратора.

3.5.1 Публічний інтерфейс користувача

Публічна частина системи побудована на основі Razor-компонентів і призначена для взаємодії звичайного користувача з каталогом товарів, кошиком, історією замовлень та окремими сторінками товарів. Всі компоненти інтегровані у структуру навігації, доступну без авторизації або після входу в систему.

Основні сторінки клієнтського інтерфейсу:

– Catalog.razor – головна сторінка з переглядом товарів та можливістю фільтрації за категоріями, пошуку, діапазоном цін;

– ProductDetails.razor – детальний перегляд товару з описом, вибором розміру, зображеннями та кнопкою додавання до кошика;

– Cart.razor – управління товарами у кошику, зміна кількості, видалення позицій, введення промокоду та підтвердження замовлення;

– OrderSuccess.razor – повідомлення про успішне оформлення замовлення; Orders.razor — перегляд історії власних замовлень;

– OrderDetails.razor – деталізація окремого замовлення зі статусом, списком товарів та підсумковими сумами.

Оформлення замовлення виконується без окремої сторінки підтвердження. Уся необхідна інформація вводиться безпосередньо в компоненті Cart.razor. Після успішної обробки користувач перенаправляється на сторінку OrderSuccess.razor.

Взаємодія між компонентами відбувається через параметри, callback-події та сервіси. Сторінки працюють без перезавантаження, що забезпечує інтерактивність і зручність навігації.

3.5.2 Адміністративна частина

Адміністративний інтерфейс реалізовано окремими компонентами, що згруповані у папку Admin і доступні лише користувачам з роллю Admin. Усі маршрути захищено через атрибут [Authorize(Roles = "Admin")], що виключає несанкціонований доступ до функцій керування.

До адміністративної частини належать такі компоненти:

– AdminProducts.razor – таблиця з переліком товарів, можливістю їх редагування та приховування;

– AddProduct.razor, EditProduct.razor – форми для створення й оновлення інформації про товар;

– AdminCategories.razor – перегляд і редагування категорій, підтримка м’якого видалення;

– AddCategory.razor, EditCategory.razor – окремі компоненти для керування назвами категорій;

– AdminStock.razor – оновлення залишків товарів по розмірах (ProductVariant);

– AdminOrders.razor – перегляд усіх замовлень, зміна їх статусу, доступ до деталей;

– AdminOrderDetails.razor – перегляд повної інформації про конкретне замовлення;

– AdminPromoCodes.razor – управління промокодами, фільтрація, активація, деактивація;

– AddPromoCode.razor – форма для створення нового промокоду;

– AdminUsers.razor – перегляд користувачів (при потребі), доступ до історії;

– AdminUserDetails.razor – перегляд інформації про конкретного користувача.

Компоненти адміністрування реалізовані з урахуванням потреб керування контентом, складу, знижками та замовленнями. Дані завантажуються асинхронно, застосовано локальні фільтри, повідомлення про помилки, drag-and-drop сортування зображень і підтвердження дій.

3.6 Сервіс роботи з кошиком CartService

Ключовим елементом бізнес-логіки у клієнтській частині є сервіс CartService [Додаток Б], що реалізує повний цикл взаємодії з кошиком користувача: від додавання товару до підтвердження замовлення. Сервіс інкапсулює логіку обробки товарів у кошику, перевірки залишків, застосування знижок, створення замовлення та зменшення кількості на складі.

Використано підхід IDbContextFactory<ApplicationDbContext>, що дозволяє уникати конфліктів у багатопотоковому середовищі Blazor Server. Усі методи є асинхронними, що забезпечує стабільність при великій кількості користувачів.

Основні функції сервісу:

– додавання товару до кошика (AddToCartAsync) – перевіряє наявність позиції та збільшує кількість, або створює новий запис;

– отримання вмісту кошика (GetItemsAsync) – повертає всі позиції користувача з навігаційними властивостями ProductVariant → Product → Category;

– обчислення підсумкової суми (GetTotalAsync) – повертає загальну вартість товарів у кошику без урахування знижки;

– оновлення кількості (UpdateQuantityAsync) – дозволяє змінити кількість або видалити товар;

– видалення окремого товару або очищення всього кошика (RemoveFromCartAsync, ClearCartAsync);

– підтвердження замовлення (PlaceOrderAsync) – здійснює перевірку наявності товарів на складі, застосування промокоду, розрахунок знижки, створення замовлення з відповідними позиціями та зменшення залишків;

– розрахунок знижки без створення замовлення (CalculateTotalWithDiscountAsync) – використовується для попереднього перегляду підсумкової суми;

– підрахунок загальної кількості товарів у кошику (GetCartCountAsync) – застосовується для динамічного відображення кількості у шапці сайту.

Сервіс також містить логіку перевірки промокодів (рисунок 3.5): враховується їх активність, строк дії, ліміт використань та унікальність застосування для кожного користувача. Після успішного оформлення замовлення відповідний запис про використання коду (PromoCodeUsage) додається до бази даних.

```
var order = new Order
{
    UserId = userId,
    CreatedAt = DateTime.Now,
    Status = OrderStatus.Pending,
    PromoCode = !string.IsNullOrWhiteSpace(promoCode) ? promoCode : null,
    TotalAmount = finalAmount,
    OriginalAmount = total,
    DiscountAmount = discountAmount,
    Items = items.Select(i => new OrderItem
    {
        ProductVariantId = i.ProductVariantId,
        Quantity = i.Quantity,
        PriceAtPurchase = i.ProductVariant.Product.Price
    }).ToList()
};
```

Рисунок 3.5 – Оформлення нового замовлення

У даному фрагменті:

– перевіряється достатність залишків на складі;

– обчислюється остаточна сума замовлення;

– створюється об’єкт Order, включаючи всі позиції;

– додається інформація про використаний промокод (якщо він валідний і дозволений).

Такий підхід дозволяє централізовано реалізувати перевірки, зберегти узгодженість даних та забезпечити коректну логіку знижок.

3.7 Адміністративна функціональність

Адміністративна частина системи забезпечує повноцінне керування вмістом: товарами, категоріями, замовленнями, залишками та промокодами. Всі компоненти згруповано у директорії /Admin, і доступ до них дозволено лише користувачам із роллю Admin. Захист реалізовано за допомогою атрибута [Authorize(Roles = "Admin")], що гарантує обмеження доступу до функцій керування системою.

Розмежування компонентів реалізовано з урахуванням логічного поділу за функціональністю. У підпунктах нижче розглянуто основні можливості адміністративного інтерфейсу.

3.7.1 Керування товарами та категоріями

Компоненти AdminProducts.razor, AddProduct.razor, EditProduct.razor відповідають за створення, редагування та м’яке видалення товарів. Кожен товар має обов’язкові атрибути: назву, опис, категорію, ціну, варіанти розмірів (ProductVariant) та галерею зображень (ProductImage). М’яке видалення реалізовано через логічне поле IsDeleted. Встановлення прапорця у значення true приховує товар із каталогу для користувачів, але не порушує цілісності історичних замовлень, що включають відповідну позицію.

Керування категоріями відбувається через компоненти AdminCategories.razor, AddCategory.razor та EditCategory.razor. Система підтримує унікальність назв (на рівні бази даних), м’яке видалення категорій та фільтрацію активних категорій у клієнтському інтерфейсі, що дозволяє уникнути помилок при редагуванні, а також забезпечує коректне групування товарів без втрати пов’язаних даних.

Для керування зображеннями товару реалізовано функціонал сортування методом drag & drop [27] (рисунок 3.6) з використанням JavaScript-бібліотеки Sortable.js. Впорядкування здійснюється динамічно у візуальному інтерфейсі, а новий порядок передається у компонент Blazor через DotNetHelper.

```
window.initSortable = (dotnetHelper) => {
    const el = document.getElementById("sortable-images");
    if (!el) return;

    new Sortable(el, {
        animation: 150,
        onEnd: () => {
            const orderedIds = Array.from(el.children).map(c => parseInt(c.dataset.id));
            dotnetHelper.invokeMethodAsync("UpdateImageOrder", orderedIds);
        }
    });
};
```

Рисунок 3.6 – JavaScript код для сортування

Серверна логіка відповідає за оновлення поля DisplayOrder у сутності ProductImage, що визначає порядок виведення зображень на сторінці. Окреме логічне поле IsMain позначає головне зображення, яке відображається в переліку товарів і на картці детального перегляду. Такий підхід дозволяє забезпечити зручність адміністрування без перезавантаження сторінки та зберегти налаштування порядку при наступному відображенні.

3.7.2 Управління складом

Компонент AdminStock.razor забезпечує зміну залишків товарів у розрізі розмірів (ProductVariant). Інтерфейс дозволяє швидко оновлювати кількість одиниць товару, відображає поточні значення, реагує на зміни статусів. Всі зміни зберігаються в реальному часі з перевірками на валідність.

Перед оформленням замовлення система автоматично перевіряє доступність кожної позиції у кошику. Якщо залишку недостатньо, замовлення не створюється, а користувач отримує повідомлення з переліком відсутніх товарів.

3.7.3 Робота із замовленнями

Компонент AdminOrders.razor надає адміністраторам можливість переглядати всі замовлення в системі, здійснювати фільтрацію за статусом, змінювати статус обробки, а також відкривати сторінку з детальною інформацією (AdminOrderDetails.razor). Інтерфейс реалізовано у вигляді таблиці з кнопками переходу до перегляду, формою зміни статусу та кольоровими індикаторами для різних етапів виконання.

Під час зміни статусу на "Скасовано" (OrderStatus.Canceled) система автоматично повертає відповідну кількість товарів на склад (рисунок 3.7), що дозволяє підтримувати узгодженість між фактичними залишками товару та базою даних.

```
private async Task ApplyStatusChange()
{
    if (order == null)
        return;

    var newStatus = selectedStatus;

    if (!GetAllowedTransitions(order.Status).Contains(newStatus))
        return;

    if (newStatus == OrderStatus.Canceled && order.Status != OrderStatus.Canceled)
    {
        foreach (var item in order.Items)
        {
            var variant = await Db.ProductVariants.FindAsync(item.ProductVariantId);
            if (variant != null)
            {
                variant.Quantity += item.Quantity;
            }
        }
    }

    order.Status = newStatus;
    await Db.SaveChangesAsync();
}
```

Рисунок 3.7 – Реалізація логіки повернення товару на склад

Сторінка з деталями замовлення (AdminOrderDetails.razor) відображає:

- ідентифікатор замовлення та користувача;
- дату створення;
- застосований промокод (якщо був);
- повний перелік товарів із кількістю, ціною та розміром;

– поточний статус виконання та історію змін.

Реалізація дозволяє адміністраторам ефективно керувати життєвим циклом замовлень і вчасно реагувати на потребу в оновленні залишків.

3.7.4 Керування промокодами

Промокоди створюються в AddPromoCode.razor, а керування ними відбувається через AdminPromoCodes.razor. Система дозволяє:

– встановлювати відсоток знижки (1–100%);

– обирати термін дії;

– задавати ліміт використань;

– перемикати активність (IsActive).

Передбачено перевірку унікальності коду (на основі індексу у базі) та обмеження мінімальної довжини ([MinLength(3)]). Усі активації зберігаються у таблиці PromoCodeUsages.

Система також перевіряє, чи вже використовував користувач конкретний промокод, та враховує перевищення ліміту.

3.8 Перевірки, валідація та UX-рішення

Надійність і зручність використання програмної системи значною мірою залежать від реалізації перевірок введення, валідації даних, дотримання бізнес-правил та інтерфейсної взаємодії. У проєкті особливу увагу приділено трьом напрямам: перевірці коректності введених значень, логіці обробки бізнес-подій та наданню користувачу зрозумілого зворотного зв’язку.

3.8.1 Валідація введення

На рівні моделей даних використовуються вбудовані атрибути DataAnnotations [28], що визначають обов’язковість заповнення, діапазони значень, обмеження на довжину рядків, унікальність тощо. Наприклад, для ціни товару задано обов’язкову умову заповнення та мінімальне значення (рисунок 3.8).

```
[Required(ErrorMessage = "Ціна обов'язкова")]
[Range(0.01, double.MaxValue, ErrorMessage = "Ціна повинна бути більше 0")]
30 references
public decimal Price { get; set; }
```

Рисунок 3.8 – Умова встановлення ціни товару

Валідація реалізована як на серверній стороні, так і на клієнті через компоненти Blazor, що забезпечує швидке виявлення помилок ще до надсилання форми. У випадку некоректного заповнення форми користувач отримує детальне повідомлення із вказанням конкретного поля.

3.8.2 Перевірки бізнес-логіки

На рівні логіки обробки запитів реалізовано додаткові перевірки, що забезпечують цілісність та відповідність даних бізнес-умовам. До прикладів таких перевірок належать:

– автоматична відмова в оформленні замовлення, якщо у кошику наявні товари або категорії з ознакою IsDeleted;

– перевірка строку дії та обмеження кількості використань промокоду;

– перевірка залишків товарів перед створенням замовлення;

– блокування доступу до адміністративних сторінок для неавторизованих користувачів;

– повернення товарів на склад у разі скасування замовлення.

Усі перевірки винесено до відповідних сервісів або методів, що зберігає чистоту коду та уніфіковану логіку перевірок по всій системі.

3.8.3 UX-рішення та зворотний зв’язок

Для покращення взаємодії з користувачем реалізовано низку рішень, що підвищують зрозумілість, швидкість і комфорт роботи з системою. Зокрема:

– автоматичне фокусування на помилкових полях при валідації;

– блокування кнопок під час обробки запиту з індикатором завантаження;

– динамічне оновлення повідомлень про успіх або помилку;

– сортування зображень за допомогою drag & drop без перезавантаження сторінки;

– попередження про недоступність товарів або перевищення залишків перед оформленням замовлення;

– підсвічування активних фільтрів та доступних варіантів вибору.

Завдяки застосованим рішенням забезпечується інтуїтивна та надійна взаємодія користувача з системою без потреби у зовнішньому супроводі чи навчанні.

4 ДЕМОНСТРАЦІЯ ФУНКЦІОНАЛУ І ПЕРЕВІРКА ПРОГРАМНОЇ СИСТЕМИ

Функціональні можливості інформаційної системи управління замовленнями споживачів у дистрибуції одягу та відповідних аксесуарів підтверджуються практичним тестуванням у середовищі, наближеному до реальної експлуатації. Визначено технічні вимоги до розгортання вебзастосунку, послідовність інсталяції та запуску.

Описано сценарії взаємодії користувача з інтерфейсом, оформлення замовлень, застосування промокодів, керування кошиком, перегляду історії покупок, а також дії адміністратора при оновленні вмісту системи, зміні статусів замовлень, редагуванні складу й категорій. Для кожної дії наводяться умови, кроки виконання та очікуваний результат.

Результати перевірки демонструють відповідність функціоналу поставленим вимогам, стабільну роботу основних компонентів та зручність взаємодії як для клієнта, так і для адміністратора.

4.1 Інсталяція та вимоги до середовища

Для розгортання та використання інформаційної системи управління замовленнями споживачів необхідно забезпечити відповідне програмне та апаратне середовище. Вебзастосунок реалізовано за технологією Blazor Server, з використанням .NET 8, Entity Framework Core та SQL Server для зберігання даних.

4.1.1 Системні вимоги

- Операційна система: Windows 10 або новіша;
- Оперативна пам’ять: не менше 8 ГБ;

– Місце на диску: не менше 1 ГБ для розміщення бази даних і сервера застосунку;

– Процесор: будь-який із підтримкою x64-архітектури.

4.1.2 Програмне забезпечення

– .NET SDK 7.0 – для запуску й компіляції застосунку;

– Visual Studio 2022 або новіша з підтримкою ASP.NET, Razor, EF Core;

– SQL Server Express – для зберігання даних;

– Microsoft SQL Server Management Studio (SSMS) — для адміністрування БД (опційно);

– Браузер: Microsoft Edge, Google Chrome, Firefox (із підтримкою WebSockets).

4.1.3 Інструкція розгортання

— Завантажити та відкрити проєкт у Visual Studio;

— Налаштувати рядок підключення до бази даних у файлі appsettings.json;

— Відкрити Package Manager Console та виконати команду для створення бази: Update-Database

— Запустити застосунок;

— Відкрити локальну адресу, наприклад https://localhost:5001/

— Для доступу до адміністративного функціоналу увійти під обліковим записом адміністратора. Логін і пароль за замовчуванням зберігаються у файлі DbInitializer.cs.

Система підтримує одночасну роботу кількох користувачів, що забезпечується завдяки правильній реалізації службових методів та перевірено за допомогою тестових сценаріїв на локальному сервері. Використання

IDbContextFactory гарантує стабільність функціонування в умовах багатопотокової взаємодії в середовищі Blazor Server.

4.2 Сценарій взаємодії та демонстрація функціоналу

Функціональні можливості системи представлені у вигляді послідовностей дій, що охоплюють ключові етапи взаємодії з інтерфейсом. Наведені сценарії демонструють процес вибору товару, роботу з кошиком, застосування промокодів, оформлення замовлення, а також дії адміністратора щодо керування вмістом, замовленнями й складом.

Кожен сценарій відображає реальну поведінку системи у відповідь на дії користувача, що дозволяє перевірити відповідність реалізації функціональним вимогам. Для ілюстрації подано фрагменти інтерфейсу, що підтверджують коректність роботи програмних компонентів у типовому середовищі експлуатації.

4.2.1 Каталог товарів і вибір позиції

Інтерфейс головної сторінки відображає повний каталог товарів, доступний для перегляду без авторизації. У верхній частині реалізовано панель фільтрації, що дає змогу обмежити відображення товарів за ключовими параметрами: назвою, категорією, діапазоном цін. Додатково передбачено live-пошук за ключовими словами та інтерактивний ціновий повзунок для зручної навігації по асортименту (рисунок 4.1). Основна частина сторінки заповнюється картками товарів. Кожна картка містить зображення, назву, ціну та кнопку переходу до детального перегляду. При наведенні відображаються стилізовані ефекти, що забезпечують зручність взаємодії. Реалізовано адаптивну сітку з підтримкою масштабування на різних екранах.

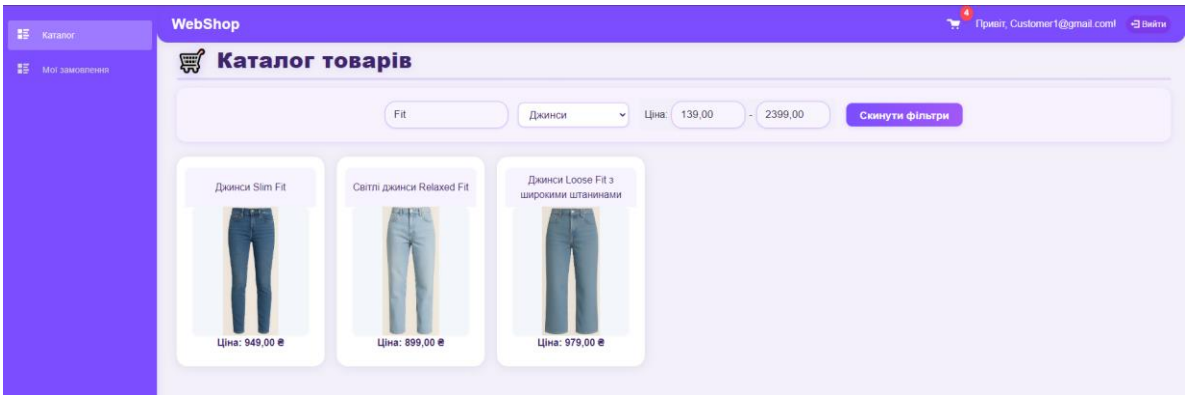

Рисунок 4.1 – Сторінка каталогу з фільтрами

У картці товару подано повну інформацію: детальний опис, ціна, вибір розміру, список зображень (рисунок 4.2). За допомогою випадаючого списку користувач обирає потрібний розмір, після чого натискає кнопку «Додати до кошика». Якщо розмір або кількість відсутні, кнопка автоматично деактивується. У разі помилки або спроби додати недоступний товар система відображає відповідне попередження.

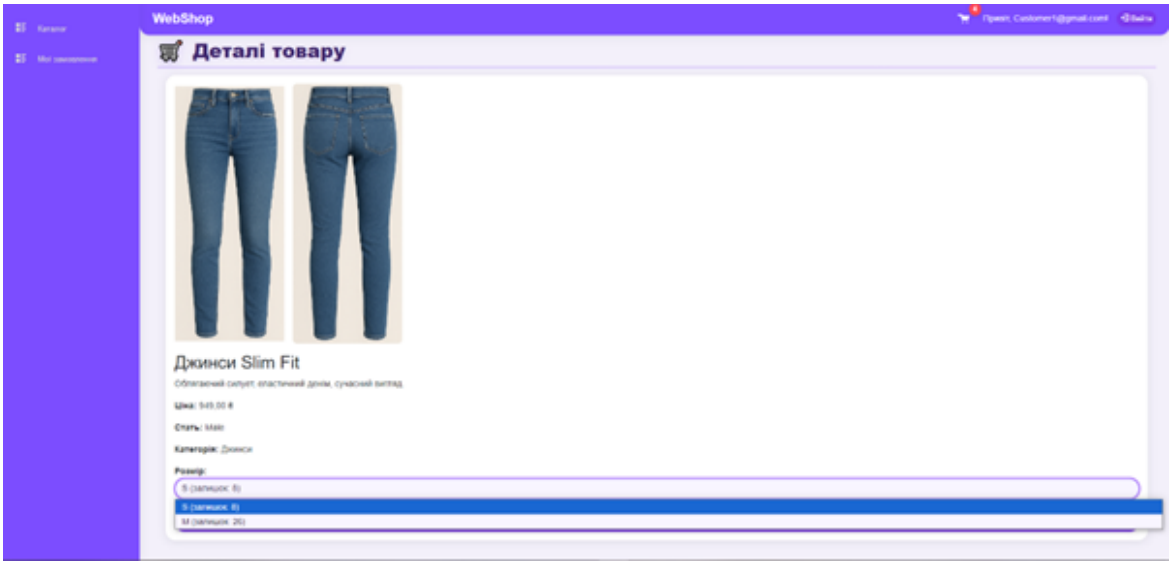

Рисунок 4.2 – Сторінка картки товару з вибором розміру

Також у випадаючому списку показується кількість кожного розміру товару.

4.2.2 Оформлення замовлення та застосування промокоду

Після додавання товарів до кошика користувач переходить до сторінки перегляду вибраних позицій. У таблиці кошика відображається перелік товарів із вказаними розмірами, кількістю, ціною за одиницю та підсумковою сумою. Передбачено можливість змінювати кількість кожної позиції або видаляти її повністю (рисунок 4.3).

Рисунок 4.3 – Кошик користувача з позиціями та кількістю

Нижче розташовано поле для введення промокоду. Після підтвердження система перевіряє:

– чи активний код;
– чи не минув термін дії;
– чи не перевищено ліміт використань;
– чи не використовувався вже даним користувачем.

У разі успішного застосування знижки система обчислює відсоткове значення знижки відповідно до вказаного у промокоді коефіцієнта, після чого автоматично перераховується загальна сума замовлення з урахуванням знижки. Користувач отримує візуальне підтвердження про застосування промокоду та оновлену фінансову інформацію щодо замовлення. У разі невдалої перевірки система відображає повідомлення з поясненням причини – наприклад, недійсність коду, вичерпання ліміту або повторне використання. Всі результати перевірки зворотно

відображаються без перезавантаження сторінки, з урахуванням інтерактивної логіки інтерфейсу (рисунок 4.4).

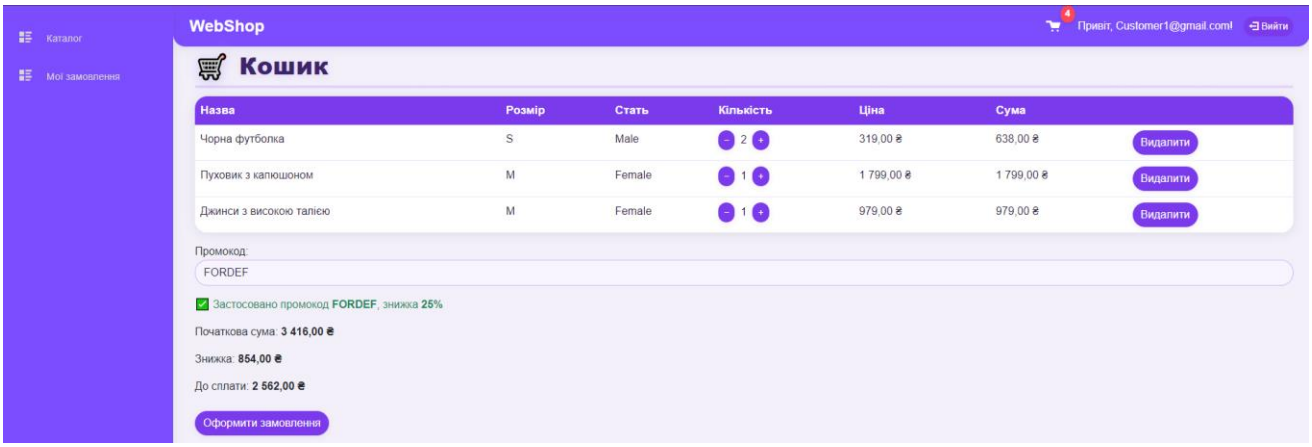

Рисунок 4.4 – Форма введення промокоду та результат перевірки

Завершення оформлення відбувається безпосередньо у кошику шляхом натискання кнопки підтвердження. У момент обробки замовлення система зберігає відповідні записи у базі даних, зменшує залишки товарів на складі відповідно до зазначених у кошику кількостей, а в разі використання промокоду – фіксує його застосування у таблиці використань. Після успішного збереження інформації користувач автоматично перенаправляється на спеціальну сторінку, де виводиться підтвердження створення замовлення та повідомлення про успішне завершення операції (рисунок 4.5).

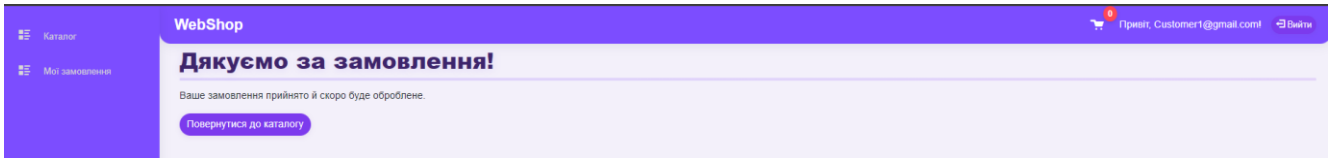

Рисунок 4.5 – Сторінка вдалого завершення замовлення

Екран підтвердження відображає результат виконання замовлення та інформує користувача про успішне завершення операції.

4.2.3 Перегляд історії замовлень

Після успішного оформлення замовлення зареєстрований користувач має змогу переглядати перелік усіх своїх покупок. На сторінці «Мої замовлення» (рисунок 4.6) відображається зведена інформація про кожне оформлене замовлення користувача: дата створення, загальна сума та поточний статус обробки. Також доступна дія «Переглянути» для переходу до розширеного перегляду.

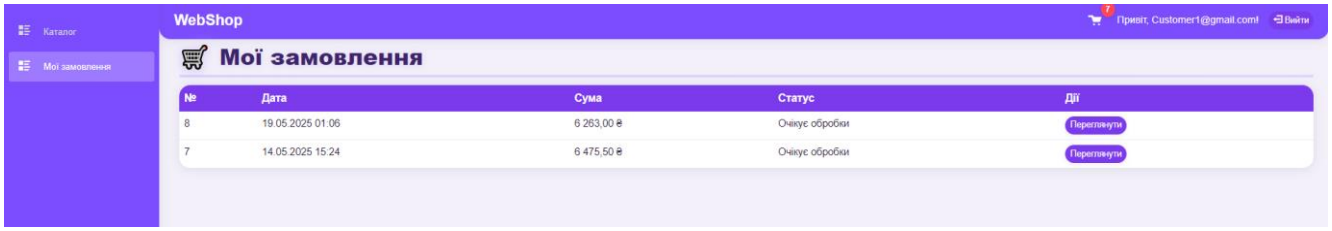

Рисунок 4.6 – Сторінка «Мої замовлення»

Передбачено деталізований перегляд кожного замовлення. Після переходу відкривається сторінка з повною інформацією (рисунок 4.7): перелік придбаних товарів із зазначенням розміру, кількості та ціни, знижка, у разі застосування промокод, а також фінальна сума до оплати.

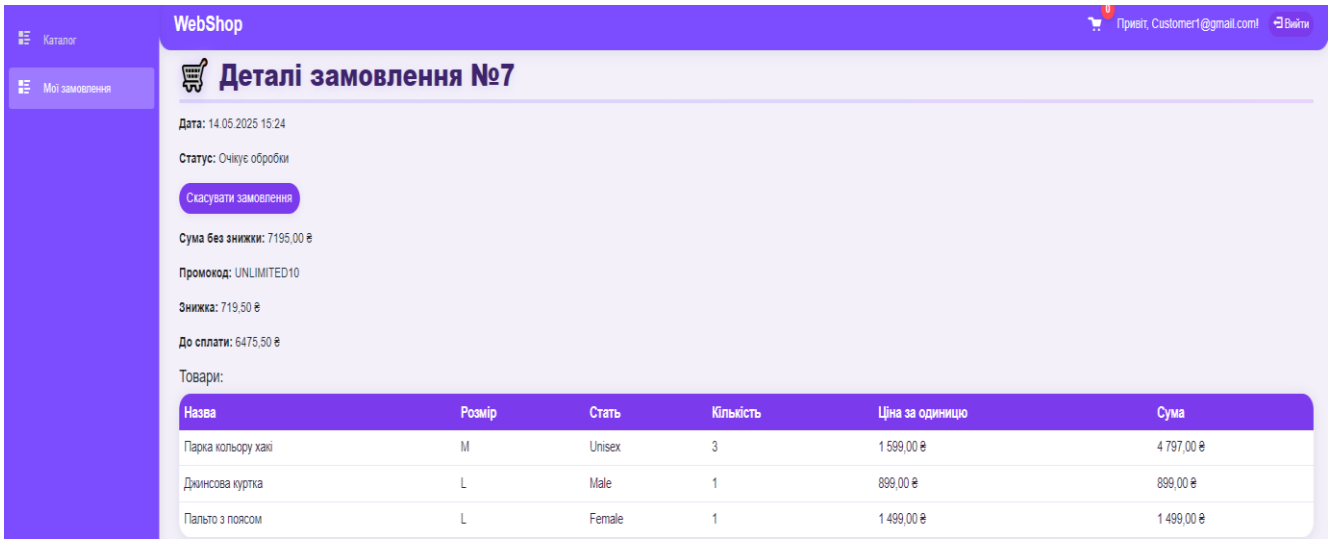

Рисунок 4.7 – Сторінка з деталями замовлення

У випадку, якщо статус замовлення ще не оброблено, користувач має можливість його скасувати. Після скасування система повертає товари на склад у відповідній кількості та змінює статус на «Скасовано».

Усі записи з історії замовлень зберігаються у базі даних та доступні користувачеві незалежно від кількості покупок чи тривалості використання системи.

4.2.4 Адміністративна панель: категорії та товари

Функціональність адміністративної панелі охоплює управління категоріями та товарами. Розділ категорій надає адміністратору змогу створювати, редагувати та приховувати категорії. Усі назви проходять перевірку на унікальність, а неактивні категорії не відображаються у клієнтському інтерфейсі. Візуальне представлення списку категорій забезпечує швидкий доступ до кожної з них (рисунок 4.8).

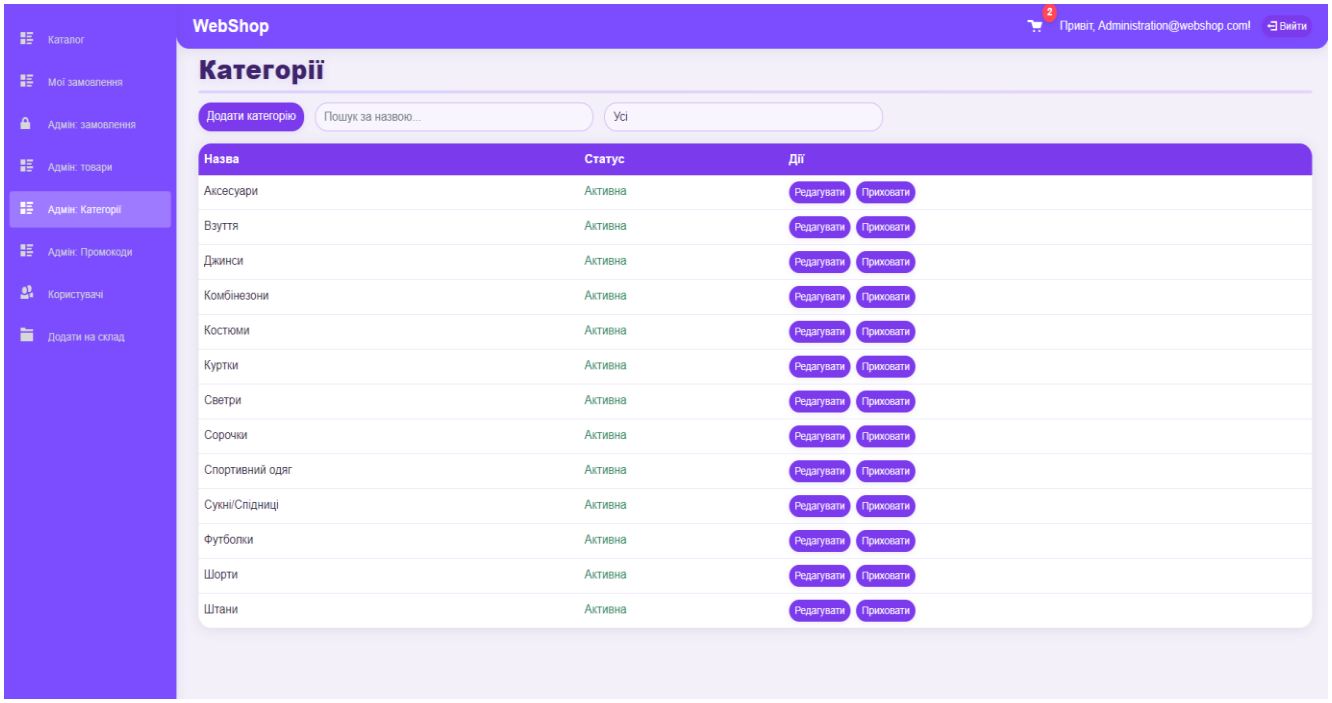

Рисунок 4.8 – Панель керування категоріями

Панель управління товарами реалізує повний набір CRUD-операцій: додавання, редагування, приховування товарів (рисунок 4.9). Під час створення товару вказується назва, опис, категорія, ціна, шаблон розмірів, стать. Додатково завантажуються зображення, визначається головне та порядок відображення. Присутні перевірки обов'язкових полів і повідомлення про помилки.

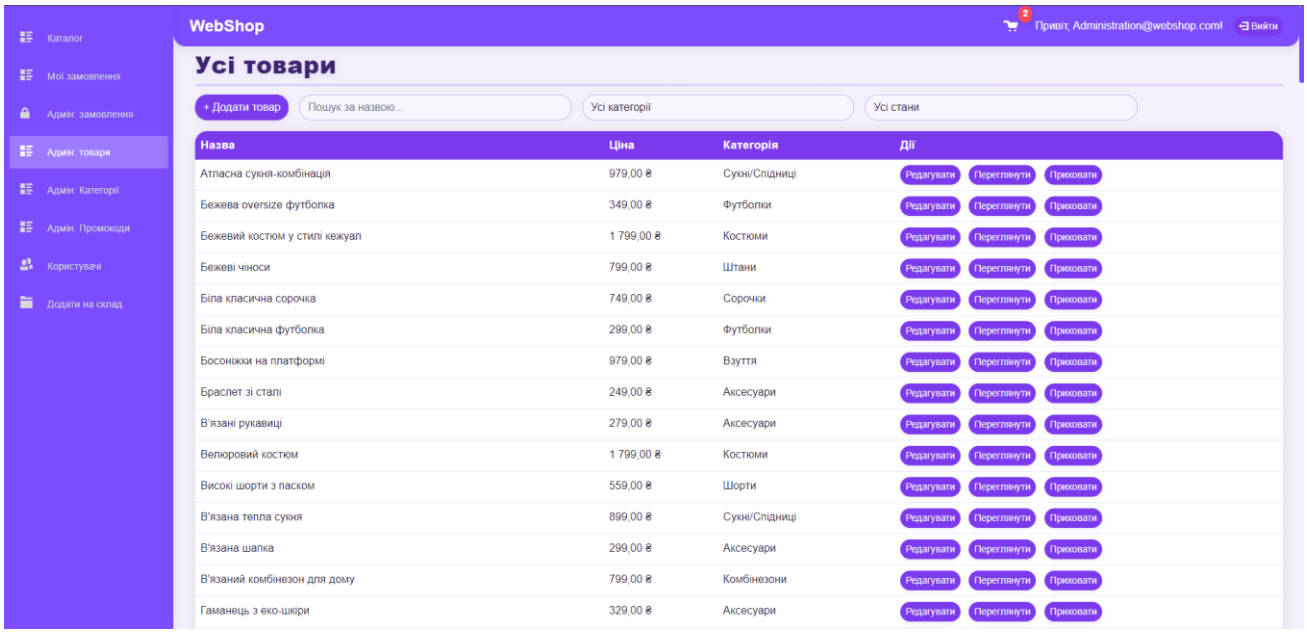

Рисунок 4.9 – Панель керування товарами

Реалізована підтримка м'якого видалення: товари та категорії не видаляються фізично, а лише приховуються за допомогою прапорця IsDeleted, що дозволяє зберігати історію замовлень та забезпечує узгодженість у базі.

4.2.5 Управління замовленнями в адмін-панелі

У розділі адміністрування замовлень передбачено повноцінний інтерфейс для перегляду всіх оформлених замовлень. Адміністратор має доступ до таблиці, де виводяться ідентифікатор, ім'я користувача, сума, дата створення та поточний статус. Передбачено пошук, сортування за датою та фільтрацію за статусом виконання (рисунок 4.10).

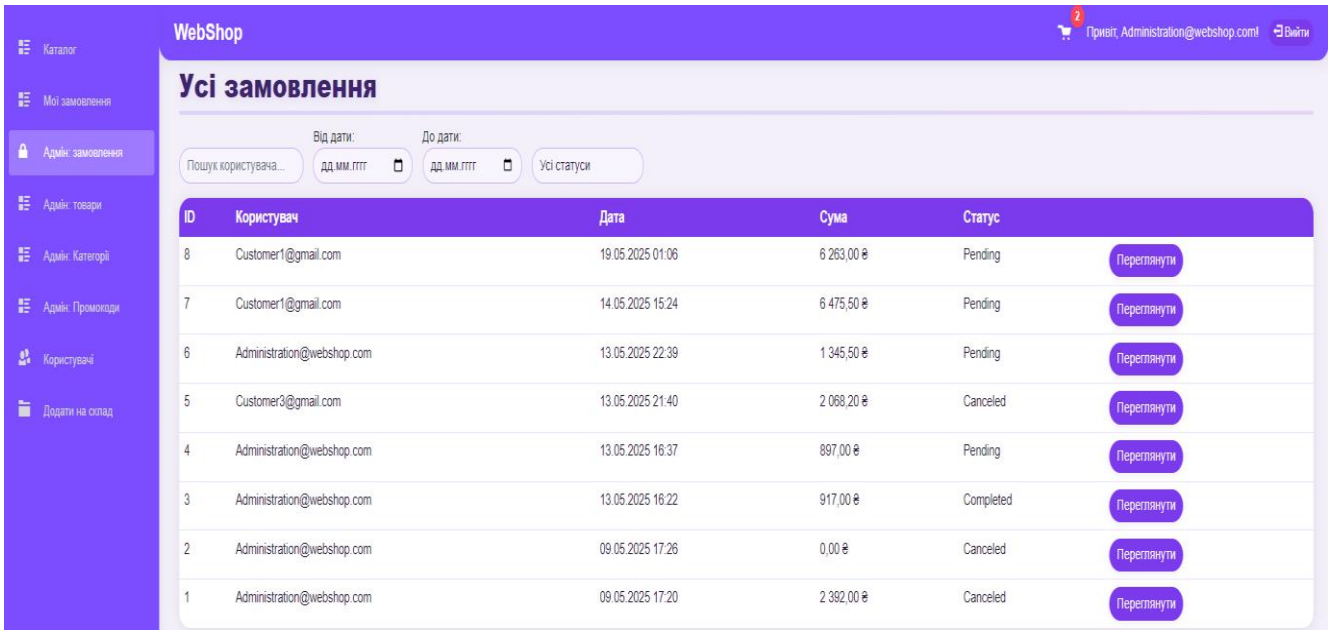

Рисунок 4.10 – Адмін-панель «Усі замовлення»

Детальний перегляд замовлення дозволяє бачити перелік позицій, їх розміри, кількість, ціни, суму без знижки, розмір знижки та кінцеву суму (рисунок 4.11). Для кожного замовлення адміністратор може змінити статус за визначеним сценарієм: «Очікує обробки» → «В обробці» → «Відправлено» → «Завершено» або «Скасовано».

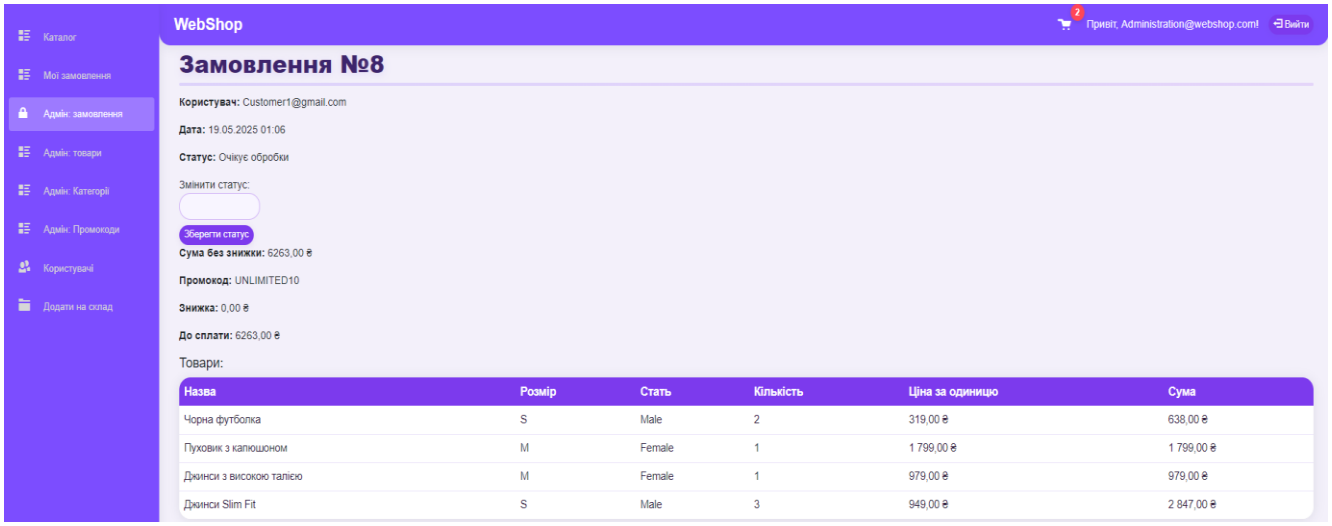

Рисунок 4.11 – Адмін-панель з деталями замовлення

У випадку скасування замовлення система автоматично повертає відповідну кількість товарів на склад, оновлюючи значення Quantity у сутності ProductVariant, що гарантує цілісність обліку та коректну роботу інвентаризації.

4.2.6 Управління промокодами

Функціональність адміністрування промокодів реалізована через окремий інтерфейс, який дає змогу створювати, редагувати, деактивувати або видаляти промокоди (рисунок 4.12). Для кожного коду визначається унікальний символьний ідентифікатор, відсоток знижки, строк дії, ліміт кількості використань або його відсутність.

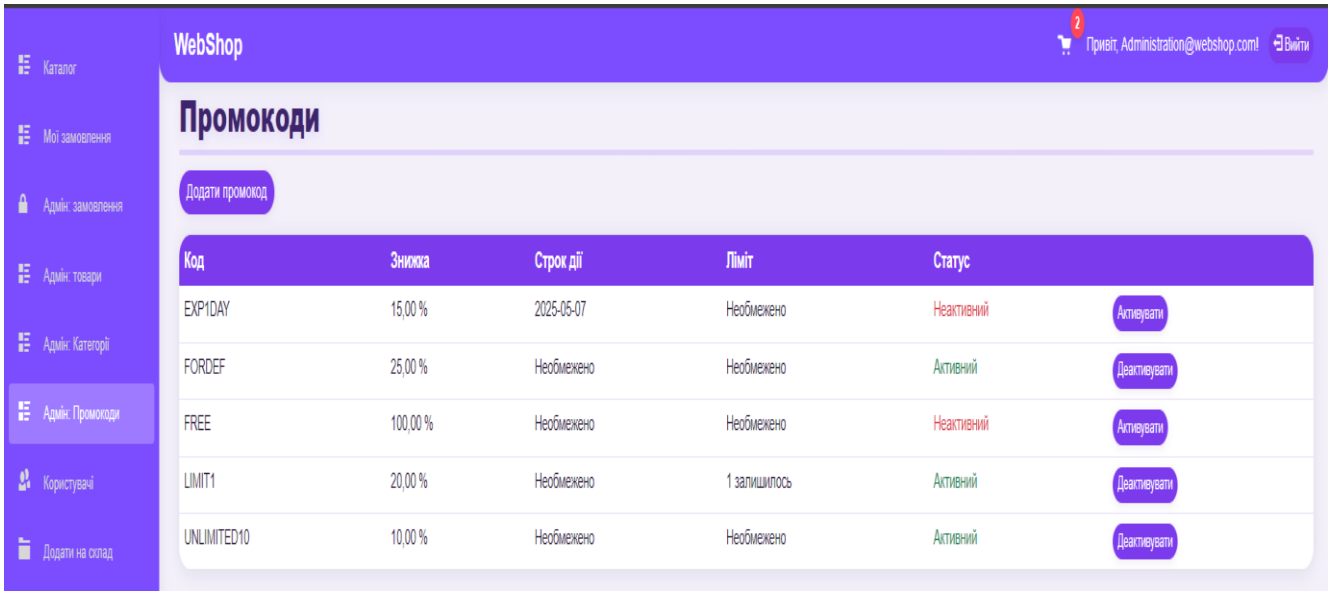

Рисунок 4.12 – Адмін-панель керуванням промокодами

Перед створенням коду система перевіряє його унікальність і коректність введених параметрів. Обов’язковою є валідація мінімальної довжини та допустимого значення знижки (1–100%). Після збереження код відображається в загальному списку з інформацією про статус активності, кількість використань і дату завершення дії.

При оформленні замовлення клієнтом усі активні промокоди проходять додаткову перевірку – зокрема, обмеження на кількість використань та заборону повторного використання одним користувачем, що гарантує контрольоване надання знижок та захист від зловживань.

4.2.7 Перегляд користувачів

У межах адміністративної частини реалізовано базовий функціонал перегляду зареєстрованих користувачів (рисунок 4.13). Виводиться список із вказанням електронної пошти.

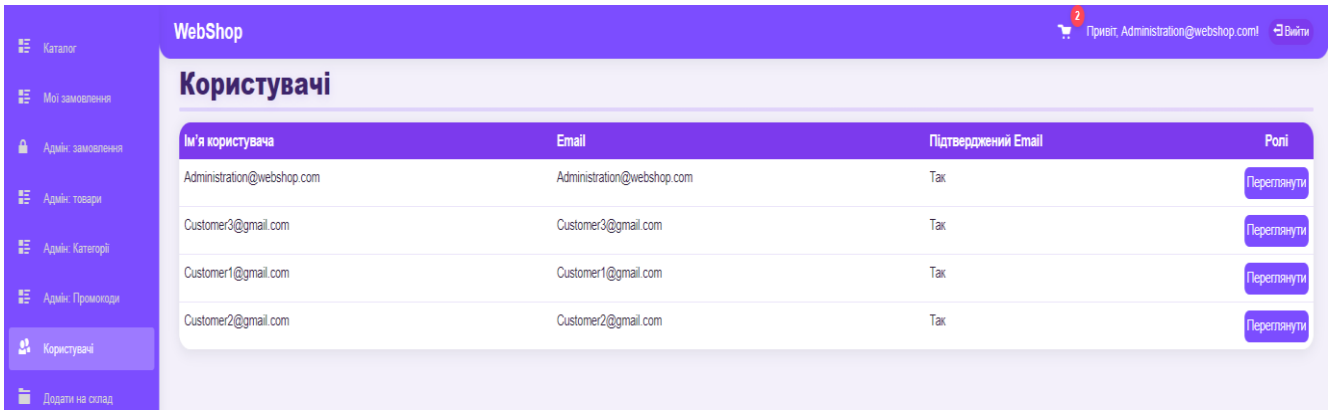

Рисунок 4.13 – Перелік користувачів в адмін-панелі

Даний функціонал дозволяє адміністратору мати загальне уявлення про активних користувачів системи, ідентифікувати їх рольову належність (звичайний користувач або адміністратор) та перевіряти наявність дублювань чи технічних облікових записів.

Функціонал редагування, блокування або присвоєння нових ролей у межах дипломної роботи не передбачений, оскільки основний акцент зроблено на реалізації механізмів замовлень і управління торговими даними.

4.2.8 Загальна навігація та зручність інтерфейсу

Інтерфейс системи реалізовано з урахуванням принципів доступності, логічної структури та послідовності взаємодії. Основна навігація представлена у вигляді верхнього та бокового меню, які адаптуються залежно від ролі користувача. У публічній частині користувач має доступ до каталогу товарів, кошика, форми замовлення та сторінки з історією покупок (рисунок 4.14). Оформлення побудовано таким чином, щоб забезпечити інтуїтивну взаємодію навіть для користувачів без досвіду роботи з подібними системами.

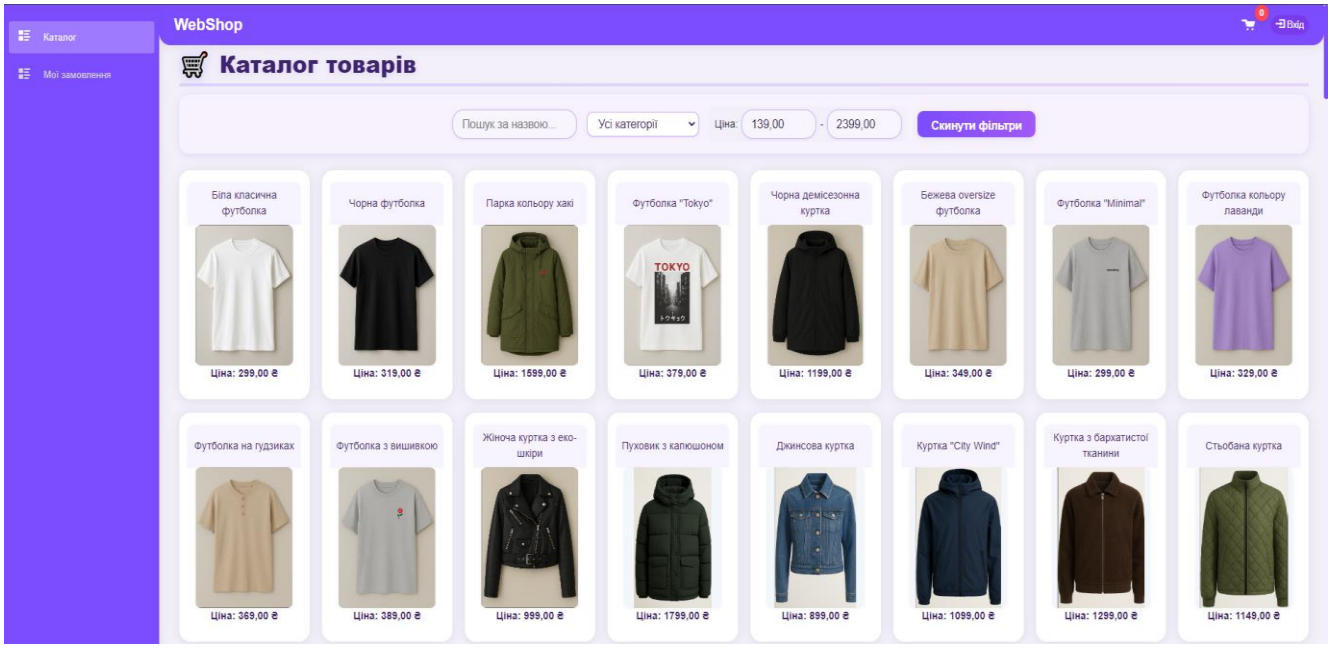

Рисунок 4.14 — Загальна навігація публічної частини

Для авторизованих адміністраторів додатково відкривається розділ адміністративного керування: категоріями, товарами, складом, замовленнями, користувачами та промокодами. Ліва панель містить структуроване меню з іконками та назвами відповідних функціональних блоків, що забезпечує швидкий перехід між розділами (рисунок 4.15). Візуальне розділення та іконографіка сприяють кращому сприйняттю функцій і зменшують час на пошук потрібного елементу.

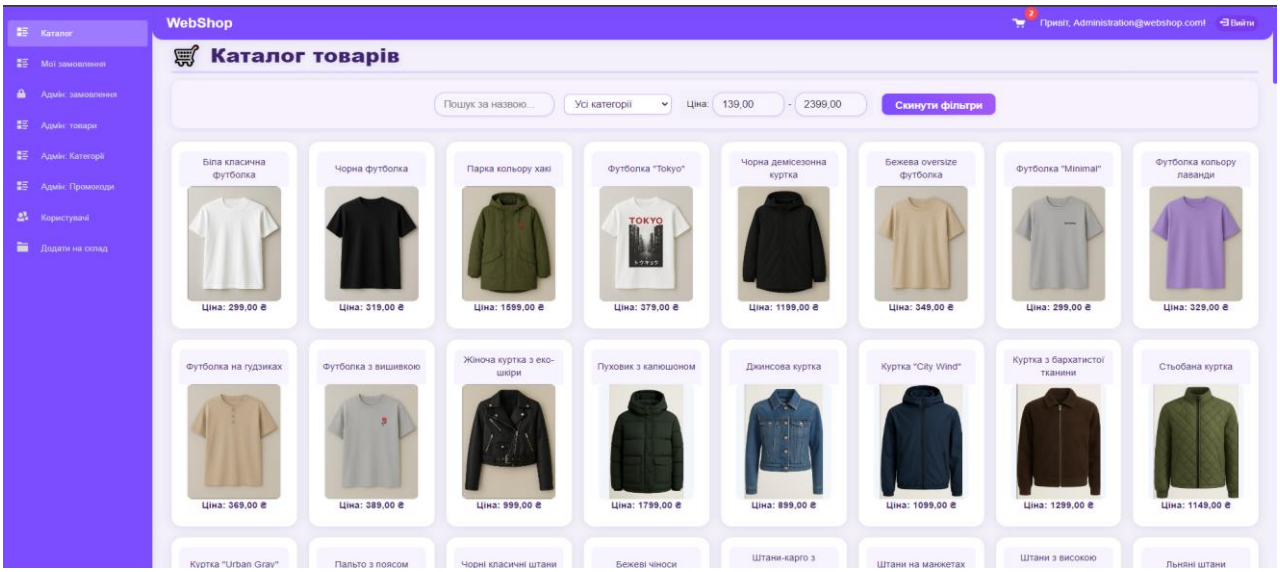

Рисунок 4.15 — Адміністративне меню

Повідомлення про дії системи (наприклад, застосування промокоду, помилки або підтвердження операцій) відображаються через інтегровану систему сповіщень.

4.3 Результати перевірки функціональності та відповідність вимогам

На етапі завершення розробки було здійснено перевірку реалізованого функціоналу відповідно до визначених у завданні вимог. Сценарії взаємодії охоплювали повний життєвий цикл замовлення: від перегляду товарів до фінального оформлення та керування на рівні адміністратора.

Всі основні компоненти системи працюють згідно з очікуваним функціональним поведінковим описом. Система забезпечує:

- перегляд і фільтрацію товарів за кількома критеріями;
- вибір розміру, облік кількості та оформлення замовлення;
- застосування та перевірку промокодів;
- ведення історії замовлень користувачем;

– керування вмістом системи через адміністративну панель.

Особливу увагу приділено узгодженості дій між клієнтською частиною та серверною логікою: після кожної операції дані коректно оновлюються, змінюється стан складу, обчислюється знижка та зберігається історія замовлення.

Візуальна структура інтерфейсу відповідає принципам сучасної розробки – усі елементи згруповано логічно, сповіщення та кнопки відповідають своїм діям, а користувач отримує чіткий зворотний зв’язок.

Тестування показало, що система є стабільною, масштабованою та придатною до розширення. Виявлені під час перевірки незначні технічні помилки було оперативно виправлено. Таким чином, реалізація інформаційної системи повністю відповідає сформульованим технічним та функціональним вимогам.

ВИСНОВКИ

На основі аналізу підходів до автоматизації замовлень у сфері дистрибуції споживчих товарів, а також оцінки існуючих систем електронної комерції (зокрема, OpenCart, WooCommerce, Prom.ua), обґрунтовано доцільність розробки власного рішення із гнучкою архітектурою, повним контролем над функціональністю та можливістю розширення. Обрано підхід із клієнт-серверною взаємодією, що передбачає незалежність інтерфейсу від внутрішньої логіки обробки замовлень.

За результатами аналізу технологічного стеку обґрунтовано використання таких програмних засобів: фреймворк Blazor Server для реалізації інтерфейсу з інтерактивною логікою; Entity Framework Core для об'єктно-реляційного доступу до бази даних; Microsoft SQL Server як надійної системи управління базами даних; бібліотеки Sortable.js для drag & drop сортування зображень; засобів авторизації ASP.NET Identity для керування користувачами та ролями.

Розроблено інформаційну систему управління замовленнями споживачів у дистрибуції одягу та відповідних аксесуарів, яка включає: модуль обробки замовлень із підтримкою варіантів товарів, складського обліку, застосування промокодів; адміністративну панель для керування товарами, категоріями, залишками, промокодами й замовленнями; механізми перевірки цілісності даних, адаптивну навігацію, фільтрацію та відображення історії замовлень.

У результаті проведеного тестування підтверджено коректність реалізації основного функціоналу системи. Верифіковано стабільність роботи кошика, логіку розрахунку знижки, облік залишків на складі, реакцію системи на скасування замовлень, а також обмеження щодо повторного використання промокодів. Перевірка інтерфейсної частини підтвердила відповідність сучасним вимогам зручності та доступності.

Серед напрямів можливого вдосконалення системи можна виокремити: реалізацію повноцінного управління користувачами з розширеними ролями; інтеграцію з платіжними сервісами для прийому онлайн-оплат.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. SendPulse. CMS – що це таке? URL: https://sendpulse.ua/support/glossary/cms
2. Shopify – офіційний сайт платформи. URL: https://www.shopify.com/fi
3. OpenCart – основні функції. URL: https://www.opencart.com/index.php?route=cms/feature
4. WooCommerce – офіційна документація. URL: https://woocommerce.com/
5. Prom.ua – платформа для онлайн-торгівлі. URL: https://prom.ua/
6. Rozetka – довідка продавця. URL: https://sellerhelp.rozetka.com.ua/
7. Демоверсія OpenCart. URL: https://demo.opencart.ua/index.php?route=common/home
8. Microsoft Docs. C# Programming Language. URL: https://learn.microsoft.com/en-us/dotnet/csharp/
9. Microsoft Docs. Blazor Server. URL: https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-9.0#blazor-server
10. Microsoft Docs. ASP.NET Core Documentation. URL: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-9.0
11. Microsoft Docs. SignalR Introduction. URL: https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-9.0
12. Microsoft Docs. What is SQL Server? URL: https://learn.microsoft.com/en-us/sql/sql-server/what-is-sql-server?view=sql-server-ver17
13. Microsoft Docs. Entity Framework Core. URL: https://learn.microsoft.com/en-us/ef/core/
14. Microsoft Docs. DbContext (EF 6). URL: https://learn.microsoft.com/en-us/dotnet/api/system.data.entity.dbcontext?view=entity-framework-6.2.0
15. Microsoft Docs. C# features that support LINQ. URL: https://learn.microsoft.com/en-us/dotnet/csharp/linq/get-started/features-that-support-linq

16. Microsoft Docs. Asynchronous programming scenarios in C#. URL: https://learn.microsoft.com/en-us/dotnet/csharp/asynchronous-programming/async-scenarios

17. Visual Studio – офіційний сайт. URL: https://visualstudio.microsoft.com/vs/

18. Microsoft Docs. NuGet – встановлення пакетів. URL: https://learn.microsoft.com/en-us/nuget/consume-packages/install-use-packages-visual-studio

19. Git – Branching and Merging. URL: https://git-scm.com/about/branching-and-merging

20. GitHub – інформація про платформу. URL: https://github.com/about

21. MDN Web Docs. HTML: HyperText Markup Language. URL: https://developer.mozilla.org/en-US/docs/Web/HTML

22. MDN Web Docs. CSS: Cascading Style Sheets. URL: https://developer.mozilla.org/en-US/docs/Web/CSS

23. Sortable.js – офіційна документація. URL: https://sortablejs.github.io/Sortable/

24. StackOverflow. Are soft deletes a good idea? URL: https://stackoverflow.com/questions/2549839/are-soft-deletes-a-good-idea

ДОДАТОК А

Діаграма прецедентів для системи управління замовленнями

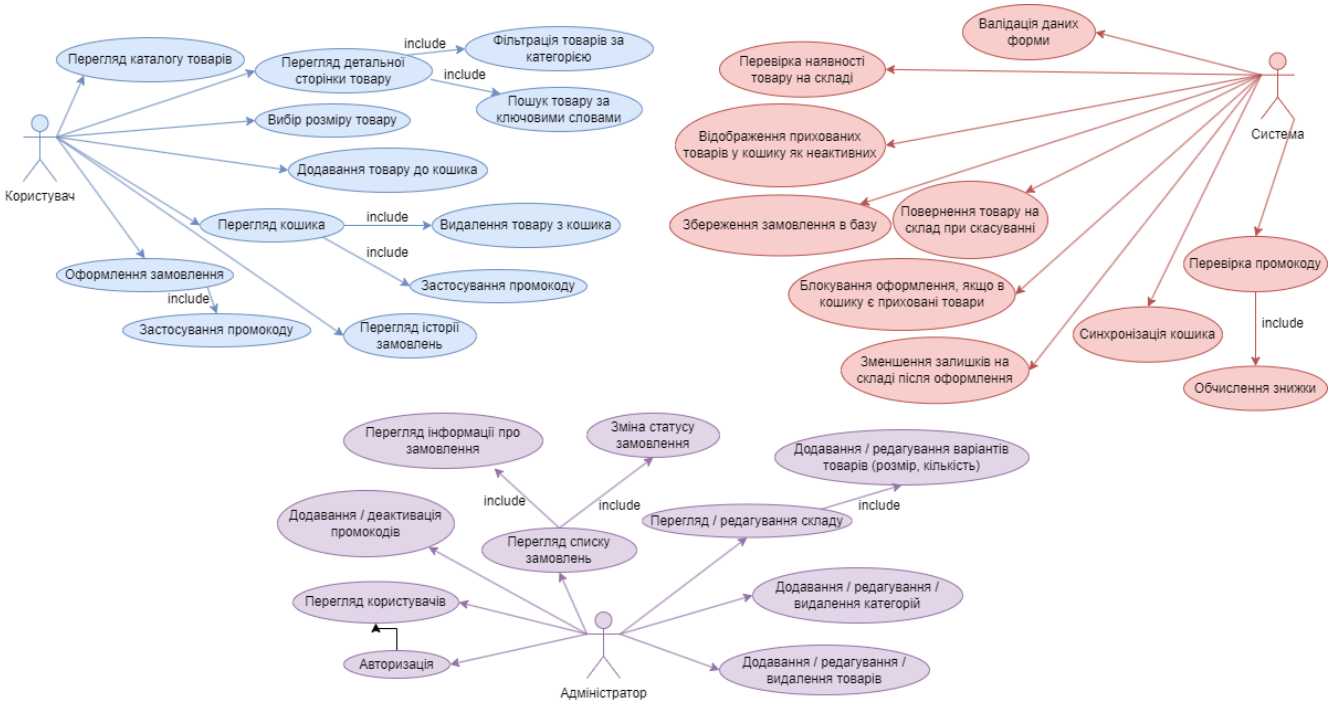

Рисунок А1 — Діаграма прецедентів

ДОДАТОК Б

Реалізація сервісу обробки замовлень

Програмні засоби:

- Мова програмування – C#
- фреймворк – .NET 7 з використанням Entity Framework Core;
- доступ до бази даних – IDbContextFactory<T> та async/await;
- логіка обробки – LINQ;

```
using WebShop.Models;
using Microsoft.EntityFrameworkCore;
using WebShop.Data;

namespace WebShop.Services
{
   public class CartService
   {
      private readonly IDbContextFactory<ApplicationDbContext> _dbFactory;

      public CartService(IDbContextFactory<ApplicationDbContext> dbFactory) => _dbFactory = dbFactory;

      // Додає товар до кошика користувача або збільшує кількість, якщо товар вже є в кошику
      public async Task AddToCartAsync(string userId, int productVariantId, int quantity = 1)
      {
         // Створюємо екземпляр контексту бази даних (ізоляція для потокобезпеки)
         using var db = await _dbFactory.CreateDbContextAsync();
```

```
            // Перевіряємо, чи такий товар уже є в кошику користувача
            var existingItem = await db.CartItems
                .FirstOrDefaultAsync(c => c.UserId == userId && c.ProductVariantId
== productVariantId);

            if (existingItem != null)
            {
                // Якщо є — збільшуємо кількість
                existingItem.Quantity += quantity;
            }
            else
            {
                // Якщо немає — створюємо новий запис кошика
                db.CartItems.Add(new CartItem
                {
                    UserId = userId,
                    ProductVariantId = productVariantId,
                    Quantity = quantity
                });
            }

            await db.SaveChangesAsync();
        }

        // Повертає повний список товарів у кошику користувача
        public async Task<List<CartItem>> GetItemsAsync(string userId)
        {
            using var db = await _dbFactory.CreateDbContextAsync();
            return await db.CartItems
```

```
            .Include(ci => ci.ProductVariant)
            .ThenInclude(pv => pv.Product)
            .ThenInclude(p => p.Category)
            .IgnoreQueryFilters()
            .Where(ci => ci.UserId == userId)
            .ToListAsync();
        }

        // Обчислює загальну суму вартості товарів у кошику користувача без знижок
        public async Task<decimal> GetTotalAsync(string userId)
        {
            using var db = await _dbFactory.CreateDbContextAsync();
            var items = await db.CartItems
                .Include(i => i.ProductVariant).ThenInclude(v => v.Product)
                .Where(i => i.UserId == userId)
                .ToListAsync();

            return items.Sum(i => i.ProductVariant.Product.Price * i.Quantity);
        }

        // Видаляє конкретний товар із кошика користувача
        public async Task RemoveFromCartAsync(string userId, int productVariantId)
        {
            using var db = await _dbFactory.CreateDbContextAsync();
            var item = await db.CartItems
                .FirstOrDefaultAsync(c => c.UserId == userId && c.ProductVariantId == productVariantId);
```

```
            if (item != null)
            {
                db.CartItems.Remove(item);
                await db.SaveChangesAsync();
            }
        }

        // Повністю очищає кошик користувача
        public async Task ClearCartAsync(string userId)
        {
            using var db = await _dbFactory.CreateDbContextAsync();
            var items = await db.CartItems
                .Where(c => c.UserId == userId)
                .ToListAsync();

            db.CartItems.RemoveRange(items);
            await db.SaveChangesAsync();
        }

        // Оформлює замовлення для користувача з урахуванням промокоду та перевіркою всіх умов
        public async Task<PlaceOrderResult> PlaceOrderAsync(string userId, string promoCode)
        {
            using var db = await _dbFactory.CreateDbContextAsync();

            // Завантажуємо всі позиції кошика з повною інформацією про товар, розмір і категорію
            var items = await db.CartItems
                .Include(ci => ci.ProductVariant)
```

```
            .ThenInclude(pv => pv.Product)
            .ThenInclude(p => p.Category)
            .IgnoreQueryFilters()
            .Where(c => c.UserId == userId)
            .ToListAsync();

        if (!items.Any())
        {
            return new PlaceOrderResult
            {
                IsSuccess = false,
                ErrorMessage = "Кошик порожній"
            };
        }

        // Якщо у кошику є товари або категорії, що приховані (IsDeleted), не дозволяємо оформлення
        if (items.Any(i => i.ProductVariant.Product.IsDeleted || i.ProductVariant.Product.Category.IsDeleted))
        {
            return new PlaceOrderResult
            {
                IsSuccess = false,
                ErrorMessage = "У кошику є товари, які більше недоступні"
            };
        }

        // Перевірка наявності кожного товару
        List<string> missing = new();
```

```
            foreach (var item in items)
            {
                if (item.ProductVariant.Quantity < item.Quantity)
                    missing.Add($"{item.ProductVariant.Product.Title}
({item.ProductVariant.Size})");
            }

            // Якщо деяких товарів не вистачає — повертаємо помилку з їх
переліком
            if (missing.Any())
            {
                return new PlaceOrderResult
                {
                    IsSuccess = false,
                    ErrorMessage = "Не вистачає деяких товарів на складі",
                    MissingItems = missing
                };
            }

            // Ініціалізуємо знижку
            decimal discountPercent = 0;
            decimal discountAmount = 0;
            decimal total = items.Sum(i => i.ProductVariant.Product.Price *
i.Quantity);

            // Якщо введено промокод — перевіряємо його валідність і умови
            if (!string.IsNullOrWhiteSpace(promoCode))
            {
                var promo = await db.PromoCodes
                    .AsNoTracking()
```

```
                .FirstOrDefaultAsync(p => p.Code.ToUpper() == promoCode.ToUpper() && p.IsActive &&
                    (p.ExpirationDate == null || p.ExpirationDate > DateTime.Now));

            if (promo != null)
            {
                discountPercent = promo.DiscountPercent;

                // Якщо є ліміт використань — перевіряємо, чи не перевищено
                if (promo.UsageLimit != null)
                {
                    var usedCount = await db.PromoCodeUsages.CountAsync(u => u.PromoCodeId == promo.Id);
                    if (usedCount >= promo.UsageLimit.Value)
                        discountPercent = 0;
                }

                // Якщо користувач вже використовував цей код — скидка не застосовується повторно
                bool alreadyUsed = await db.PromoCodeUsages
                    .AnyAsync(u => u.UserId == userId && u.PromoCodeId == promo.Id);

                if (alreadyUsed)
                    discountPercent = 0;

                discountAmount = total * discountPercent / 100m;
            }
        }
```

```
            var finalAmount = total - discountAmount;

            var order = new Order
            {
                UserId = userId,
                CreatedAt = DateTime.Now,
                Status = OrderStatus.Pending,
                PromoCode = !string.IsNullOrWhiteSpace(promoCode) ? promoCode :
null,
                TotalAmount = finalAmount,
                OriginalAmount = total,
                DiscountAmount = discountAmount,
                Items = items.Select(i => new OrderItem
                {
                    ProductVariantId = i.ProductVariantId,
                    Quantity = i.Quantity,
                    PriceAtPurchase = i.ProductVariant.Product.Price
                }).ToList()
            };

            // Зменшуємо залишки товарів на складі
            foreach (var item in items)
                item.ProductVariant.Quantity -= item.Quantity;

            db.Orders.Add(order);
            db.CartItems.RemoveRange(items);
            await db.SaveChangesAsync();

            // Якщо промокод застосовано — фіксуємо факт використання
            if (!string.IsNullOrWhiteSpace(promoCode) && discountPercent > 0)
```

```
            {
                var promo = await db.PromoCodes
                    .FirstOrDefaultAsync(p => p.Code.ToUpper() == promoCode.ToUpper() && p.IsActive);

                if (promo != null)
                {
                    db.PromoCodeUsages.Add(new PromoCodeUsage
                    {
                        UserId = userId,
                        PromoCodeId = promo.Id,
                        UsedAt = DateTime.Now
                    });
                    await db.SaveChangesAsync();
                }
            }

            return new PlaceOrderResult { IsSuccess = true };
        }

        // Обчислює підсумкову суму в кошику з урахуванням знижки та повертає інформацію про застосування промокоду
        public async Task<(decimal Total, decimal DiscountAmount, decimal FinalTotal, bool PromoUsed)> CalculateTotalWithDiscountAsync(string userId, string promoCode)
        {
            using var db = await _dbFactory.CreateDbContextAsync();

            var items = await db.CartItems
                .Include(ci => ci.ProductVariant)
```

```
            .ThenInclude(pv => pv.Product)
            .ThenInclude(p => p.Category)
            .Where(i => i.UserId == userId)
            .ToListAsync();

        decimal total = items.Sum(i => i.ProductVariant.Product.Price * i.Quantity);
        decimal discountAmount = 0;
        bool promoCodeAlreadyUsed = false;

        if (!string.IsNullOrWhiteSpace(promoCode))
        {
            var promo = await db.PromoCodes
                .FirstOrDefaultAsync(p => p.Code.ToUpper() == promoCode.ToUpper() && p.IsActive);

            if (promo != null)
            {
                // Якщо промокод прострочено — повертаємо суму без знижки
                if (promo.ExpirationDate != null && promo.ExpirationDate <= DateTime.Now)
                    return (total, 0, total, false);

                // Якщо перевищено ліміт використань — без знижки
                if (promo.UsageLimit != null)
                {
                    var usedCount = await db.PromoCodeUsages.CountAsync(u => u.PromoCodeId == promo.Id);
                    if (usedCount >= promo.UsageLimit.Value)
                        return (total, 0, total, false);
```

```
                }

                // Перевіряємо, чи вже використовував цей промокод цей користувач
                promoCodeAlreadyUsed = await db.PromoCodeUsages
                    .AnyAsync(u => u.UserId == userId && u.PromoCodeId == promo.Id);

                // Якщо промокод ще не використовувався — обчислюємо знижку
                if (!promoCodeAlreadyUsed)
                    discountAmount = total * promo.DiscountPercent / 100m;
            }
        }

        return (total, discountAmount, total - discountAmount, promoCodeAlreadyUsed);
    }

    // Змінює кількість конкретного товару в кошику користувача
    public async Task UpdateQuantityAsync(string userId, int productVariantId, int delta)
    {
        using var db = await _dbFactory.CreateDbContextAsync();
        var item = await db.CartItems
            .FirstOrDefaultAsync(c => c.UserId == userId && c.ProductVariantId == productVariantId);

        if (item != null)
        {
            item.Quantity += delta;
```

```
                if (item.Quantity <= 0)
                    db.CartItems.Remove(item);

                await db.SaveChangesAsync();
            }
        }

        // Повертає загальну кількість товарів у кошику користувача (сума всіх одиниць)
        public async Task<int> GetCartCountAsync(string userId)
        {
            using var db = await _dbFactory.CreateDbContextAsync();
            return await db.CartItems.Where(c => c.UserId == userId).SumAsync(c => c.Quantity);
        }
    }
}
```